实时音视频

无 UI 集成

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

版权所有：腾讯云计算（北京）有限责任公司

无 UI 集成
概述

最近更新时间：2025-05-22 16:56:12

无 UI 集成目录下的文档主要是腾讯实时音视频（Tencent Real-Time Communication，TRTC）SDK 和 RTC Room Engine SDK 的接入和介绍文档。
TRTC SDK 将腾讯多年来在网络与音视频技术上的深度积累，以多人音视频通话和低延时互动直播两大场景化方案，通过腾讯云服务向开发者开放，致力于帮助开发者快速搭建低成本、低延时、高品质的音视频互动解决方案。快速接入TRTC SDK 。

> **注意：**
> 目前 TRTC SDK 仅提供媒体服务相关的能力和 API，如果您想使用 组件介绍（TUIRoomKit） 或 组件介绍（TUILiveKit） 产品，想要基于 TUIRoomKit或 TUILiveKit 的能力通过自定义 UI 来实现对应的业务，又或者您的业务期望能快速实现有类似媒体控制（例如禁言、禁画），麦位管理（例如上下麦、锁麦），以及房间成员管理（邀请、移人）等能力支持，您可以尝试接入我们的 RTC Room Engine SDK 实现。

RTC Room Engine SDK 是 多人会议（RoomKit） 和 直播与语聊房（TUILiveKit） 共同使用的软件开发工具包。它支持房间管理、屏幕分享、成员管理、麦位控制、基础美颜等丰富功能，同时确保 720P 和 1080P 高清画质，在70%丢包率的弱网环境下仍能保持高质量音视频传输。此外，SDK 采用 48kHz 高音质采样，结合腾讯天籁实验室的 3A 处理算法，消除回声和啸叫，实现全链路 128kbps 高音质立体声，为用户带来清晰沉浸的互动体验。快速接入RoomEngine SDK 。

接入音视频引擎（TRTC-SDK）
SDK 下载
SDK 下载

最近更新时间：2025-05-14 14:39:02

下载 SDK

实时音视频（TRTC）是腾讯云提供的一套低延时、高质量的音视频通讯服务，致力于为腾讯云客户提供稳定、可靠和低成本的音视频传输能力。该服务由一套遍布全球的音视频传输网络和一组终端 SDK 组成，您可以在本页面下载到涵盖目前主流客户端平台和热门框架的 实时音视频（TRTC）SDK。
实时音视频TRTC SDK 提供方为深圳市腾讯计算机系统有限公司，其个人信息处理规则见 实时音视频 TRTC SDK 个人信息保护规则，当前可下载版本为 12.5，发布日志见 发布日志（App），合规使用说明见 合规使用指南 。

说明：
- 如果您当前网络访问 Github 速度不理想，可以 单击这里 访问腾讯云专属的加速仓库。
- 若您在下载新版 SDK 后，请在上线前进行必要的功能验证测试，以免原有功能表现与老版本不一致导致产生集成问题。

Web SDK

TRTC Web SDK

包含 TRTC 功能，无需安装 App 即可音视频通话，兼容主流的桌面和移动端浏览器。

NPM　ZIP 下载　GitHub　集成指引　运行 Demo

Android SDK

精简版（TRTC）SDK

包含 TRTC 和直播播放（TXLivePlayer）两项功能，适配主流模拟器和声卡，SDK 体积小巧，功能稳定。

Gradle　ZIP 下载　GitHub　集成指引　运行 Demo

全功能版（Professional）SDK

包含 TRTC、直播、短视频、点播等多项功能，功能丰富，SDK 体积较精简版略大。

Gradle ZIP 下载 集成指引 运行 Demo

iOS SDK

精简版（TRTC）SDK

包含 TRTC 和直播播放（TXLivePlayer）两项功能，SDK 体积小巧，功能稳定。

ZIP 下载 GitHub 集成指引 运行 Demo

全功能版（Professional）SDK

包含 TRTC、直播、短视频、点播等多项功能，功能丰富，SDK 体积较精简版略大。

ZIP 下载 集成指引 运行 Demo

HarmonyOS SDK

说明：
鸿蒙 SDK 现已发布，该版本已完成基础功能验证，现优先对部分企业级付费客户开放试用申请，如需使用，请 填写表单 联系您的商务。

Windows SDK

Windows SDK（C++ 版）

包含实时音视频（TRTC）、直播推流（TXLivePusher）、直播播放（TXLivePlayer）和点播播放（TXVodPlayer）等四项功能

ZIP 下载 GitHub 集成指引 运行 Demo

Windows SDK（C# 版）

包含实时音视频（TRTC）、直播推流（TXLivePusher）、直播播放（TXLivePlayer）和点播播放（TXVodPlayer）等四项功能。

ZIP 下载 GitHub 集成指引 运行 Demo

Windows SDK（ActiveX 版）

基于 ActiveX 插件封装的 TRTC SDK，让您在 ActiveX 场景下快速便捷的集成实时音视频服务。

ZIP 下载 GitHub 集成指引 运行 Demo

Mac OS SDK

Mac OS SDK

包含实时音视频（TRTC）、直播推流（TXLivePusher）和播放器（TXLivePlayer）等功能。

ZIP 下载 GitHub 集成指引 运行 Demo

微信小程序 SDK

微信小程序 SDK

以 live-pusher 和 live-player 两个标签封装的 TRTC SDK，体积小巧，无需安装，效果媲美 App。

ZIP 下载 GitHub 集成指引 运行 Demo

跨平台 SDK

Electron SDK

基于 Electron 框架封装，让您基于 Web 技术快速构建 Windows 和 Mac 平台上的应用。

ZIP 下载 GitHub 集成指引 运行 Demo

Flutter SDK

基于 Flutter 框架封装的 TRTC SDK，让您用一套代码快速构建出能够运行于各平台的 App。

ZIP 下载 GitHub 集成指引 运行 Demo

uni-app SDK

基于 uni-app 插件封装的 TRTC SDK，让您快速便捷集成实时音视频服务。

GitHub 运行 Demo

发布日志（App）

最近更新时间：2025-06-04 15:16:41

此页面仅更新 TRTC SDK 的版本历史，如您想了解全功能版 SDK 的版本历史，请参见 全功能版 SDK 版本历史 。

> **说明：**
> 全功能版 SDK 是多个基础 SDK 的集合，它包含了直播、短视频、音视频通话和播放器等子产品 SDK 的功能模块。

> **注意：**
> 若 Android 项目 targetSdkVersion 为31或者目标设备涉及到 Android 12及更高系统版本，官方要求需要在代码中动态申请 `android.permission.BLUETOOTH_CONNECT` 权限，以正常使用蓝牙功能，具体信息请参见 Android 官方说明 。

Version 12.5 @ 2025.5.12

功能优化：

- 全平台：降低大块内存频繁创建，低内存场景下crash率降低50%。
- Android：优化立体声音质效果。
- iOS：提升声卡识别率。

缺陷修复：

- Android：修复部分设备屏幕分享，调整横竖屏分辨率产生的黑屏问题。
- Android： 修复通过 Activity 设置横屏本地预览显示异常的问题。
- Windows：修复 D3D11 像素格式转换失败导致黑屏的问题。

接口行为调整：

1. setLocalRenderMode 接口行为调整。
   - VIDEO_MIRROR_MODE_AUTO (0)：自动模式：如果正使用前置摄像头则本地与远端默认开启镜像，如果是后置摄像头则不开启本地与远端镜像（仅适用于移动设备）。
   - VIDEO_MIRROR_MODE_ENABLED (1)：强制开启镜像，不论当前使用的是前置摄像头还是后置摄像头，此时本地与远端都有镜像效果。
   - VIDEO_MIRROR_MODE_DISABLED (2)：强制关闭镜像，不论当前使用的是前置摄像头还是后置摄像头，此时本地与远端都没有镜像效果。
2. setVideoEncoderMirror 不建议再使用，如果调用也只影响远端画面，不再影响本地画面。
3. 自定义采集和屏幕分享场景下不再响应 setVideoEncoderMirror 和 setVideoEncoderRotation 接口。

Version 12.4 @ 2025.4.01

新特性：

合图支持设置边框颜色。

功能优化：

- 全平台：优化 TRTCVolumeInfo 状态中 vad 检测准确性。
- Android ： 支持低功耗蓝牙链接。
- macOS：全屏幕采集过滤高亮边框。

缺陷修复：

- Android：修复部分机型在开麦克风后切换耳机时，出现外放的情况。
- 全平台：修复感冒音效无声的问题。
- Android & iOS：摄像头上长时间不出帧检测策略实现。

接口行为调整：

1. 自定义预处理格式匹配逻辑调整
   - TRTCVideoPixelFormat_Texture_2D格式内部映射为RGBA32。
   - TRTCVideoBufferType_Texture只支持和TRTCVideoPixelFormat_Texture_2D组合。
2. iOS 平台屏幕共享增加因系统操作暂停与恢复的原因
   - 屏幕共享开启后，点击系统关闭屏幕录制按钮后，会回调 onScreenCapturePaused事件，并且回调的 reason 值为-4。
   - 屏幕共享开启后，点击系统开启屏幕录制按钮后，会回调 onScreenCaptureResumed事件，并且回调的 reason 值为-4。

Version 12.3 @ 2025.1.22

新特性：

Android：

- 将屏幕分享功能单独拆分成一个独立的 aar，aar 名为 LiteAVSDK_VideoScreenCapture。
- 如果是通过本地 aar 的方式集成，需要将新的 aar 同步集成到工程中。
- 如果是通过 maven 的方式集成，无需改动即可自动依赖下载。若您不需要屏幕分享功能，请在 gradle 配置文件中，通过 exclude 的方式把 LiteAVSDK_VideoScreenCapture 移除掉。

功能优化：

- Windows：下行开启自定义渲染模式时的解码性能优化。
- Windows：多显卡选择策略优化。
- Android：采集增加不出帧检测重启策略。
- 全平台：优化音量100 ~ 150区间增益上限从8dB调整到12dB。
- 全平台：优化纯音频场景下行弱网抗性。
- 全平台：优化 KTV 场景人声与 BGM 混合对齐效果。

缺陷修复：

- iOS：优化视频硬件编码卡死问题。
- Mac：修复麦克风开关导致 Loopback 无声问题。
- Mac：会自动切到新接入的耳机的问题。
- Windows：修复 win7 切换麦克风卡死问题。
- 全平台：修复 bgm 音量极低情况下，远端听 bgm 卡顿问题。

Version 12.2 @ 2024.11.27

新特性：

- Android：支持 16KB page sizes。
- 全平台：实时探测链路质量并自动切换更优的链路的功能。

功能优化：

- Android：硬件耳返支持更多手机品牌：小米、oppo、一加、荣耀。
- iOS：优化自动曝光下的对焦效果。
- iOS：优化视频画面缩放后的画质效果。

- 全平台：优化 Speech 音质下的声音效果。

缺陷修复：

- 修复 MaxOS 15 audio loopback 采集失败问题。
- 修复 Android 设备屏幕采集授权窗口弹出失败问题。
- 修复 Windows 硬件编码器的兼容性问题。

接口行为调整：

- iOS&Android TRTCCloud::generateCustomPTS 接口调整为静态接口。
- Android、iOS 以及 Windows 对外渲染画面调整模式增加 TRTCVideoFillMode_ScaleFill 模式，详见 TRTCVideoFillMode。

Version 12.1 @ 2024.10.08

新特性：

- Windows： Intel 显卡支持 H.265 编码和 B 帧编码。
- Windows： Nvida 显卡支持 ROI（区域兴趣）。

功能优化：

- 全平台：优化降噪和回声消除算法，提升音质听感。
- 全平台：优化 AGC（自动增益控制）音量曲线，预防爆音。
- 全平台：优化纯视频场景弱网抗性。
- Android：优化设备初始化参数，大幅提升音频采集稳定性。
- iOS：提升通话音量音质听感。

缺陷修复：

- Android： 自定义采集的视频帧旋转角度不生效问题。
- 修复 Android 14 屏幕分享偶现失败问题。
- 修复 Android 设备采集失败黑屏问题。
- 修复 Windows 系统混音双声道效果失效问题。
- 修复 Windows 特定设备采集无声问题。
- 修复 iOS/Mac 蓝牙耳机下开耳返出现杂音问题。

Version 12.0 @ 2024.8.20

新特性：

- iOS：提升视频编码画质。
- 支持自定义混响参数设置能力。

功能优化：

- 移动端重力感应支持 UIFixLayout 模式。
- 优化极差弱网下音视频传输表现。
- AI 降噪支持保护主讲人语音的能力。
- 移动端进房补齐当前 Router 类型事件。
- 识别不同语料进行智能编码节省带宽。
- Android 支持专业声卡识别和降低主播音质损伤。
- Windows SystemLoopback 支持排除某个进程混音能力。

版权所有：腾讯云计算（北京）有限责任公司

缺陷修复：

- Android：解决停止本地采集没有触发 GLContextDestory 回调的问题。
- Android：解决学习机设置 ResolutionMode 无效的问题。
- 全平台：修复 onMixedPlayAudioFrame 在设置采样点帧长出现杂音的问题。
- iOS：修复麦克风弹框期间调用 stopLocalAudio 无声问题。

接口行为调整：

- 文档变更：Android 14 屏幕采集接口用法变更，详见 API startScreenCapture 。
- 从 Android 14 版本开始，如果您不需要使用屏幕分享功能，则需要在您工程项目中的 AndroidManifest.xml 中移除屏幕分享前台服务的权限，操作如下：

```
<uses-permission android:name="android.permission.FOREGROUND_SERVICE_MEDIA_PROJECTION"
tools:node="remove" />
```

如果您需要使用屏幕分享功能，则需要按照 Google 的要求填写 Play 管理中心声明，参考文档如下所示。

Version 11.9 @ 2024.6.12

新特性：

- 全平台：优化 AI 降噪效果，大幅提升在网吧、会议等场景声音体验。
- 全平台：支持播放音速达全量的版权音乐，提升K歌、合唱、语聊等热门场景的互动体验。
- 全平台：支持播放 ape 格式的音频文件。
- 全平台：优化现有操作事件准确性，提升无声、声音小等异常场景排障效率。
- iOS&Android：支持高清截图能力。
- iOS：TXDeviceManager 新增设置采集参数接口 setCameraCapturerParam。

缺陷修复：

- Android：解决麦克风采集偶现电音问题。
- 全平台：修复特定BGM格式及非法数据引起的稳定性问题。
- Android：解决外放模式偶现声音小的问题。
- Windows：修复偶现获取当前摄像头不对的问题。

Version 11.8 @ 2024.5.10

新特性：

iOS：支持画中画能力。

功能优化：

- Android：音频兼容性问题优化，支持更加完善的机型采播选路策略。
- Window：优化AI降噪性能。
- Windows：单主播场景上行画质优化。
- 全平台：优化 Bgm 错误提示信息。
- Windows：缩放效果优化。

缺陷修复：

- Android：优化外放声音小 badcase。
- Mac：修复开 Loopback 杀进程无声问题。
- Mac：修复链接 HDMI、DisplayPort 无声问题。

- 全平台：修复实验性接口打开3A无效问题。
- Windows：修复不同音质类型切换导致 AGC 失效问题。
- Android：屏幕分享适配高 TargetVersion。

接口行为调整：

全平台：C++接口支持 onMixedAllAudioFrame 回调。

Version 11.7 @ 2024.3.4

新特性：

- 全平台：增加摄像头采集的警告码。
- 全平台：新增重力感应接口 setGravitySensorAdaptiveMode 。
- 全平台：支持回调本地人声音高信息，详见 enableAudioVolumeEvaluation 。

功能优化：

- 全平台：优化进房流程，大幅降低二次进房耗时。
- 全平台：仪表盘监控单个用户端到端通话质量最大路数从 16 路提升到 50 路。
- Android：优化 Android 音频采集处理策略，降低杂音率，提升声音采集效果。
- iOS：优化打断恢复重启策略，降低无声率。
- iOS：优化unity 3D引擎兼容性问题。
- iOS：优化后置三摄、双摄采集对焦效果，提升对焦速度。

缺陷修复：

- Android 修复部分漏回声 case。
- Android 修复部分蓝牙被打断中途外放问题。

接口行为调整：

- 全平台：频繁调用 `switchRole` 接口时， `onSwitchRole` 只返回最后一次调用结果。
- 全平台：增加摄像头采集相关警告码

警告码	描述
WARNING_CAMERA_IS_OCCUPIED = 1114	摄像头被占用
WARNING_CAMERA_DEVICE_ERROR = 1115	摄像头设备异常
WARNING_CAMERA_DISCONNECTED = 1116	摄像头无法连接
WARNING_CAMERA_START_FAILED = 1117	摄像头启动失败
WARNING_CAMERA_SERVER_DIED = 1118	系统异常

- 全平台： setVideoMuteImage 接口默认垫片帧率由 0fps 调整为 5fps，限定垫片最大帧率为 10fps 。
- iOS： setSubStreamEncoderParam 接口行为由设置屏幕分享的编码参数调整为设置辅路编码参数。
- Windows： selectScreenCaptureTarget 中默认高亮颜色（highLightColor）由绿色调整为黄色。
- 全平台：调整重力感应相关接口：

新增接口	(void)setGravitySensorAdaptiveMode: (TRTCGravitySensorAdaptiveMode)mode;	参见接口文档

废弃接口	setVideoEncoderRotation	废弃
	setVideoEncoderMirror	废弃
	setGSensorMode	废弃

Version 11.6 @ 2024.1.15

新特性：

- iOS：新增 `TXLivePlayer` 支持画中画。
- Windows：新增 startSystemAudioLoopback 支持采集指定的第三方进程音频，详情见 startSystemAudioLoopback。

功能优化：

- Android&iOS：优化使用 URL 播放 BGM 的成功率。
- Windows：优化并适配 Intel HEVC 软件解码器（Quick Sync Video）。
- 全平台：优化低带宽条件下的音频弱网表现。
- 全平台：优化低带宽条件下的视频弱网表现。
- 全平台：优化高丢包高延时下的音质。
- 全平台：优化 SDK 底层逻辑，提升整体稳定性。

缺陷修复：

- Android：修复频繁 `switchRoom` 时偶现首帧解码慢的问题。
- Android：修复在单次会话中尝试再次 `startScreenCapture` 时偶现的画面比例不正确的问题。
- Windows：修复设置竖屏分辨率时部分虚拟摄像头采集失败的问题。

Version 11.5 @ 2023.11.27

功能优化：

- 全平台：优化视频模块的整体性能及稳定性。
- 全平台：优化音频模块的整体稳定性。
- 全平台：优化部分 API 接口行为策略，详情见接口行为调整。
- 全平台：优化音频背景音乐模块整体策略及性能占用，减少背景音乐播放异常的情况。
- Windows：优化 HEVC 硬件解码策略，适配 AMD、Nvidia 显卡。
- Windows：优化屏幕共享整体性能，提升屏幕采集帧率及稳定性。
- Android：优化 TRTC + VODPlayer 场景下的播放效果。
- iOS&Mac：优化使用 Metal 进行预处理及渲染的性能。

接口行为调整：

- 全平台：视频设置分辨率为540P竖屏时（预期540x960），具体的编码处理分辨率由544x960调整为536x960。
- 全平台：背景音乐进度回调 `onPlayProgress` 回调间隔由200ms 调整为300ms。
- 全平台：背景音乐模块内部实现调整为单例，多实例情况下 musicID 需要全局唯一。开发者在使用子实例播放背景音乐时，请确保不同实例使用不同的 musicID。
- 全平台：本地录制事件状态码调整为异步返回，相关接口调用后默认返回 0，具体的状态码通过相应的事件回调获取。
- 全平台：启动录制事件回调 `onLocalRecordBegin` 调整以下状态码：

事件	11.5 之前版本状态码	11.5 版本状态码

录制已经启动，需要先停止录制	-1	-6
录制目录无写入权限，请检查目录权限问题	-2	-8
文件后缀名有误（例如不支持的录制格式）	-3	-2

- iOS&Android：优化移动端屏幕共享连续性，在共享暂停时保留发送最后一帧，发送帧率为1 - 2fps。
- iOS&Android：调整重力感应响应行为，只响应重力感应开或者关。

Version 11.4 @ 2023.08.30

新特性：

- 全平台： TRTCLocalRecordingParams 新增 `maxDurationPerFile` 用于控制分片录制时长，分片文件路径可通过 onLocalRecordFragment 回调获取。
- Android&iOS: V2TXLivePusher 推流本地预览增加渲染模式设置接口 setRenderFillMode 。
- Mac: 新增 `enableCrashMonitoring` ，支持捕获 crash 信息并进行本地存储，使用时需要在项目中添加 TXCCrashMonitor.framework。

功能优化：

- 全平台: 优化提升 IPv6 网络环境下的整体表现。
- 全平台: 优化合唱场景下歌词精准对齐。
- 全平台: 优化 AI 降噪算法，进一步提升降噪效果。
- 全平台: 优化提升纯音频场景下观众拉流播放流畅度。
- 全平台: 优化 switchRoom 切换房间的平滑性，避免闪黑帧。
- Android&iOS: 优化提升直播播放秒开率。
- Android&iOS: 优化音频采集处理策略，降低因采集设备异常导致的无声问题概率。
- Android: 优化麦克风被系统静默后的回调通知。
- Android: 优化特定 Android 定制设备重力感应适配逻辑，避免设备返回的重力感应方向错误时画面旋转角度不对。
- Android: 优化渲染处理方式，支持 View 双指缩放时画面实时跟随，提升浮窗播放时的用户体验。
- iOS: 优化后台状态下音频采集策略，降低因系统打断引起的无声问题概率。
- iOS: 优化提升音频设备重启速度。

Version 11.3 @ 2023.07.07

新特性：

- 全平台：新增视频画面梯形校正功能（仅 Professional 版本支持），用于手动修正透视画面角度畸变。详见 setPerspectiveCorrectionPoints 。
- 全平台：新增声音频谱回调，可用于声浪动画或音量频谱展示。详见 enableAudioVolumeEvaluation 及 TRTCVolumeInfo 。
- 全平台：新增混响效果“录音棚2”，详见 TXVoiceReverbType 。
- 全平台：新增混流接口 SEI 参数设置，用于转推 CDN 分发时透传 SEI。详见 TRTCTranscodingConfig 。
- Windows：新增音速达版权曲库音乐打分功能，可用于合唱实时评分，详见 `createSongScore`。
- iOS&Android：新增 startPlayMusic 对 .ogg 格式音乐文件的支持。
- Flutter：新增 `setSystemAudioLoopbackVolume` 方法（iOS）。

功能优化：

- 全平台：优化自适应数字增益算法，提升声音听感，增强音量稳定性，避免音量过大或过小。
- 全平台：优化视频秒开耗时，提升进房后视频第一帧画面的加载速度。
- 全平台：优化单主播推流的弱网抗性，提升网络延时抖动情况下的流畅度。
- Android：优化音频采集播放逻辑，避免在部分 Android 设备上出现声音异常的问题。
- Android：优化视频辅流硬编表现，提升共享屏幕的画面质量。
- iOS：优化音频设备重启逻辑，减少声音中断的发生次数。
- iOS&Android：TXLivePlayer 删除点播相关接口，点播视频请使用 TXVodPlayer 进行播放。

缺陷修复：

Android：修复 Android 12及以上版本系统部分本地录制的视频在苹果Safari 浏览器上无法播放的问题。

Version 11.2 @ 2023.06.05

新特性：

- 全平台：支持合唱场景下 BGM 伴奏、原唱无缝切换，详见 setMusicTrack 。
- Android：全功能版（Professional）、直播基础版（Smart）支持x86架构，并支持通过Maven获取。
- Android：满足 Android 12 及以上版本的操作系统要求，在屏幕采集时启动前台服务，详见：enableForegroundService 。
- iOS：支持在 Apple 芯片设备上通过 Xcode 模拟器运行 SDK。
- Mac：支持获取屏幕窗口信息时返回宽高等信息，对齐 Windows，详见：TXCScreenSourceInfo。

功能优化：

- 全平台：优化合唱场景的整体音质，提升合唱效果，减少合唱延迟。
- 全平台：优化上下麦时的音频效果，上下麦体验更平滑。
- 全平台：优化极限弱网下的音频体验。
- 全平台：优化直播单主播推流时的弱网体验。
- 全平台：优化视频通话场景大小流切换过程的流畅度。
- Android&iOS：优化音乐场景下的音质表现，提升合唱体验。
- Android&iOS：优化不同音量类型下使用蓝牙耳机的体验。
- Android：优化硬件解码延时，提升首帧体验。
- Android：优化耳返功能，提升开关耳返时的体验。
- Android：优化 Android 设备的采集兼容性，减少音频异常问题。
- iOS：优化画质表现，提升视频体验。

缺陷修复：

- Windows：修复窗口分享时偶现闪烁问题。
- Mac：修复 Intel 芯片设备上使用摄像头采集时偶现的硬编码器编码画面呼吸效应问题。

Version 11.1 @ 2023.04.17

新特性：

- 全平台：新增 onVoiceEarMonitorAudioFrame 数据回调，用于获取或修改耳返数据。
- 全平台：数据回调 onCapturedRawAudioFrame 优化命名为 onCapturedAudioFrame。
- Mac：窗口分享支持 PPT 放映模式。

功能优化：

- 全平台：优化 log 文件自动清理逻辑，防止 log 文件夹体积超标。
- iOS&Android：优化解码渲染时的颜色矩阵兼容性，避免引入色彩偏差。
- Android：优化低端机在高分辨率场景下偶现硬编无法启动导致性能开销增大的问题。
- Android：优化 Android 12以上系统偶现硬编码率不受控的问题。
- Android：优化在合唱场景中少量机型主播采集声音小的问题。
- Android：优化在合唱场景中偶现的声音剪切严重的问题。

缺陷修复:

- Android：修复蓝牙耳机连接状态但未启用时漏回声。
- Windows：修复开关系统混音偶现漏回声的问题。

Version 11.0 @ 2023.03.08

新特性

Android：接口变更，TXLiveBase.setLibraryPath 返回类型调整为 bool，表示加载 SDK 动态库是否成功。

功能优化:

- 全平台：提升弱网情况下在线 BGM 的播放成功率。
- 全平台：优化了 VideoCall 进房场景下，首帧播放流畅度。
- Android：优化音频兼容性，减少电流杂音、无声类问题。

缺陷修复

- 全平台：修复了使用 sendCustomCmdMsg 功能时，在频繁进退房情况下偶现的 crash 问题。
- 全平台：修复了本地退房时，错误回调远端主播的 onRemoteUserLeaveRoom、 onUserVideoAvailable、onUserAudioAvailable 问题。
- 全平台：修复了远端主播静音时，可能听到杂音的问题。

Version 10.9 @ 2023.01.09

新特性

Android：增加对外置麦克风设备（例如领夹式麦克风）的音频采集支持。

功能优化

- 全平台：优化音画同步问题，提升视频播放平滑度。
- 全平台：优化部分弱网场景的上行延迟，提升视频通话的互动效果。
- Windows&Android：优化特定场景下设定音乐音质后易产生爆音的问题。
- iOS：优化外录屏在系统横竖屏切换时随系统方向自动转正，提升观看端的体验。
- Mac：优化 MacOS 12.3 及之后版本的录屏性能，降低 CPU 开销及内存占用。
- Android：优化少量机型在媒体音量下，插入耳机后仍存在的声音外发问题。

Version 10.8 @ 2022 10.31

新特性

全平台：新增搓碟音效，提供更加全面的在线 K 歌体验，详见：TXAudioEffectManager. setMusicScratchSpeedRate 。

功能优化

- Android：优化视频解码启动速度，有效提升画面秒开速度，最快可以达到 50ms。

- 全平台：优化 NTP 时间的准确性，详见：TXLiveBase.updateNetworkTime。

缺陷修复

- 全平台：修复下特定场景下（无音视频上行）混流机器人回推 TRTC 房间场景中，偶现的拉流异常以及回调错误的问题。
- 全平台：修复观众进房后切换角色时，因网络类型变化偶现的音视频上行失败问题。
- 全平台：修复在断网重连过程中出现的音质切换不生效问题。
- 全平台：修复在断网重连过程中偶现的上行无声问题。
- Android & iOS：修复当调用 muteRemoteVideoStream 时会移除最后一帧视频画面的问题。

Version 10.7 @ 2022.09.20

新特性

- 全平台：云端混流支持调整每路输入流的音量，详见 TRTCMixUser.soundLevel。
- 全平台：新增了 onRemoteAudioStatusUpdated 回调接口，可用于更好地识别和监控远端音频流状态。

功能优化

- 全平台：升级编码内核，提升屏幕分享场景的画质。
- 全平台：优化弱网下编码码控效果。

缺陷修复

- iOS：修复 iPad 部分设备采集音量较小的问题。
- Android：修复偶现连接蓝牙耳机但是声音外放的问题。
- 全平台：修复频繁进退房场景下偶现的 crash 问题。

Version 10.6 @ 2022.09.05

功能优化

- 全平台：提升在 IPv6 网络环境下的进房速度。
- 全平台：优化弱网络环境下音频的恢复效率以及音画同步效果，提升通话体验。
- 全平台：优化弱网络环境下的连接保持能力，减少断网重连概率。
- 全平台：优化 Music 档位（在 startLocalAudio 时指定）下音量较小的问题，提升用户体验。
- Mac：优化使用蓝牙耳机时的沟通体验，杂音更少，声音更清晰。
- Android：优化立体声采集的兼容性，支持更多机型。
- Android：优化偶现的漏回声问题，提升沟通体验。

缺陷修复

- Android & iOS：修复在 Speech 档位（在 startLocalAudio 时指定）下偶现的漏字问题。
- Mac：修复切换麦克风时偶现的回声消除失效的问题。

Version 10.5 @ 2022.08.23

功能优化

- 全平台：优化 qos 策略，提升弱网体验。
- iOS&Android：全链路降低延迟；优化耳返体验。
- Android：优化视频解码的内存管理，防止出现内存堆积。
- Windows：优化内置麦克风的降噪效果，尤其是在音乐模式下，表现更佳。

- Mac：优化开启麦克风采集时，大概率出现的杂音问题。

缺陷修复

全平台：修复频繁进退不同房间时，回调事件：OnUserVideoAvailable 和 OnUserAudioAvailable 偶现异常的问题。

Version 10.4 @ 2022.07.25

新特性

- iOS&Android：自定义视频采集支持 RGBA32 格式，详见：sendCustomVideoData。
- Windows & Mac：水印设置支持本地预览，详见：setWaterMark。

功能优化

- Android：优化低延迟耳返及双声道采集的兼容性。
- Android：优化硬解码切软解码的策略，提升解码性能。
- iOS：优化 iPad 采集音量小的问题。

缺陷修复

- 全平台：修复偶现的进退房回调异常的问题。
- Windows：修复切换分享窗口，新窗口的内容被剪裁的问题。

Version 10.3 @ 2022.07.08

新特性

- Windows：新增录制本地录制功能，可用于在本地录制互动直播或音视频通话完整内容。详见 ITXLiteAVLocalRecord。
- Windows&Mac：新增参数支持在 startMicDeviceTest 接口中开启/关闭 播放麦克风检测时麦克风采集到的声音。详见 startMicDeviceTest。

功能优化

全平台：优化 Music 音质下的声音效果。

缺陷修复

- 全平台：修复房间用户列表偶现的回调异常问题。
- Windows：修复视频播放过程偶现的画面卡住问题。
- Windows：修复视频播放过程偶现的播放失败问题。
- Windows：修复音频自定义采集场景中出现回声的问题。

Version 10.2 @ 2022.06.23

新特性

- 全平台：全新推出更加灵活，且功能强大的混流转推 API。详见：startPublishMediaStream。
- 全平台：新增 3D 音频特效功能，详见：enable3DSpatialAudioEffect。
- 全平台：新增人声检测功能，当 muteLoalAudio 和 setAudioCaptureVolume 为 0 时不会影响人声检测结果。详见 enableAudioVolumeEvaluation，Tips：方便提示用户开麦。
- 全平台：切换角色时，增加支持权限校验的功能。详见：`switchRole(TRTCRoleType role, const char* privateMapKey)` 。
- iOS&Mac：自定义预处理的 C++ 接口，支持以纹理方式对接视频处理。

功能优化

- Android：优化耳返效果，降低延迟。
- Android：优化音频的采集链路，解决部分机型存在的杂音问题。
- iOS：优化上行视频处理链路，节省 CPU、GPU 占用。
- Windows&Mac：优化窗口分享时的编码性能，编码宽高不再受采集窗口大小的影响。
- Windows：优化性能，减少内存碎片及其分配时造成性能开销。

缺陷修复

- 全平台：修复切换网络类型时，偶现的上行失败问题。
- iOS：修复在部分 iOS 14系统上，本地录制文件存在的杂音问题。

Version 10.1 @ 2022.06.06

新特性

- 全平台：支持平滑切换角色，音视频播放不会因为切角色短暂中断。
- iOS：支持立体声音频采集。
- Android：在 Android 10 及以上系统支持采集系统播放音频（startSystemAudioLoopback）。

功能优化

- 全平台：优化音乐场景下的回声消除能力，音质效果更自然。
- 全平台：优化切换角色 + muteLocalAudio 下的音质和启动效果。
- 全平台：优化带宽预测 onSpeedTest 回调。
- iOS：优化内存管理，避免内存堆积的问题。
- Android：优化部分机型手机上耳返的延迟。
- Windows：优化视频下行时视频渲染链路的性能。
- Windows：优化立体声采集逻辑，有效避免漏回声问题。

缺陷修复

- 全平台：修复退房回调（onExitRoom）的 reason 异常问题。
- 全平台：修复上行自定义视频发送时，时间戳相等情况下的黑屏问题。
- 全平台：修复先 muteLocalAudio 再 startLocalAudio 音频时 crash 问题。
- 全平台：修复不手动设置3A场景下开启自定义音频采集会打开3A。
- 全平台：修复音频自定义渲染偶现的杂音问题。
- iOS：修复中途设置 log 路径（setLogDirPath）且沙盒变化时，内存泄漏的问题。
- iOS&Mac：在系统音频服务异常时，BGM 连播场景的崩溃问题。
- Android：修复偶现的蓝牙耳机不断重连接问题。
- Android：修复部分手机上偶现的无声问题。
- Android：修复红米等部分机型反复插拔耳机导致的崩溃问题。
- Windows&iOS：修复截图失败的问题。
- Windows：修复点播播放器开启镜像后，关闭 vod 必现 crash。
- Windows：修复播片 pts 未使用 generateCustomPts，多个播片播放可能导致 pts 回退问题。
- Windows：修复偶现选择禁画显示图片后崩溃问题。

Version 10.0 @ 2022.05.17

功能优化

- 全平台：优化主播进退房通知回调（ onRemoteUserEnterRoom / onRemoteUserLeaveRoom ）的速度。
- Windows：优化屏幕分享的性能，在未设置过滤窗口时，性能提升一倍。

缺陷修复

- iOS&Mac：修复开始播放 BGM 时，偶现的 onComplete 回调错误的问题。
- Android：修复一例网络模块导致的崩溃问题。
- 全平台：修复 SEI 发送异常的问题。

Version 9.9 @ 2022.05.06

功能优化

- Windows：优化视频链路，降低性能开销。
- Windows：优化 Systemloopback 采集前处理，保留双声道的效果。
- Mac：优化采集音量过大时导致的爆音问题，提升音质体验。
- Mac：提升屏幕分享（辅路）的画质。
- Android：优化采集延迟，提升耳返体验。

缺陷修复

- Android：修复房间号不支持21亿以上号段的问题。

Version 9.8 @ 2022.04.21

新特性

- Windows：新增“重金属”、“萝莉音”等音效接口，详见 `ITXAudioEffectManager.setVoiceChangerType` 。
- Windows：支持本地画面被暂停期间设置替代的图片进行推流，即垫片推流。

功能优化

全平台：优化视频场景下的性能。

缺陷修复

- Mac：修复录制系统声卡音频时，驱动安装失败的问题。
- 全平台：修复本地屏幕分享（辅路）时自定义渲染失效的问题。

Version 9.7 @ 2022.04.06

功能优化

- iOS&Android：优化 Music 音质的效果。

> **Tips:**
> 全平台均可通过 API 接口 TRTCCloud.startLocalAudio (TRTCAudioQualityMusic) 开启 Music 音质，下同。

- Windows：优化 Music 音质下的采播效果，降低对音质的损伤。
- Windows：提升部分专业声卡上的兼容性适配问题，有效提升音质。
- Windows：第三方进程混音优化，让第三方进程混音功能可以适配更多的场景。

缺陷修复

- 全平台：修复 CDN 播放偶现视频画面花屏的问题。

- iOS&Android：修复直播播放听筒扬声器切换无效的问题。
- iOS&Android：修复通过 API 设置 Music 音质时实际音质不符合预期的问题。
- iOS：修复软编码过程中偶现的内存泄露问题。
- iOS：修复本地视频画面渲染，偶现无首帧回调的问题。
- Windows：修复屏幕分享模式下鼠标采集偶现的异常崩溃。
- Windows：修复 Music 音质下，扬声器播放声音异常问题。
- Windows：修复部分摄像头 startCameraDeviceTest 无法正常打开的问题。

Version 9.6 @ 2022.03.24

重要更新

- 全平台：完成三方库合规整改，符合国内、海外的合规要求。
- 全平台：优化 TRTC SDK 体积，详细数据见下表：

平台	优化前	优化后
Android	armv7：6.95Marm64：7.94M	armv7：4.32Marm64：4.85M
iOS	arm64：3.23M	arm64：3.15M
Windows	Win32：21.3MWin64：26.9M	Win32：15.0MWin64：17.2M
Mac	x86_64：18.1M	x86_64：15.8M

故障修复

全平台：修复已知问题，提升稳定性。

功能优化

- iOS：修复在补光灯下偶现的曝光过度问题。
- Mac：优化纹理上传，提升性能。
- Android：优化美颜等预处理流程，修复低端机器下的采集卡顿问题。
- Windows：Live V1 升级到 V2 接口，提供更稳定的 Live 组件。
- Windows：提升了低端机上的显卡兼容性。

Version 9.5 @ 2022.01.11

故障修复

- 全平台：提升 API 易用性，修复部分 API 特定调用时序导致自定义渲染播放黑屏的问题。
- Windows：修复屏幕分享采集区域不完整的问题。
- iOS：修复 muteLocalVideo 调用后退房下次进房还是不推流状态的问题。
- iOS：修复混流设置背景图无效的问题。

功能优化

- 全平台：优化通话场景在弱网时的流畅度。
- Windows：优化摄像头兼容性，解决部分设备采集帧率与设定值不符或开启失败的问题。
- iOS：提升兼容性，降低和其他渲染组件如 cocos2D 共用时的冲突。
- Android：修复上行关闭再开启摄像头，播放端先显示关闭前最后一帧再正常显示的问题。

Version 9.4 @ 2021.12.08

功能新增

- 全平台：新增语音追光功能，适用于大型语音连麦场景，即使在多人同时开麦的嘈杂的环境下，仍然能聚焦关键用户的声音。您可以通过 setRemoteAudioParallelParams 接口进行设置。
- Mac：增加对系统声音采集 startSystemAudioLoopback 的双声道支持。
- iOS：增加对 24 位 wav 格式的背景音乐文件的支持。
- Android&iOS：本版本符合国家隐私安全规范的规定，已经经过腾讯内部多款产品的验证。

问题修复

- 全平台：修复快速调用 switchRoom 可能导致切换房间失败的问题。
- iOS：修复在应用内录屏 startScreenCaptureInApp 过程中设置 setVideoEncoderRotation 无效的问题。
- iOS：修复系统录屏 startScreenCaptureByReplaykit 过程中偶现的内存上涨问题。

功能优化

- 全平台：提升进房速度，减少进房耗时的波动。
- Mac：解决了在屏幕采集的过程中开启鼠标采集后，CPU 和 内存占用率较高的问题。
- Android：调整屏幕分享时的屏幕采集分辨率，使其能始终对齐屏幕分辨率，避免分享出的画面出现黑边等问题。
- Android：提升视频硬解兼容性，解决部分手机在播放视频分辨率发生变化时可能出现的黑屏问题。
- Windows：优化音量增益算法，解决部分设备出现增益过大导致杂音比较明显的问题。

Version 9.3 @ 2021.11.03

故障修复

- 全平台：修复 point2PointDelay 有时获取不到，数值为0的问题。
- 全平台：修复偶现解析失败 SEI 消息丢失的问题。
- Mac：修复在 MacOS 12 beta 上摄像头不出帧的问题。
- iOS&Mac：修复特定顺序提前调用 startRemoteView 看不到画面的问题。
- Windows：修复使用竖屏编码并开启美颜的情况下画面出现锯齿的问题。
- Windows：修复第三方美颜开启情况下，切换分辨率后自定义渲染不回调的问题。

功能优化

- 全平台：优化弱网情况下视频秒开速度。
- 全平台：优化弱网调控策略，同场景下更流畅。
- 全平台：优化测速功能，支持对当前网络带宽进行检测。
- 全平台：优化对 TCP 传输协议的支持，更好地应对复杂的网络环境。

Version 9.2 @ 2021.09.23

功能新增

- Android&iOS：支持 Socks5 代理。
- Windows：TRTCAudioQualityMusic 高音质场景新增自适应回声消除功能，自动平衡音质与回声消除强度。
- 全平台：新增设置语音音调功能。

故障修复

- Windows：修复 Mac 安装 Windows 的环境部分摄像头不吐数据问题。

版权所有：腾讯云计算（北京）有限责任公司

- Android：修复偶现 CDN/TRTC 互切后没有上行音频的问题。
- iOS：修复 Web 端屏幕分享导致 iOS 接受端自定义渲染花屏的问题。

功能优化

- Android：优化硬解码时导致 ANR 的问题。
- Android：优化摄像头本地预览角度兼容的问题。
- Android：优化首帧秒开速度。
- Android&iOS：优化合唱模式 3A 策略。
- Windows：优化 AGC 算法，降低出现声音过小和声音过大的问题的概率。
- 全平台：优化弱网环境抗抖动算法，视频播放更流畅。

Version 9.1 @ 2021.09.04

功能新增

- 全平台：C++ 接口支持音频帧回调格式设置。
- Windows：播片增加 ac3 格式的支持。
- Windows：摄像头信息支持获取支持的分辨率列表，具体请参见 ITXDeviceCollection.getDeviceProperties 。
- Windows：支持 Nvidia、Intel、AMD 硬解。
- Mac：新增本地媒体录制支持。

问题修复

- 全平台：修复偶现的进房失败问题。
- Mac：修复屏幕分享时切换分辨率，预览画面闪一下的问题。
- Android：修复子房间切回主房间时，辅路视频画面显示异常的问题。
- Android：修复特定场景下偶现的帧率设置不生效问题。
- Windows：修复观众切换为 CDN 流后无法拉流的问题。
- Windows：修复播片功能播放特定格式视频画面消失的问题。

质量优化

- 全平台：优化弱网下音视频体验。
- Android：优化退房时的音频状态管理。
- Android：优化音频采集启动失败后的恢复逻辑，提升成功率。
- Android：优化特定条件下视频画面过曝的问题。

Version 9.0 @ 2021.08.06

功能新增

- iOS：支持设置系统采集音量，详情请参见 setSystemAudioLoopbackVolume 。
- 全平台：支持设置自定义音轨的音量，详情请参见 setMixExternalAudioVolume 。
- 全平台：状态回调可区分音频和视频的丢包率，详情请参见 TRTCRemoteStatistics 。

质量优化

- 全平台：优化订阅流程，提升手动订阅的秒开速度。
- 全平台：修复特定场景 onExitRoom 回调重复的问题。

问题修复

- Android：修复自定义采集动态设置码率和帧率无效的问题。
- iOS：修复先开启录屏辅路，再开启摄像头推流导致的推流失败问题。
- iOS：修复本地视频录制模糊的问题。
- iOS：修复若干稳定性问题。
- Windows：修复屏幕分享时采集帧率异常的问题。
- Windows：修复屏幕分享切换目标时，播放端会先显示一帧旧画面的问题。

Version 8.9 @ 2021.07.15

功能新增

- Android：自定义渲染支持指定外部 GLContext，可以更灵活使用 OpenGL 环境。
- Windows：采集系统播放声音 startSystemAudioLoopback 时支持指定扬声器设备。
- Windows：支持 NVIDIA 平台硬编码，提升推流性能表现。
- 全平台：新增云代理支持，针对企业防火墙内部的环境，安全配置更友好。
- 全平台：接口 muteLocalVideo 和 muteRemoteVideoStream 增加对流类型的支持。
- 全平台：统计状态回调 onStatistics 新增对本地网关延迟的统计 gatewayRtt，用于判断用户到 Wi-Fi 路由器的网络质量。
- 全平台：音频录制接口 startAudioRecording 支持录制成更多的音频格式。

质量优化

- 全平台：优化某些场景下的声音播放出现颤抖的问题。
- Android：优化画面秒开速度。
- Android：升级音频前处理算法，通话声音更清晰。

问题修复

- Windows：修复 VODPlayer 播片推流时本地录制音频文件会有重音的问题。
- Windows：修复高 DPI 环境下并启用过滤窗口时部分场景 crash 的问题。
- iOS：修复外录屏辅路推流设置横屏无效的问题。
- iOS：修复只开启远端自定义渲染并指定使用 RGBA 格式数据时的内存泄漏问题。
- 全平台：修复偶现进房失败问题。

Version 8.8 @ 2021.06.21

功能新增

Android&Mac&iOS：支持外部接管音频播放，请参见 API enableCustomAudioRendering 。

质量优化

- 全平台：优化 mixExternalAudioFrame 易用性，现在无需严谨把控调用时机。
- Mac：降低屏幕分享开启鼠标捕捉时 CPU 的开销。
- Windows：优化 AGC 声音增益效果，更快更及时地进行调整。
- Windows：优化启用窗口过滤时屏幕分享的性能开销。

问题修复

- iOS：修复播放 AAC 格式本地音频文件总时长不准的问题。
- Android：修复部分机型切换后台时播放声音卡顿的问题。

Version 8.7 @ 2021.5.25

功能新增

- 全平台：增加外接音频设备的异常检测。注册 onStatistics 回调后， 您可以用 TRTCLocalStatistics 中的 audioCaptureState 来实时检测长时间静音、破音、异常间断问题。
- Windows：自定义采集支持输入 RGBA 格式的视频数据。

质量优化

- 全平台：优化 BGM 资源管理，及时释放内存占用。
- 全平台：推流端退后台暂停视频上行时，播放端能及时收到 onUserVideoAvailable(false) 的通知。
- Mac：优化屏幕分享时鼠标捕捉的 CPU 和内存占用。

问题修复

- Android ： 修复 setRemoteViewFillMode 部分机型偶现不生效的问题。
- iOS/Mac：修复停止自定义美颜时的内存资源释放问题。

Version 8.6 @ 2021.05.08

- 全平台：优化网络流控算法，进一步提升音视频传输质量。
- 全平台：优化切换角色上下麦时音频播放的流畅度。
- iOS&Mac&Windows：优化音频处理模块，提升了 SPEECH 模式和 DEFAULT 模式的语音质量。
- iOS&Mac：优化自定义音频采集在高 CPU 场景下的适应性。
- iOS&Android ： 支持录屏视频通过辅路进行分享，对齐桌面端版本。
- Mac：增加对苹果 M1 架构的原生支持。
- Windows：优化内存分配逻辑，提高稳定性。

Version 8.5 @ 2021.03.24

功能新增

- Mac：优化屏幕分享功能，您可以在分享目标窗口的同时指定其他窗口一起分享出去，请参见 API addIncludedShareWindow 。
- 全平台：新增播片功能，您可以使用 TXVodPlayer 与 TRTCCloud 绑定，把点播正在播放的内容通过 TRTC 的辅路推流分享出去。
- 全平台：新增辅路自定义采集，请参见 API sendCustomVideoData。
- 全平台：新增自定义混音功能，您可以将自己的一路音轨混入 SDK 的音频处理流程中，SDK 会先将两路音轨混合后再一起发布出去，请参见 API mixExternalAudioFrame。
- 全平台：支持指定纯视频混流，混流控制更灵活。

质量优化

- Mac：startSystemAudioLoopback 支持双声道。
- Windows：选择幻灯片窗口进行屏幕分享时，支持自动切换到放映窗口。
- 全平台：状态回调增加端到端延迟。

问题修复

- iOS：优化部分设备偶现后台 OpenGL 渲染 crash 的问题。
- iOS：优化屏幕画面静止时屏幕分享在播放无法播放的问题。

Version 8.4 @ 2021.02.08

功能新增

- Mac：开始支持采集 Mac 操作系统的输出声音，也就是跟 Windows 端一样的 SystemLoopback 能力，该功能可以让 SDK 采集当前系统的声音，开启这个功能后，主播就可以很方便地向其他用户直播音乐或者电影文件。
- Mac：屏幕分享开始支持本地预览功能，您可以通过一个小窗口向用户展示屏幕分享的预览内容。
- Windows：新增进程音量调整能力，使用 setApplicationPlayVolume 可以设置系统的音量合成器的音量大小。
- 全平台：新增本地音视频录制功能，主播可以在推流过程中把本地的音频和视频录制成一个 mp4 文件，请参见 startLocalRecording 。

质量优化

- 全平台：优化 Music 模式下的声音质量，更加适合类似 cloubhouse 的语音直播场景。
- 全平台：优化音视频链路的网络抗性，在 70% 的极端网络环境下，音视频依然较为流畅。
- Windows：优化部分场景下的直播音质，大幅减少了声音损伤问题。
- Windows：性能优化，在部分使用场景下的性能较旧版本有 20%-30% 的提升。

问题修复

- Windows：修复 Windows Server 2019 Datacenter x64 系统上启动桌面分享 crash 的问题。
- Windows：修复分享窗口的同时改变目标窗口大小会偶发分享意外终止的 BUG。
- Windows：修复部分型号的摄像头采集不出画面的问题。
- iOS：修复 snapvideoshot 会造成 CAAnimation 动画卡顿的问题。
- iOS&Mac：修复使用同一个 View 轮流显示摄像头和屏幕分享画面时，屏幕分享画面黑屏的问题。
- iOS：修复使用第三方美颜组件时在 iPhone 6s 上可能会出现花屏的问题。
- iOS：修复点播与 TRTC 同时使用时，在停止点播播放时偶现 crash 的问题。
- Android：修复使用蓝牙耳机时被电话打断，拒绝接听电话后声音通过扬声器播放的问题。

Version 8.3 @ 2021.01.15

功能新增

这个版本我们重点优化了自定义采集相关的业务逻辑：

- iOS & Android & Mac：优化音频模块，以确保在您使用 enableCustomAudioCapture 采集音频数据送给 SDK 处理时 SDK 依然能够保持很好的回声抑制和降噪效果。
- iOS & Android：若需在 TRTC SDK 的基础上，继续增加自己的声音特效和声音处理逻辑，使用 8.3 版本会更加简单，因为您可以通过 setCapturedRawAudioFrameDelegateFormat 等接口，设置音频数据的回调格式，包括音频采样率、音频声道数和采样点数等，以便您能够以自己喜欢的音频格式处理这些音频数据。
- 全平台：若需自己采集视频数据，并同时使用 TRTC SDK 自带的音频模块，可能会遇到音画不对齐的问题。这是因为 SDK 内部的时间线有自己的控制逻辑，因此我们提供了 generateCustomPTS 接口。您可以在采集到的一帧视频画面时，调用此接口并记录一下当前的 PTS（时间戳），随后调用 sendCustomVideoData 时带上这个时间戳，即可很好地保证音画同步。
- Windows：版本 SDK 增加了对域名格式的 Socks5 代理地址的支持。

问题修复

- 全平台：修复偶现音频数据时间戳异常导致录制内容音画不同步的问题。
- Windows：优化窗口分享在高 DPI 环境下的兼容性。
- Windows：获取可分享的窗口列表时增加最小化的窗口，最小化窗口的缩略图是其进程的图标。
- Windows：修复 SDK 启动后非必要的 DXGI 占用问题。
- iOS：修复手动设置焦点会导致 ANR 的问题。
- iOS：修复偶现切换前后摄像头无效的问题。
- iOS：修复 VODPlayer 减速播放 crash。

- iOS：修复偶现进房后默认从听筒播放的问题。
- iOS & Android：优化回声消除和噪声抑制的效果，并且耳返也能听到混响的效果。
- Android：修复偶现硬解绿屏花屏的问题。
- Mac：修复窗口分享并开启高亮时，窗口贴边会造成高亮边框闪烁的问题。
- Mac：修复渲染视图移动时会黑屏的问题。

Version 8.2 @ 2020.12.23

功能新增

- iOS&Android：新增回调混合本地采集与所有播放的音频数据，本地音频录制更方便啦。
- Android：视频渲染组件 TXCloudVideoView 支持通过 `addVideoView(new TextureView(getApplicationContext()))` 接口将 TextureView 用于本地渲染。
- Android：自定义渲染回调支持 RGBA 格式的视频数据。
- Windows：支持本地摄像头采集和播放远端视频流截图，请参见 ITRTCCloud.snapshotVideo。
- Windows：屏幕分享支持通过 addExcludedShareWindow 和 addIncludedShareWindow 接口排除或强制包含您所指定的窗口，从而实现更灵活的屏幕分享能力。
- Mac&iOS：自定义渲染的模式下也可以调用 TRTCCloud.snapshotVideo 截取视频流图片。

质量优化

- Android：在线直播编码质量优化，视频画面更清晰。
- Windows：优化回声消除算法，进一步提升回声消除的效果。

问题修复

- iOS：修复 VODPlayer 和 TRTC 同时使用时偶现的音频播放异常的问题。
- Android：修复自定义美颜引起的本地渲染黑屏问题。
- Windows：修复偶现的当前进程无法退出的问题。

Version 8.1 @ 2020.12.03

功能新增

- 全平台：统计信息（onStatistics）中新增远端视频卡顿的相关统计指标。
- 全平台：支持通过音量调节接口 setAudioPlayoutVolume（100-150）实现声音的增益效果。
- iOS&Android：新增 setLocalVideoProcessListener 接口，能更好地支持第三方美颜 SDK 的集成。
- C# ：同步升级至最新版本的 API 接口。

质量优化

- 全平台：优化戴耳机时的声音处理算法，提高声音音质。
- Android：优化音频前处理算法，降低 3A 算法对音质的影响。

问题修复

- iOS：修复部分偶现的强杀 App 导致的崩溃问题。
- Android：修复当采集帧率比较高时出现的美颜效果异常问题。
- Windows：修复高 DPI 下屏幕分享偶现的崩溃问题。

Version 8.0 @ 2020.11.13

新增

- 全平台新增 C++ 统一 API，请参见 cpp_interface/ITRTCCloud.h。
- 全平台支持字符串房间号，请参见 TRTCParams.strRoomId。
- 全平台新增 TXDeviceManager 设备管理类。
- 全平台新增 API TRTCCloud.switchRoom，支持不停止采集，直接切换房间。
- 全平台新增 API TRTCCloud.startRemoteView 开始渲染远端视频画面。
- 全平台新增 API TRTCCloud.stopRemoteView 停止渲染远端视频画面。
- 全平台新增 API TRTCCloud.getDeviceManager 获取设备管理类。
- 全平台新增 API TRTCCloud.startLocalAudio 开启本地音频的采集和上行。
- 全平台新增 API TRTCCloud.setRemoteRenderParams 设置远端图像的渲染配置。
- 全平台新增 API TRTCCloud.setLocalRenderParams 设置本地图像的渲染配置。

优化

- Android 优化软硬解切换逻辑。
- Windows 优化 System loopback 音频采集音质及回声消除效果。
- Windows 优化音频设备选择逻辑，降低无声率。
- Windows 优化双讲剪切效果。
- 全平台优化手动接收模式切换角色时的秒开效果。
- 全平台优化音频接收逻辑，提升音频效果。
- 全平台优化 sendCustomCmdMsg 可靠性。

修复

- iOS 修复 muteLocalVideo 调用导致本地视频渲染暂停的问题。
- iOS 修复在前后台切换时偶现调用系统组件可能导致卡死的问题。
- iOS 修复开启音效时，耳返音频断断续续的问题。
- Android 修复切通话音量播音效的时候电话打断，音效不会停止播放的问题。
- Android 修复偶现音频采集启动失败的问题。
- Windows 修复偶现本地视频渲染黑屏的问题。
- Windows 修复进程退出时可能 crash 的问题。
- Windows 优化蓝牙耳机支持，修复蓝牙耳机无声问题。
- Windows 修复屏幕分享结束时抢焦点的问题。
- 全平台修复状态回调丢包率统计异常问题。

Version 7.9 @ 2020.10.27

新增

- Mac：屏幕分享支持过滤选定的窗口，用户可以将自己不希望分享出去的窗口排除掉，从而更好地保护用户的隐私。
- Windows：屏幕分享支持设置“正在分享”提示边框的描边颜色以及边框宽度。
- Windows：屏幕分享在分享整个桌面时支持开启高性能模式。
- 全平台：支持自定义加密，用户可以对编码后的音视频数据通过暴露的 C 接口进行二次处理。
- 全平台：在 TRTCRemoteStatistics 中新增音频卡顿信息回调 `audioTotalBlockTime` 和 `audioBlockRate` 。

优化

- iOS：优化了音频模块的启动速度，让首个音频帧可以更快地采集并发送出去。
- Windows：优化系统回采的回声消除算法，让开启系统回采（SystemLoopback）时有更好的回声消除能力。

- Windows：优化屏幕分享功能中的窗口采集抗遮挡能力，支持设置过滤窗口。
- Android：针对大部分 Android 机型进行了耳返效果的优化，使耳返延迟降低到一个更舒适的水平。
- Android：针对 Music 模式（在 startLocalAudio 时指定）下的点对点延迟进行了进一步的优化。
- 全平台：在手动订阅模式下，优化了观众和主播角色互切时的声音流畅度。
- 全平台：优化了音视频通话中的弱网抗性，在较差的网络下也能有更优质的音频流畅度。

修复

- iOS：修复部分场景下偶现的视频画面不渲染问题。
- iOS：修复用户在戴耳机并且是 Default 音质下偶现的杂音问题。
- iOS：修复部分已知的内存泄露问题。
- iOS：修复偶现的 replaykit 扩展录屏结束后的 crash 问题。
- iOS：解决模拟器环境下的编译问题。
- Android：修复部分手机在 App 长时间切到后台，之后又再次切回前台时偶现的音画不同步问题。
- Android：修复切后台后没有释放麦克风的问题。
- Android：修复 SDK 内部部分 OpenGL 资源未及时释放的问题。
- Windows：修复个别场景下偶现的杂音问题。
- 全平台：修复部分偶现的崩溃问题，提升 SDK 的稳定性。

Version 7.8 @ 2020.09.29

新增

- Mac：新增系统音量变化回调，详见 TRTCCloudDelegate.onAudioDevicePlayoutVolumeChanged 。
- Windows：新增支持跨屏指定区域进行屏幕分享。
- Windows：新增窗口分享支持过滤指定窗口进行抗遮挡，详见 TRTCCloud.addExcludedShareWindow 和 TRTCCloud.removeExcludedShareWindow。
- Windows：新增系统音量变化回调，详见 ITRTCCloudCallback.onAudioDevicePlayoutVolumeChanged。

优化

- iOS：支持 VODPlayer 和 trtc 一起使用，并且支持回声消除。
- iOS&Mac：支持垫片推流，使用方法见 TRTCCloud.setVideoMuteImage 。
- Android：支持垫片推流，使用方法见 TRTCCloud.setVideoMuteImage 。
- Android：优化声音路由策略，支持戴耳机时，声音只从耳机播放。
- Android：支持部分系统下采用低延迟采集播放，降低 Android 系统通话延迟。
- Android：支持 VODPlayer 和 trtc 一起使用，并且支持回声消除。
- Windows：兼容虚拟摄像头 e2eSoft Vacm。
- Windows：支持同时调用 startLocalPreview 和 startCameraDeviceTest。
- Windows：支持屏幕分享走主路的同时，调用 startLocalPreview 开启本地预览。
- Windows：降低因 SDK 内部播放缓冲引发音频延迟较大的问题。
- Windows：优化音频启动逻辑，在仅播放的情况下不占用麦克风。

修复

- iOS：修复 iPhone SE 播放声音小的问题。
- iOS：修复子房间 (TRTCCloud.createSubCloud) 调用 muteRemoteAudio 触发 crash 的问题。
- iOS：修复偶现渲染 crash 问题。

版权所有：腾讯云计算（北京）有限责任公司

- iOS：修复前后台切换时在部分 iPad 视频渲染偶现卡死主线程的问题。
- iOS：修复已知内存泄露。
- iOS：修复 iOS14 提示“查找并连接本地网络上的设备”的问题。
- Mac：修复 getCurrentCameraDevice 始终返回 nil 的问题。
- Mac：修复部分 USB 摄像头无法打开的问题。
- Mac：修复屏幕分享指定区域面积为0时的 crash 问题。
- Android：修复未配置 READ_PHONE_STATE 权限时，Android5.0 设备 crash 的问题。
- Android：修复蓝牙耳机断开再连上之后音频采集和播放异常的问题。
- Android：修复已知 crash 问题。
- Windows：修复64位 SDK 多次开关屏幕分享会 crash 的问题。
- Windows：修复部分系统使用 OpenGL 会 crash 的问题。

Version 7.7 @ 2020.09.08

优化

- 全平台：优化辅路（即屏幕分享）的秒开速度。
- iOS：优化内部线程模型，提升在30路以上并发播放的场景中的运行稳定性。
- iOS&Android：优化 Audio 模块的性能，提升首帧的采集延迟，新版本可以更快的获得首个音频帧。
- iOS&Android：优化点播播放器（VodPlayer）和 TRTC 同时使用时的音量大小和音质表现。
- iOS&Android：增加对 wav 音频格式的背景音乐和音效文件的支持。
- Windows：优化在某些低端摄像头下 CPU 使用率过高的问题。
- Windows：优化对多款 USB 摄像头和麦克风的兼容性，提升设备的打开成功率。
- Windows：优化摄像头和麦克风设备的选择策略，避免由于摄像头或麦克风在使用中插拔导致的采集异常问题。

修复

- 全平台：修复弱网情况下调用 muteLocalVideo 和 muteLocalAudio 接口时会偶现播放异常的 BUG。
- iOS：修复播放音效在低端 iPhone 或 iPad 上可能会失败的 BUG。
- iOS：修复 iPad Pro 屏幕分享出的画面出现变形拉伸的问题。
- iOS：修复 App 内屏幕共享在用户拒绝权限之后，还会持续弹出几次屏幕录制权限申请提示的问题。
- Windows：解决笔记本或者台式机在长时间休眠后，退房 onExitRoom 事件通知不会回调的问题。
- Windows：修复在 Music 音质模式下，开启系统混音 stopSystemAudioLoopback 后会导致漏回声的问题。
- Windows：修复在快速调用 enterRoom 和 exitRoom 进退房的情况下，偶现的播放端无声的 BUG。
- Windows：修复 SDK 对 Visual Studio 2010 项目的编译兼容性问题。
- Windows：修复手动接收模式（即 setDefaultStreamRecvMode(false，false)）下会重复收到 onUserVideoAvailable 事件回调的问题。

Version 7.6 @ 2020.08.21

新增

- Windows：新增 updateLocalView 和 updateRemoteView 接口，用于优化实时调整 HWND 类型的渲染窗口时的体验。
- Windows：新增 getCurrentMicDeviceMute 接口用于获取当前 Windows PC 是否被设置为静音。
- Windows：新增 setCurrentMicDeviceMute 接口用于将当前 Windows PC 设置为全局静音。
- Mac：新增 updateLocalView 和 updateRemoteView 接口，用于优化实时调整 View 渲染区域时的体验。
- Mac：新增 getCurrentMicDeviceMute 接口用于获取当前 Mac 电脑是否被设置为静音。
- Mac：新增 setCurrentMicDeviceMute 接口用于将当前 Mac 电脑设置为全局静音。

版权所有：腾讯云计算（北京）有限责任公司

- iOS：新增 updateLocalView 和 updateRemoteView 接口，用于优化实时调整 View 渲染区域时的体验。
- iOS: 为 TRTCCloudDelegate 增加了 onCapturedRawAudioFrame 回调，并修改了其他几个回调函数的名称，依次修改为 onLocalProcessedAudioFrame 、onRemoteUserAudioFrame 和 onMixedPlayAudioFrame 。
- Android：为 TRTCCloudListener 增加了 onCapturedRawAudioFrame 回调，并修改了其他几个回调函数的名称，依次修改为 onLocalProcessedAudioFrame 、onRemoteUserAudioFrame 和 onMixedPlayAudioFrame 。

优化

- 全平台：优化 enterRoom 的协议策略，提升加入房间的速度，并提升成功率。
- 全平台：优化同时订阅超多路音频时的总体性能消耗和卡顿问题。
- Mac：屏幕分享开始支持分享指定窗口的指定区域。

修复

- 全平台：修复在不退房的情况下进入同一个房间时，SDK 不触发 onEnterRoom 返回的 BUG。
- 全平台：修复几种可能导致黑屏的偶现内部 BUG 的问题。
- 全平台：修复提前调用 startRemoteSubStreamView 无法正常显示屏幕分享画面的问题。
- Windows：修复已知的几处句柄及 GDI 泄露。
- Windows：修复多个已知的 crash 问题。
- Windows：修复摄像头和麦克风拔掉后重新插入不会自动开启设备的问题。
- iOS：修复在 iOS 10 上背景音乐接口在传入特定规则的文件路径时会崩溃的 BUG。
- Android：修复频繁快速的 enterRoom 和 exitRoom 后偶现的无声问题。
- Android：修复偶现的录屏推流黑屏的问题。

Version 7.5 @ 2020.07.31

新增

- 新增对双栈 IPV6 和 IPV6 only 的支持。
- 新增进多房间拉流能力，用于支持超级小班课。
- 云端 MCU 混流新增支持设置背景图片（由于监管需要，图片必须先通过 TRTC 控制台进行上传）。
- 云端 MCU 混流新增支持 A+B=>C 和 A+B=>A 两种模式。
- 实时状态回调 onStatistics 新增播放缓冲时长字段 jitterBufferDelay。

优化

- 降低了端到端的连麦延时，7.5 版本的端到端通话和连麦延时在 7.4 版本的基础上缩短了40%。
- 降低了移动端的耳返延时，并支持对耳返设置变声和混响等音效。
- 优化播放端网络抖动评估算法，降低播放延迟。
- 降低 Android SDK 的端到端连麦通话延时。
- 进一步优化耳返时延。
- 优化播放 view 动态切换时画面黑屏的问题。

修复

- 修复在一个函数中连续调用 playBGM 和 pauseBGM 后播放不生效的问题。
- 修复偶现退房之后还能收到 onEnterRoom 回调的问题。
- 修复部分机型对超低分辨率编码失败无法恢复的问题。

Version 7.4 @ 2020.06.24

优化

耳返支持音量设置。

修复

- 修复 Android 版本横竖屏切换时本地画面闪一下的问题。
- 修复部分 Android 手机发送自定义视频无法正常编码的问题。
- 修复音频处理时偶发的一处数据包处理崩溃。

Version 7.3 @ 2020.06.01

新增

- 在兼容老接口的情况下，增加了全新的音效管理接口 TXAudioEffectManager，用于支持更加灵活和多样的音效能力。
- 视频编码参数 setVideoEncoderParam 新增 minVideoBitrate 选项，推荐对画质要求高的直播客户进行设置。

优化

- 音频新增瞬态降噪支持，您可以通过 setAudioQuality(TRTCAudioQualitySpeech) 开启。
- 音效文件支持 asset 打包的音效文件。
- 提升本地视频清晰度。
- 播放端自定义渲染支持纹理的方式，降低性能开销。
- 优化摄像头采集分辨率选取逻辑，提升视角效果。
- 优化了回声处理效果。
- 支持全链路 128kbps 高音质立体声，通过 setAudioQuality(TRTCAudioQualityMusic) 接口即可设置。
- 支持 SPEECH 语音模式，适合会议场景下的语音通话，拥有更强的降噪（ANS）能力，通过 setAudioQuality(TRTCAudioQualitySpeech) 可以设置。
- 支持多路背景音乐并行播放，用于支持原声和伴唱分离的 K 歌场景。同时支持背景音乐循环播放。
- 支持先调用 muteLocalVideo 再调用 startLocalPreview 实现“只预览，不推流”的效果，您也可以通过在 enterRoom 前调用 startLocalPreview 实现该能力。

修复

- 修复自定义视频采集时，偶现 SDK 内部 OpenGL 上下文错误 crash。
- 修复进房前 setLocalVideoRenderListener 自定义渲染回调不触发的问题。
- 修复横屏模式下切换前后摄像头，播放端画面会倒置的问题。
- 修复进房前调用 startLocalPreview，进房后播放端概率花屏问题。
- 修复硬编码器偶现 crash。
- 修复本地音频录制偶现的断断续续的 Bug。
- 修复暂停推流（muteLocalVideo，muteLocalAudio）时，发生强杀或 crash 后重进房，播放端不会自动播放音视频的问题。

Version 7.2 @ 2020.04.16

新增

新增 Android 支持手机录屏，适用于手机端录屏直播。

优化

- 优化中低端 Android 手机在通话场景下的性能消耗，提升语音体验。
- 优化滤镜、绿幕等视效接口，归并到 TXCBeautyManager 类下，实现统一的调用方式。

版权所有：腾讯云计算（北京）有限责任公司

修复

修复切换角色时，自定义流 ID 偶现未及时生效的问题。

Version 7.1 @ 2020.03.27

优化

- C++ STL基础库全静态编译。
- 通话音量默认开启 ANS、AGC，提高通话模式下的音质。
- 优化混流预设模板易用性。
- 混流优化，提升成功率。

修复

- 修复进房频繁开关 AGC 的时候，处理声音变成全零的问题。
- 修复测速导致其他 API 调用响应较慢的问题。
- 修复被系统电话打断后上行音量翻倍及声音有噪音问题。
- 修复进房自动旁路的问题。

Version 7.0 @ 2020.03.09

- 优化 3A 开启策略。
- 提升 mcu 混流易用性。
- 优化弱网抗抖动能力，弱网下，音频更流畅。
- 解决多次交替进退房导致的内存泄露问题。

Version 6.9 @ 2020.01.14

新增

- 新增对 Android 10.0 系统的支持。
- 新增 API： snapshotVideo()，支持本地及远端视频画面截图。
- 新增 API：pauseAudioEffect、resumeAudioEffect 音效支持暂停/恢复控制。
- 新增 API：setBGMPlayoutVolume、setBGMPublishVolume，BGM 支持分别设置本地播放和推流混音音量。
- 新增 API：setRemoteSubStreamViewRotation 辅路视频播放支持调整渲染旋转角度。
- 新增一种全局音量类型模式：setSystemVolumeType(TRTCSystemVolumeTypeVOIP)，支持一直采用通话音量，主要用于解决蓝牙耳机自带麦克风的采集切换问题。
- enterRoom 参数 TRTCParams 中新增 streamId 属性，用于设定当前用户在 CDN 上的直播流 ID，更方便您绑定直播 CDN。
- enterRoom 参数 TRTCParams 中新增 cloudRecordFileName 属性，您可以设置本次直播在云端录制的文件名。
- 新增场景 TRTCAppSceneAudioCall，在 enterRoom 时可以设置。该场景下，TRTC SDK 针对语音通话进行了全方位的优化。
- 新增场景 TRTCAppSceneVoiceChatRoom，在 enterRoom 时可以设置，可以开启 TRTC SDK 专门针对语音互动聊天室场景所作的各项优化。

优化

- 优化录制服务对视频流中断的抵抗能力，使得远程录制的文件更加完整。
- 优化某些机型硬解时音画不同步的问题。
- 视频画面支持 1080P 高分辨率采集，让手机直播 PC 观看的场景获得更佳的画面清晰度。
- 优化错误码，简化进房错误码。
- 优化偶现秒开慢的问题。

修复

- 修复偶现 HTTP 组件 crash。
- 修复音效播放偶现没有完成回调的问题。
- 修复偶现进房失败后无法恢复的问题。

Version 6.8 @ 2019.11.15

新增

- 新增耳返能力。
- 新增进房可指定不自动拉流。
- 新增接口 getBeautyManager，聚合美颜、P 图动效接口。
- 企业版新增 P 图新功能，包括美肤、亮眼、白牙、祛皱、祛眼袋等新特性。
- 新增回调 onRemoteUserEnterRoom / onRemoteUserLeaveRoom，支持未上麦的主播进退房通知。

优化

- pts 生成机制优化。
- 优化网络切换后，自动选择较优的接入点。
- startRemoteView 支持提前调用。

修复

修复已知 crash 等稳定性问题。

Version 6.7 @ 2019.09.30

新增

- AAR 打包新增权限获取配置。
- 新增 Android 8.0 以上系统 CPU 占用评估。

优化

- 转推耗时优化。
- 支持单个用户播放音量独立调节能力。

Version 6.6 @ 2019.08.02

新增

- 新增音频本地录制功能。
- 新增首帧音频、首帧视频发送回调接口。
- 新增系统音量类型设置接口。
- 新增音效接口，支持播放短音效。
- 音频自定义回调接口输出的数据支持可修改。

优化

- 进房优化，降低进房耗时，提升进房成功率。
- 支持 mute 远端视频接口。
- 进房错误码统一，通过 onEnterRoom 回调，result<0 表示进房错误。
- Demo 优化，新增低延时大房间支持。

- 播放器新增音量设置接口及音量大小回调接口。
- 自定义发送视频支持本地渲染。
- 自定义采集发送视频支持 1080P。
- 本地及远端渲染支持 SurfaceView 方式。

修复

- 修复旁路混流相关的问题。
- 修复本地预览角度不对的问题。

Version 6.5 @ 2019.06.12

新增

直播模式（TRTCAppSceneLIVE）新增“低延时大房间”功能：

- 采用专为音视频优化过的 UDP 协议，超强抗弱网能力。
- 平均观看延迟一秒作为，提升观众和主播之间的互动积极性。
- 最多支持10万人进入同一个房间。

优化

- 优化弱网下音画不同步的 Bug。
- 优化 onStatistics 状态回调，仅回调存在的流。
- 优化直播 TXLivePlayer 播放缓冲逻辑，降低卡顿率。
- 优化先 muteLocalVideo 之后再取消播放端画面的恢复速度。
- 优化高延迟和高丢包网络环境下的 QoE 算法，增强弱网抗性。
- 优化解码器性能，修复超低端 Android 手机上延迟越来越高的 Bug。
- 优化音量评估算法（enableAudioVolumeEvaluation），音量评估更灵敏。
- 优化视频通话（TRTCAppSceneVideoCall）模式下的 QoE 算法，进一步提升 1v1 通话模式下的弱网流畅性。

修复

- 修复偶现的 enterRoom 没有回调的 Bug。
- 修复关闭音频采集之后，播放也没有声音的 Bug。
- 修复移除后再添加本地渲染 view 之后绿屏的 Bug。
- 修复自定义渲染回调（setRemoteVideoRenderDelegate），远端画面在分辨率是 540P 以上（包括 540P）时只回调10次的 Bug。

Version 6.4 @ 2019.04.25

新增

- 新增本地显示镜像和编码器输出镜像接口。
- 新增混流 setMixTranscodingConfig API 的设置回调函数。
- 新增企业版支持大眼、瘦脸、V 脸和动效挂架功能。

优化

- 提升弱网环境下的流畅度。
- 优化音量大小的回调算法，音量回调数值更加合理。
- 发送自定义音频、视频数据支持外部指定数据帧时间戳。
- 强化 setMixTranscodingConfig 接口，支持 roomID 参数，用于跨房连麦流混流。

- 强化 setMixTranscodingConfig 接口，支持 pureAudio 参数，用于纯语音通话场景下的语音混流和录制。
- 优化低端 Android 设备上解码 720p 视频的性能问题。

修复

- 修复声音免提切换无效 Bug。
- 修复直播（TXLivePlayer）延时可能会升高且不恢复的 Bug。
- 修复直播场景 setVideoEncoderRotation 无效的 Bug。
- 修复 Android 禁用麦克风权限后，没有错误回调 Bug。
- 修复 Android 9.0 系统上 Demo 打开后弹窗的问题。
- 修复音量调节按钮无法调整观众端声音大小的问题。

Version 6.3 @ 2019.04.02

新增

- 新增 Android 平台64位的支持。
- 新增自定义视频采集接口：TRTCCloud >> sendCustomVideoData。
- 新增自定义音频采集接口：TRTCCloud >> sendCustomAudioData。
- 新增自定义视频渲染接口：TRTCCloud >> setLocalVideoRenderDelegate + setRemoteVideoRenderDelegate。
- 新增自定义音频数据回调接口：TRTCCloud >> setAudioFrameDelegate，支持：
  - 返回麦克风采集数据：TRTCAudioFrameDelegate >> onCapturedAudioFrame。
  - 返回每一路远程用户的音频数据：TRTCAudioFrameDelegate >> onPlayAudioFrame。
  - 返回混合后送入喇叭播放的音频数据：TRTCAudioFrameDelegate >>onMixedPlayAudioFrame。

Version 6.2 @ 2019.03.08

新增

- 增加滤镜浓度设置接口 setFilterConcentration() 。
- 新增 sendSEIMsg() 接口，支持通过视频帧中的 SEI 头信息发送自定义消息，一般用于在视频流中塞入时间戳信息。
- 新增跨房间通话能力 connectOtherRoom，即已存在的两个 TRTC 房间可以相互连通，该功能可用于直播间中的主播 PK 功能。

优化

- 优化 CPU 使用率和稳定性。
- 提升弱网（即较差的网络环境）下的画面清晰度。
- 取消 TRTCCloud 的多实例能力，创建模式改为单例模式，避免多个 TRTCCloud 实例相互抢占网络资源，影响体验效果。

修复

修复纯语音通话场景（例如狼人杀）下的旁路推流功能，需要配合 TRTCParam 中的 bussInfo 字段使用。

Version 6.1 @ 2019.01.31

优化

- 支持观看屏幕分享流 。
- 支持发送自定义视频数据 。
- 优化转推 CDN 和混流实现 。
- 进房区分直播和视频通话场景 。
- 提升稳定性，解决一些偶现 crash 。

- 优化流控，提升弱网表现。

Version 6.0 @ 2019.01.18

优化

- 更新架构为 LiteAV 内核 。
- 采用全新 QoS 算法，更低的卡顿率，更高的流畅性 。
- 采用全新的 Audio 模块，深度优化了各种网络情况下的声音质量 。
- 支持大小流双路编码功能（推荐仅在 Windows 和 Mac 设备上开启） 。
- 支持 CDN 转推和混流功能。

发布日志（Web&H5）

最近更新时间：2025-06-17 14:24:22

版本号 `major.minor.patch` 具体规则如下：

- major：主版本号，如有重大版本重构则该字段递增，通常各主版本间接口不兼容。
- minor：次版本号，各次版本号间接口保持兼容，如有接口新增或优化则该字段递增。
- patch：补丁号，如有功能改善或缺陷修复则该字段递增。

> ⚠ **注意：**
>
> - 建议您及时更新至最新版本，以便获得更好的产品稳定性及在线支持。
>
> 版本升级注意事项请参见：升级指引 。
> - 5.x 与 4.x 版本的区别 。
> - 4.x 版本的发布日志 。

Version 5.11.0 @2025.06.13

Feature

- 支持 trtc.switchRoom API。对于 'live' 场景的观众角色，在弱网场景切换房间耗时可降低 40%。
- 支持 trtc.startLocalVideo 设置 rotate 参数，控制摄像头旋转角度。
- 支持 本地视频混流插件，可以将摄像头 + 屏幕分享在本地混合后推流。
- 支持 虚拟背景插件 人脸自动居中。
- 支持根据下行网络情况自动切换大小流，相关文档可参见：开启大小流。
- 支持 AUDIO_FRAME 事件回调远端音频数据。
- 支持 STATISTICS 事件回调端到端延迟。

Improvement

- 提升小流性能和稳定性（在小流拉流黑屏时，SDK 自动切换到大流）。
- 规避 Chrome 137(部分版本) 屏幕分享系统音频回声 bug，需要：
  - 开通 AI 降噪套餐包。
  - 部署 npm 包中的 assets 目录，SDK 需要用到 `audioProcessor-wasm.js` 。
  - `TRTC.create({ assetsPath })` 传入您的 cdn 资源路径。

Bug Fixed

- 修复偶现重连后未恢复推流的问题。
- 修复开启美颜或者虚拟背景插件时，mute/unmute 会导致拉流黑屏问题。
- 修复特定场景下，TRTC 多实例事件通知异常的问题。
- 修复 iOS 系统打断后，摄像头渲染异常问题。
- 修复部分安卓手机无法编码 120p 小流问题。

Version 5.10.1 @2025.05.15

Bug Fixed

- 修复开启小流/美颜/虚拟背景后，切换摄像头偶现黑屏的问题。
- 修复与 WebAR SDK 共用时，更新摄像头分辨率失效问题。
- 修复 iPad Chrome 访问桌面版网站时，无法拉流的问题。
- 修复使用 receiveWhenViewVisible 参数时，偶现黑屏问题。

Version 5.10.0 @2025.04.17

Breaking Changed

- 插件产物格式由 iife 转变为 umd，支持了更多接入环境。若您通过 script 标签加载插件，升级时需要注意。
- npm 包中的 wasm 资源文件统一迁移至 assets 目录，您部署时可以直接部署 assets 目录。

Feature

- 支持美声插件。

Improvement

- 优化软解插件降级策略。
- 优化 Android H264 检测逻辑。
- 优化耳返逻辑，耳返播放前处理后的音频。

Bug Fixed

- 修复复用 WebSocket 时，换 userId 进房出现报错的问题。
- 修复 PC Chrome 360p 采集时，画幅被裁切问题，需指定参数规避：
  `TRTC.startLocalVideo({ option: { avoidCropping: true }})`

Version 5.9.2 @2025.04.10

Improvement

- trtc.startRemoteVideo 新增 poster 参数。
- 优化弱网通话体验。

Version 5.9.1 @2025.03.07

Bug Fixed

- 修复 iOS 中被其他应用播放声音打断后无声的问题。
- 修复页面切后台后，音频回调频率不均匀的问题。
- 修复动态更新分辨率可能导致虚拟背景失效的问题。

Version 5.9.0 @2025.02.17

Feature

- 支持 视频解码插件，提升解码成功率。
- 支持 Android 设备切换听筒和扬声器，参考：TRTC.setCurrentSpeaker。
- 新增视频首帧回调事件，参考：FIRST_VIDEO_FRAME。

Improvement

- 降低秒开耗时，若您期望获得非常流畅的秒开体验，请联系 技术支持。
- 优化音量计算逻辑，支持在 http 协议下获取音量。
- AUTOPLAY_FAILED 事件回调参数新增 resume 方法。

Bug Fixed

- 修复 iOS 低概率无声问题。
- 修复 Safari & Firefox 发送 sei 丢消息的问题。
- 修复特定场景 autoplay failed 事件未触发的问题。
- 修复 muteRemoteAudio 静音偶现无效的问题。
- 修复在不支持 h264 的环境下，无法进行纯语音通话的问题。

Version 5.8.6 @2024.12.27

Improvement

- 优化弱网重连逻辑，提升重连成功率。
- 优化 setRemoteAudioVolume 接口，支持在拉流之前设置音量。
- 规避部分荣耀机型拉流无声问题。
- 规避 iOS 16 & 18 开启水印后，内存泄露导致页面崩溃的问题。

Bug Fixed

- 修复屏幕分享 + 系统音频时无声问题。
- 修复 iOS 16 动态更新分辨率后，偶现渲染画面出现拉伸的问题。
- 修复在 iOS WKWebview 中，偶现 AudioContext 报错的问题。
- 修复在切换麦克风后，audio-frame 事件的 pcm 空数据的问题。

Version 5.8.5 @2024.12.03

Breaking Changed

Debug 插件变为内置插件，无需外部引入。参见 Debug 插件 。

Feature

- 支持从主流推屏幕分享。
- Debug 插件支持 Dump 音视频能力。
- CDNStreaming 插件支持混流回推。

Improvement

- 提升音量回调的准确性和兼容性。

Bug Fixed

- 修复偶现分享系统音频后，上行无声的问题。
- 修复 Android X5 内核加载 SDK 报错的问题。
- 修复在 iOS 16 使用 WebAR SDK 的兼容性问题。
- 修复特定场景下频繁进退房出现无声问题。
- 修复跨房连麦时，mute/unmute 画面渲染异常问题。
- 修复开启虚拟背景时，切换前后台导致渲染卡顿问题。

Version 5.8.3 @2024.10.25

Feature

新增 AUDIO_FRAME 事件，可获取麦克风原始 PCM 数据。

Improvement

- 提升 Safari 中视频渲染性能
- 支持在用户未点击页面的情况下，获取麦克风音量。

Bug Fixed

- 修复采集设备时可能采集到非指定设备的问题。
- 修复 CDNStreaming 插件在未填写 roomId 时可能出现混流异常的问题。

Version 5.8.2 @2024.10.11

Improvement

- 降低进房耗时。
- 提升虚拟背景插件的稳定性。

Bug Fixed

- 修复在采集系统音频情况下，关闭屏幕分享会导致上行无声的问题。
- 修复特定场景下跨房连麦插件无法重新启动的问题。
- 修复偶现自定义消息丢失的问题。

版权所有：腾讯云计算（北京）有限责任公司

Version 5.8.1 @2024.09.12

Feature

- 支持在 Safari 和 Firefox 中收发 SEI 消息。
- 新增 VirtualBackground.isSupported 方法检测环境是否支持使用虚拟背景插件。

Improvement

- 优化类型声明文件。
- 提升虚拟背景插件兼容性（Firefox 和 Safari）。

Bug Fixed

- 修复偶现自定义消息丢失的问题。
- 修复 Firefox 中偶现观看视频黑屏的问题。
- 修复 Safari 中推视频流最高只有 15fps 的问题。
- 修复 Safari 16 中移除摄像头 view 会导致推流停止的问题。

Version 5.8.0 @2024.08.23

Feature

- 支持 跨房连麦插件。
- 支持 设备检测插件。
- 支持 Debug 插件。

Bug Fixed

- 修复开启 SEI 时偶现推流失败的问题。
- 修复 Firefox 屏幕分享分辨率与预期不符的问题。

Version 5.7.1 @2024.08.07

Bug Fixed

- 修复 iOS 17 中视频渲染异常的问题。
- 修复 unifiedProxy 失效的问题。

Version 5.7.0 @2024.07.19

Feature

- 支持 基础美颜插件 。
- 支持辅流 sei 消息收发。

Improvement

- 使用 captureElement 参数采集屏幕分享时，预览框只显示当前页面。
- 规避 Android Webview 插拔耳机可能导致无声的问题，chrome issue。
- 提升 Android 设备 h264 支持度检测的准确性。

Bug Fixed

修复在支持 h264 解码，但不支持 h264 编码的环境中，远端视频黑屏的问题。

Version 5.6.3 @2024.06.28

Feature

- 混音插件支持监听播放进度回调。
- 虚拟背景插件支持设置虚化程度。

Improvement

- 提升 iOS 音频自动播放的成功率。

- 提升 iOS 通话打断后，恢复正常通话的成功率。

Bug Fixed

- 修复 Chrome 91 以下版本偶现重连的问题。
- 修复 mute/unmute audio 后，音频播放出现延迟的问题。
- 修复特定场景下，误抛远端推屏幕分享事件的问题。

Version 5.6.2 @2024.06.07

Improvement

- 优化观众拉流通话体验，减少卡顿。
- 提升断网重连成功率。

Bug Fixed

- 修复移动端偶现自动播放失败后，无法恢复播放的问题。
- 修复 Mac Safari 采集1920*1280分辨率失败的问题。
- 修复偶现拉流无声的问题。
- 修复特定调用顺序下 muteRemoteAudio 误报 abort error 的问题。

Version 5.6.1 @2024.05.23

Bug Fixed

修复在手动拉音频的场景下，偶现无声的问题。

Version 5.6.0 @2024.05.17

Breaking Changed

关闭默认进房自动拉视频流，由业务侧按需调用 trtc.startRemoteVideo 来拉流播放视频。参见：升级指引。

Feature

支持收发自定义消息，参见：trtc.sendCustomMessage。

Bug Fixed

- 修复在 iOS 12.0 版本进房失败的问题。
- 修复编码镜像偶现失效的问题。
- 修复混流转推 CDN 收不到流的问题。
- 修复混音插件 loop 不生效的问题。
- 修复在混用 roomId 和 strRoomId 多次进房后，进房异常的问题。

Version 5.5.2 @2024.04.29

Improvement

- 优化背景虚化插件资源加载逻辑，可通过自行部署模型文件加快加载速度，参见：开启虚拟背景。
- 规避 iOS 17 播放视频闪烁的问题，webkit bug。

Bug Fixed

- 修复 Android qq 浏览器偶现 startLocalVideo 失败的问题。
- 修复 Chrome 123+ 版本订阅小流时，偶现报错的问题。
- 修复偶现播放黑屏问题。

Version 5.5.1 @2024.04.12

Improvement

- 提升弱网重连成功率。
- 提升设备恢复采集成功率。

Bug Fixed

- 修复 iOS 15.1 版本播放远端视频没有显示的问题。
- 修复使用 updateScreenShare 重新推屏幕分享后，从辅流变成主流的问题。
- 修复 high audio profile 码率不符合预期的问题。
- 修复水印插件传入 base64 无法渲染的问题。

Version 5.5.0 @2024.03.29

Improvement

- 优化移动端 AI 降噪插件的兼容性。
- 提升设备异常自动恢复采集的成功率。

Bug Fixed

- 修复 iOS 16 中偶现本地视频播放黑屏的问题。
- 修复 iOS 14 偶现无声的问题。
- 修复 startLocalAudio 偶现报错问题。

Version 5.4.3 @2024.03.15

Feature

- 支持 混音插件 传入 MediaStreamTrack。
- 支持 trtc.getAudioTrack 获取屏幕分享音频 MediaStreamTrack。

Bug Fixed

- 修复偶现 setRemoteAudioVolume 0 不生效的问题。
- 修复 updateScreenShare({ publish: true }) 后，屏幕分享音频没有推流的问题。
- 修复 Safari 中无法开启虚拟背景的问题。

Version 5.4.2 @2024.03.01

Bug Fixed

- 修复偶现 startRemoteVideo 失败的问题。
- 修复偶现取消推流失败的问题。
- 修复部分低端机型出现音画不同步的问题。
- 修复 Nginx 代理无法进房的问题。

Version 5.4.1 @2024.02.05

Improvement

优化重连逻辑，提升弱网重连成功率。

Bug Fixed

- 修复 updateLocalVideo 导致 mirror 失效的问题。
- 修复 CONNECTION_STATE_CHANGED 事件没有 CONNECTING 状态的问题。

Version 5.4.0 @2024.01.16

Feature

- 支持获取视频帧，参见 getVideoSnapshot()。
- 支持 mute video 时设置图片垫片，参见 startLocalVideo() 的 mute 参数。
- 支持只拉可视区域视频流，非可视区域的视频不拉流。参见 多人视频通话最佳实践。
- 支持屏幕分享采集页面某个 DOM 元素。参见 startScreenShare()。

Improvement

- 优化进房逻辑，降低进房耗时。
- 优化笔记本盒盖重新打开时的重连逻辑。

Bug Fixed

- 修复 Chrome 69 以下版本偶现推流失败的问题。
- 修复 iOS 13 & 14 1080P 推流黑屏问题。

Version 5.3.2 @2023.12.26

Feature

- 支持水印插件，参考：开启水印。
- 支持编码翻转，参考：startLocalVideo() 的 mirror 参数。

Improvement

- 优化背景虚化插件性能，减少 CPU 占用。
- 优化音视频编码稳定性，提升编码质量。

Bug Fixed

- 修复 CDNStreaming 插件的已知问题。
- 修复 setRemoteAudioVolume 0 后，音量事件返回的音量值为 0 的问题。
- 修复部分外接麦克风开启降噪偶现丢音的问题。

Version 5.3.1 @2023.12.08

Bug Fixed

- 修复混音插件异常问题。
- 修复 Chrome 74 以下版本无法进房问题。
- 修复开启 AI 降噪时，部分音频接口表现不符合预期的问题。
- 修复多 trtc 实例场景下，销毁其中一个实例，其他实例收不到 DEVICE_CHANGED 事件的问题。

Version 5.3.0 @2023.12.01

Feature

- 支持 SEI 消息收发，可用于实现歌词同步、直播答题等功能，参考 sendSEIMessage。
- 支持动态开关大小流，参考 updateLocalVideo 的 option.small 参数。
- 支持静音推流，参考 startLocalAudio() 的 mute 参数。
- 支持切换角色时更新 privateMapKey，参考 switchRole 的 privateMapKey 参数。
- 新增 TRTC.EVENT.TRACK 事件。

Improvement

- 优化进房流程，缩短进房耗时。
- 优化高分辨率通话场景、低版本 Android Chrome 设备的编码质量。
- 优化获取设备逻辑，在无媒体访问权限的情况下，SDK 可能会暂时请求获取媒体权限，以保证能正常获取到媒体设备，随后会释放媒体设备。
- 优化混音插件的 url 参数的解析逻辑。
- 提升 AI 降噪插件的降噪效果。

Bug Fixed

- 修复 Android Chrome 无法编码 120p 的问题。
- 修复屏幕分享不推流场景下，停止屏幕分享会导致摄像头推流停止的问题。
- 修复 CDN 混流插件参数失效问题。

Version 5.2.1 @2023.11.08

Feature

- startLocalAudio & updateLocalAudio 接口新增 captureVolume 参数，支持调节麦克风采集音量。
- 移动端设备支持 TRTC.EVENT.DEVICE_CHANGED 事件，可实现在连接耳机时，自动切换到耳机麦克风。参考：处理设备插拔 。

Bug Fixed

- 修复切换麦克风后无声的问题。
- 修复停止屏幕分享时，TRTC.EVENT.PUBLISH_STATE_CHANGED 事件的 mediaType 类型错误的问题。

Version 5.2.0 @2023.10.31

Feature

- 支持 背景虚化插件。
- 新增 TRTC.EVENT.STATISTICS 事件，可用于获取通话相关统计数据。

Improvement

- 提升设备采集成功率。
- 优化“画中画模式”的镜像处理逻辑。
- 当用户系统拒绝浏览器授权时，可调用 `RtcError.handler()` 跳转至系统授权设置，引导用户快速打开授权。参考错误码：5302。

Bug Fixed

修复低版本 Chrome 偶现拉流无声问题。

Version 5.1.3 @2023.09.11

Feature

trtc.setRemoteAudioVolume 支持设置音量大于 100，可以增益远端播放音量。

Improvement

规避 iOS 15.1 切换摄像头导致页面崩溃的 iOS bug。

Bug Fixed

- 修复 Firefox 取消推流后再重新推流失效的问题。
- 修复 Firefox 采集特定分辨率视频失败的问题，例如：640 * 360。
- 修复偶现远端视频未播放的问题。

Version 5.1.2 @2023.08.25

Improvement

降低进房耗时。

Bug Fixed

- 修复 webpack 打包构建 trtc.esm.js 偶现报错的问题。
- 修复 startLocalAudio 传入自定义采集 audioTrack 不生效的问题。

Version 5.1.1 @2023.08.18

Improvement

- 默认 video profile 变更为 480p_2，在保障画质的情况下，降低上行带宽消耗。
- 规避 Android Chrome 115 低于 360p 分辨率偶现无法编码的 Chrome Bug。

Bug Fixed

- 修复 Chrome 57、iOS 12 无法进房推拉流的问题。
- 修复仪表盘视频码率异常问题。

Version 5.1.0 @2023.08.11

Breaking Change

限制 trtc.enterRoom 接口的 roomId 参数为 number 类型，不再支持传入 string 类型。若要使用字符串房间号，请使用 strRoomId 参数。升级时需注意，详情参考：升级指引 。

Feature

- 支持背景音乐插件，参考教程：实现背景音乐。
- 支持 AI 降噪插件，参考教程：实现 AI 降噪。
- 支持 CDN 混流插件，参考教程：实现云端混流及转推 CDN。

Bug Fixed

- 修复设置屏幕分享采集分辨率不生效的问题。
- 修复偶现播放远端屏幕分享失败的问题。

Version 5.0.3 @2023.07.31

Improvement

优化重连机制，提升网络连接稳定性。

Bug Fixed

修复 trtc.stopRemoteVideo 停止播放主流时，辅流也被停止播放的问题。

Version 5.0.2 @2023.07.21

Improvement

- 优化多人音视频场景下的性能表现及弱网抗性。
- 优化设备采集逻辑，规避部分联想设备无法开启摄像头的问题。
- 优化屏幕分享采集参数，规避长时间屏幕分享偶现采集掉帧的问题。

Bug Fixed

- 修复小流码率设置不生效的问题。
- 修复 systemAudio 参数不生效的问题。
- 修复偶现远端用户关闭屏幕分享后，video 标签未销毁的问题。

Version 5.0.1 @2023.06.25

Feature

支持同时在多个位置播放视频

Bug Fixed

修复点击浏览器悬浮窗口关闭屏幕分享后，无法重新开启屏幕分享的问题。

Version 5.0.0 @2023.05.26

TRTC Web SDK 新架构版本，提供扁平化的接口，大幅简化 API，降低接入成本。新 API 的特点：

- 更易于接入的扁平化 API。
- 更好的稳定性。
- 更好的性能表现。

发布日志（Electron）

最近更新时间：2025-02-11 09:11:12

Version 12.3.703 @ 2025.1.20

功能优化

升级 底层库 。

问题修复

修复偶现的因日志打印失败，导致崩溃问题。

Version 12.2.703 @ 2025.1.10

功能新增

- 修改 TRTCMediaMixingManager.setDisplayParams() 接口，入参支持传入 DOM 节点，本地混流合图时，新增媒体源排版功能。新增 TRTCMediaMixingEvent 事件类型。
- 新增直播推流类 V2LivePusher，与 直播 SDK 互通。

问题修复

- 修复跨进程模式下（isIPCMode 为 true）， setLogLevel 接口不支持问题。
- 修复关闭视频时，播放器未清理干净，导致达到浏览器 WebMediaPlayer 上限，视频渲染黑屏问题。
- 修复使用音频自定义处理插件时内存泄漏问题。
- 修复跨进程模式下（isIPCMode 为 true），设置摄像头镜像无效的问题。

Version 12.2.702 @ 2024.12.12

功能新增

- 支持新的 CDN 转推、混流接口：startPublishMediaStream 、updatePublishMediaStream 和 stopPublishMediaStream 。
- 新增本地混流管理类 TRTCMediaMixingManager 及创建接口 TRTCCloud.getMediaMixingManager() 。

功能优化

优化部分接口控制台日志打印没有时间戳问题。

问题修复

- 修复窗口从隐藏变为显示时，视频渲染短暂卡顿问题。
- 修复 TRTCCloud.destroyTRTCShareInstance() 销毁共享实例后，再调用 TRTCCloud.getTRTCShareInstance() 获取实例后，调用设备相关接口偶现崩溃问题。

Version 12.2.701 @ 2024.11.26

功能优化

- 优化设备管理、音效管理、插件管理内部实现。
- 完善 onLocalRecordComplete 接口错误码说明。

问题修复

- 修复 Mac 下偶现刷新页面后，SDK 事件回调丢失问题。

- 修复调用 setAudioFrameCallback 接口后无事件回调问题。
- 修复音频录制接口 startAudioRecording 分段录制参数设置不生效问题。

Version 12.1.607 @ 2024.11.8

问题修复

修复 setLogCallback 接口不可用问题。

Version 12.1.607 @ 2024.9.26

功能新增

新增音效接口 enableVoiceEarMonitor、setVoiceEarMonitorVolume、setVoiceChangerType、setVoiceCaptureVolume、setVoicePitch。

功能优化

优化视频渲染。

问题修复

- 修复 onWarning 事件中 extraInfo 参数转换异常问题。
- 修复获取屏幕/窗口列表接口 getScreenCaptureSources 返回窗口坐标错误问题。

Version 12.0.606 @ 2024.8.26

功能新增

优化接入方式，支持在工程的 package.json 文件中配置需要下载的二进制库类型，详细说明请参阅 npm 仓库中的 README 文件。

Version 12.0.605 @ 2024.8.8

功能新增

- 底层 SDK 版本从 11.9 升级 12.0， 详细变更请参阅 SDK发布日志 。
- API 文档：新增"如何查询和申请设备权限"。

问题修复

API 文档：更新关于 onSystemAudioLoopbackError 事件错误描述。

Version 11.9.605 @ 2024.7.24

功能新增

- 底层 SDK 版本从 11.8 升级到 11.9，详细变更请参阅 SDK发布日志 。
- TRTCPluginType 新增 TRTCPluginTypeAudioProcess 类型，支持音频自定义处理插件。
- Windows 端支持 onSystemAudioLoopbackError 事件回调。

Version 11.8.603 @ 2024.5.29

功能新增

- 增加 setAudioFrameCallback 接口。
- TRTCVolumeInfo 结构体增加 pitch 字段。

功能改进

优化本地上行视频渲染性能。

问题修复

修复调用 setMixTranscodingConfig 接口时设置的 URL 为空的问题。

Version 11.7.603 @ 2024.4.8

问题修复

- 修复获取设备列表时偶现崩溃问题。
- 修复 onStatistic 接口统计数据缺少内存字段问题

Version 11.7.602 @ 2024.3.27

功能新增

- 增加 setAudioQuality 接口。
- startMicDeviceTest 接口支持全平台的 playback 参数。
- TRTCMixInputType 增加水印类型。

Version 11.6.601 @ 2024.3.7

功能新增

- 新增 setMusicObserver 接口，用于注册背景音乐播放事件监听；修改 startPlayMusic 接口，之前用于背景音乐监听的第二个入参已废弃，后续停止支持，请尽快改为使用 setMusicObserver 接口注册监听。
- 新增 createSubCloud 接口，支持创建子实例。

功能改进

- 远端用户退房时，本地不再自动销毁远端用户的视频渲染器，视频画面停留在最后一帧。在用户调用 stopRemoteView 接口结束流订阅或者退房时，远端用户的视频渲染器才会销毁。
- 视频渲染性能优化。

Version 11.4.503 @ 2023.11.22

功能新增

新增 setVoiceReverbType 接口，支持设置人声混响效果。

功能改进

优化性能。

问题修复

修复 Mac 下，千人大房间场景，拉流到100多路后，出现拉流失败，视频显示黑屏问题。

Version 11.4.502 @ 2023.9.25

功能新增

- 新增接口 setRemoteAudioParallelParams 接口，支持设置远端音频流智能并发播放策略。
- 新增音频自定义采集接口：enableCustomAudioCapture 、sendCustomAudioData 、enableMixExternalAudioFrame 、mixExternalAudioFrame 、setMixExternalAudioVolume 、generateCustomPTS 。

- 新增接口 addIncludedShareWindow、removeIncludedShareWindow、removeAllIncludedShareWindow，支持分享屏幕时指定要分享的窗口。

功能改进

- 优化视频渲染功能，startLocalPreview、updateLocalView、startRemoteView、updateRemoteView 接口支持传入多个 HTML 节点，对同一路视频多次渲染。
- 优化 getScreenCaptureSources 接口，Mac 下返回值新增 width 和 height 字段，之前仅支持 Windows。

问题修复

- 修复 Mac 下，npm install trtc-electron-sdk 安装依赖后，开发模式启动，找不到 "TXFFmpeg" 动态库问题。
- 修复视频渲染时，偶现 canvas 画布崩溃问题。
- 修复调用 startRemoteView 接口订阅远端用户小流视频时，如果远端用户未开启双路编码，无法渲染默认的大流视频问题。
- 修复 Windows 下分享 WPS 窗口，点幻灯片放映不会自动切换成全屏窗口的问题。

Version 11.0.501 @ 2023.6.30

功能改进

- API 文档新增 onStartPublishing 和 onStopPublishing 两个事件。
- API 文档完善 onScreenCapturePaused 接口字段说明。
- Electron SDK 支持 Linux（beta 版）。

Version 10.9.405 @ 2023.4.17

功能新增

- 新增接口 setCameraCaptureParams，支持设置摄像头采集参数，目前仅支持 Windows。
- 新增接口 setVideoMuteImage，支持摄像头 mute 后，设置垫片图片。
- 新增接口 enableFollowingDefaultAudioDevice，支持扬声器、麦克风跟随当前系统设备。
- 修改接口 setMixTranscodingConfig，支持设置每路视频流的输入类型、渲染模式、垫片图片。
- 修改接口 getScreenCaptureSources，返回值新增 isMainWindow 字段，目前仅支持 Windows。

问题修复

修复应用层刷新页面，偶现应用奔溃问题。

Version 10.7.405 @ 2023.2.27

功能新增

- 新增接口 updateLocalView 和 updateRemoteView，支持修改页面上视频观看、预览的位置。
- Windows 下，getScreenCaptureSources 接口返回的屏幕、窗口信息新增 isMainScreen 字段。

问题修复

- 修复房间中多人屏幕分享时，不能同时渲染观看问题。
- 视频渲染 DOM 元素缩放从 transform scale 改为 zoom，保持向后兼容。
- 解决频繁开关摄像头，本地预览偶现绿屏帧问题。
- Mac 下音视频软解的动态库从实体文件改外链接文件，解决 Mac 下构建应用包时出现的签名异常。
- 解决 setRemoteRenderParams 接口设置的 fillMode 参数在 startRemoteView 之前调用不生效问题。

Version 10.3.406 @ 2023.2.4

功能新增

Windows 下，getScreenCaptureSources 接口返回的屏幕、窗口信息新增 isMainScreen 字段。

问题修复

- Windows 下，启动麦克风时，若麦克风为 mute 状态，主动取消 mute。
- 修复 selectScreenCaptureTarget 接口设置高亮边框不生效问题。
- Windows 下，视频渲染流程优化。

Version 10.3.405 @ 2022.12.12

问题修复

修复发现的一些易用性问题。

Version 10.7.404 @ 2022.10.31

功能新增

- 设置水印接口 setWaterMark 支持 Windows 系统；Mac 下新增支持通过图片路径设置水印。
- 推流到非腾讯云 CDN 接口 startPublishCDNStream 接口新增 streamId 入参，支持设置流 ID。

Version 10.6.404 @ 2022.10.31

功能新增

- 设置水印接口 setWaterMark 支持 Windows 系统；Mac 下新增支持通过图片路径设置水印。
- 推流到非腾讯云 CDN 接口 startPublishCDNStream 接口新增 streamId 入参，支持设置流 ID。

问题修复

- 修复调用 setRemoteVideoStreamType() 切小流视频渲染卡住问题。
- 修复视频渲染偶现绿屏帧问题。
- 修复 Mac 下摄像头检测不支持镜像问题，解决 Mac 下首次分享窗口不出现高亮绿框问题。
- 修复 Mac 下分享窗口、屏幕时上行帧率为零的问题，导致远端用户收不到 onUserSubStreamAvailable 事件问题。

Version 10.3.404 @ 2022.10.31

问题修复

修复调用 setRemoteVideoStreamType() 切换小流视频渲染卡住问题。

Version 10.6.403 @ 2022.09.09

功能新增

Windows & Mac: 新增本地媒体录制接口，支持直播时，将本地音视频数据录制保存到本地文件。具体接口包括：startLocalRecording、stopLocalRecording、onLocalRecordBegin、onLocalRecording、onLocalRecordComplete。

功能修改

Windows&Mac：废弃 setRenderMode 接口，不再支持调用此接口修改视频的默认渲染方式（WebGL 或 Canvas 2D），SDK 内部会自动选择合适的渲染方式，以提高视频渲染性能。

功能改进

- 视频渲染性能优化。
- 升级底层库。Mac 下支持构建 ARM64 指令集的应用程序，发挥 M1 芯片优势，提升性能。

Version 10.3.402 @ 2022.08.12

问题修复

- Window & Mac：调用混流接口后，混流事件返回 -3324 user id invalid 报错问题。

Version 10.3.401 @ 2022.07.20

功能改进

- 视频渲染性能优化。
- 升级底层库。

Version 9.3.201 @ 2022.01.05

功能新增

Windows & Mac：新增 onSpeedTestResult 网速测试的结果回调。

改进

- Windows & Mac：改进 startSpeedTest 开始进行网络测速。
- Windows & Mac：改进 muteLocalVideo 暂停/恢复发布本地的视频流，新增 streamType 参数。
- Windows & Mac：改进 muteRemoteVideoStream 暂停接收指定的远端视频流，新增 streamType 参数。
- Windows & Mac：改进 startScreenCapture 启动屏幕分享，新增 params 参数。

问题修复

- Mac：Mac OS 12 新系统下的摄像头采集问题。
- Windows & Mac：优化弱网调控策略，同场景下更流畅。
- Windows：优化 AGC 算法，降低出现声音过小和声音过大的问题的概率。
- Windows：修复屏幕分享时采集帧率异常的问题。

Version 8.9.102 @ 2021.08.11

功能新增

Windows & Mac：onStatistics 回调新增字段 gatewayRtt onStatistics 。

问题修复

- Mac：修复特殊机型写日志引起 crash。
- Mac：修复禁麦的操作使用 API 接口 setAudioCaptureVolume(0) 后，发现麦克风检测音量为 0。
- Windows：性能优化，修复打开摄像头后黑屏。
- Windows：修复屏幕捕获自动减低分辨率后不恢复。
- Windows & Mac：其他 bug 修复。

Version 8.6.101 @ 2021.05.28

功能新增

- Windows & Mac：新增接口，支持屏幕分享时屏蔽应用窗口：addExcludedShareWindow 、removeExcludedShareWindow 、removeAllExcludedShareWindow 。

- Windows & Mac：获取可共享的窗口列表接口 getScreenCaptureSources，返回值列表元素新增 isMinimizeWindow 字段。
- Windows & Mac：支持构造函数传入参数。

问题修复

- Windows：修复不支持加载包含中文路径的插件问题。
- Windows & Mac：修复 webgl context lost 问题。
- Windows & Mac：开启双路编码，进入房间后，切换小画面的视频流，本地显示的远端成员画面卡住问题。
- Windows & Mac：在客户端进入房间拉流的时候出现远端成员画面先模糊一下，然后逐渐清晰的问题。

Version 8.4.1 @ 2021.03.26

功能新增

- Mac：开始支持采集 Mac 操作系统的输出声音 startSystemAudioLoopback，也就是跟 Windows 端一样的 SystemLoopback 能力，该功能可以让 SDK 采集当前系统的声音，开启这个功能后，主播就可以很方便地向其他用户直播音乐或者电影文件。
- Mac：系统音频采集回调 onSystemAudioLoopbackError，您可以获取系统音频驱动的运行情况。
- Mac：屏幕分享开始支持本地预览功能，您可以通过一个小窗口像用户展示屏幕分享的预览内容。
- 全平台：支持美颜插件机制。

质量优化

- 全平台：优化 Music 模式下的声音质量，更加适合类似 cloubhouse 的语音直播场景。
- 全平台：优化音视频链路的网络抗性，在 70% 的极端网络环境下，音视频依然较为流畅。
- Windows：优化部分场景下的直播音质，大幅减少了声音损伤问题。
- Windows：性能优化，在部分使用场景下的性能较旧版本有 20%-30% 的提升。

问题修复

- Mac：修复 Mac mini (m1) 换到全屏分享后，再切回某个窗口，远端还是展示的全屏分享窗口的问题。
- Mac：解决 Mac 下屏幕分享无高亮的问题（Mac 系统 11.1，10.14.5 不出现绿框；Mac 系统 10.3.2 会出现绿框，但放大窗口时，会出现视频画面闪烁的问题）。
- Mac：修复 Mac mini M1 获取分享屏幕列表 crash，针对底层 sourceName 为 null 时上层返回""的问题。
- Mac：修复 Mac mini M1，getCurrentMicDevice 导致 crash (sourceName) 可能为空问题。
- Windows：修复 Windows Server 2019 Datacenter x64 系统上启动桌面分享 crash 的问题。
- Windows：修复分享窗口的同时改变目标窗口大小会偶发分享意外终止的 BUG。
- Windows：修复部分型号的摄像头采集不出画面的问题。

Version 8.2.7 @ 2021.01.06

新增

- Windows & Mac：新增 switchRoom 切换房间。
- Windows & Mac：新增 setLocalRenderParams 设置本地图像（主流）的渲染参数。
- Windows & Mac：新增 setRemoteRenderParams 设置远端图像的渲染参数。
- Windows & Mac：新增 startPlayMusic 启动播放背景音乐。
- Windows & Mac：新增 stopPlayMusic 停止播放背景音乐。
- Windows & Mac：新增 pausePlayMusic 暂停播放背景音乐。
- Windows & Mac：新增 resumePlayMusic 恢复播放背景音乐。
- Windows & Mac：新增 getMusicDurationInMS 获取背景音乐文件总时长，单位毫秒。

- Windows & Mac：新增 seekMusicToPosInTime 设置背景音乐播放进度。
- Windows & Mac：新增 setAllMusicVolume 设置背景音乐的音量大小，播放背景音乐混音时使用，用来控制背景声音音量大小。
- Windows & Mac：新增 setMusicPlayoutVolume 设置背景音乐本地播放音量的大小。
- Windows & Mac：新增 setMusicPublishVolume 设置背景音乐远端播放音量的大小。
- Windows & Mac：新增 onSwitchRoom 切换房间回调。
- Windows & Mac：新增 setRemoteAudioVolume 设置远程用户播放音量。
- Windows & Mac：新增 snapshotVideo 视频截图。
- Windows & Mac：新增 onSnapshotComplete 完成截图时回调。

改进

- Windows & Mac：enterRoom 和 switchRoom 支持 string 类型 strRoomId。
- Windows & Mac：其他 bug 修复。

Version 7.9.348 @ 2020.11.12

改进

- Windows：修复录音文件路径不支持中文的问题。
- Windows：窗口捕获指定区域支持抗遮挡。

Version 7.8.342 @ 2020.10.10

新增

- Windows & Mac：新增 onAudioDeviceCaptureVolumeChanged 当前音频输入设备音量变化回调。
- Windows & Mac：新增 onAudioDevicePlayoutVolumeChanged 当前音频播放设备音量变化回调。

Version 7.7.330 @ 2020.09.11

新增

Windows & Mac：新增 setAudioQuality 用于设置音频质量。

改进

- Windows：优化在某些低端摄像头下 CPU 使用率过高的问题。
- Windows：优化对多款 USB 摄像头和麦克风的兼容性，提升设备的打开成功率。
- Windows：优化摄像头和麦克风设备的选择策略，避免由于摄像头或麦克风在使用中插拔导致的采集异常问题。
- Windows & Mac：其他 bug 修复。

Version 7.6.300 @ 2020.08.26

新增

Windows & Mac：新增 setCurrentMicDeviceMute 、getCurrentMicDeviceMute、setCurrentSpeakerDeviceMute、getCurrentSpeakerDeviceMute 用于控制 PC 的麦克风和扬声器。

Version 7.5.210 @ 2020.08.11

改进

- Windows & Mac：修复 SDK 回调乱序问题。
- Windows & Mac：解决切换渲染模式导致崩溃的问题。

- Windows & Mac：修复某些分辨率渲染失败的问题。
- Windows & Mac：其他 bug 修复。

Version 7.4.204 @ 2020.07.01

改进

- Windows：优化 Windows 平台的回声抵消（AEC）效果。
- Windows：增强 Windows 平台的摄像头采集的设备兼容性。
- Windows：增强 Windows 平台的音频设备（麦克风和扬声器）的设备兼容性。
- Windows：修复 Windows 版本 onPlayAudioFrame 回调的 UserID 不正确的问题。
- Windows：64 位支持系统混音

Version 7.2.174 @ 2020.04.20

改进

- Mac：修复 Mac 偶现本地自定义渲染分辨率不一致问题。
- Windows：优化 Windows 端 getCurrentCameraDevice 逻辑，在未使用摄像头时，返回第一个设备作为默认设备。
- Windows：修复高亮窗口在屏幕分享时显示为灰屏的问题。
- Windows：修复 Win10 系统获取屏幕分享缩略图偶现卡死问题。
- Windows & Mac：修复切换角色时，自定义流 ID 偶现未及时生效的问题。
- Windows & Mac：修复屏幕分享设置编码参数不生效的问题。
- Windows：修复 Windows 端屏幕分享后，webrtc 要很久才能看到画面的问题。

Version 7.1.157 @ 2020.04.02

新增

支持使用 主路 进行 屏幕分享 。

改进

- 优化 混流预设模版 易用性。
- 优化 混流，提升成功率。
- 优化 Windows 屏幕分享。

Version 7.0.149 @ 2020.03.019

新增

trtc.d.ts 文件，方便使用 typescript 的开发者。

发布日志（uni-app）

最近更新时间：2023-02-07 09:50:29

此页面仅更新 TRTC SDK 的版本历史，具体接口使用请参见 API 文档。

Version 1.1.0 @ 2022.12.30

新特性

- 新增 设置视频编码器的编码接口 setVideoEncoderParam，可以用来实现横屏推流。
- 新增 背景音乐接口：开始播放背景音乐 startPlayMusic、停止播放背景音乐 stopPlayMusic、暂停播放背景音乐 pausePlayMusic、恢复播放背景音乐 resumePlayMusic。
- 新增 背景音乐事件：背景音乐开始播放事件 onStart；背景音乐的播放进度事件 onPlayProgress；背景音乐已经播放完毕事件 onComplete。
- 新增 音视频技术指标的实时统计回调 onStatistics 事件。
- 新增 用户视频大小发生改变回调 onUserVideoSizeChanged 事件。
- 支持 HBuilder 选择 vue3 打包使用。

功能优化

优化调用 startLocalPreview 抛出 -2 错误问题。

Version 1.0.9 @ 2022.09.27

新特性

- iOS 支持 setLocalRenderParams 设置本地渲染（旋转、填充模式及左右镜像）。
- 支持 setRemoteRenderParams 设置远端渲染（旋转、填充模式及左右镜像）。

Version 1.0.8 @ 2022.08.08

功能优化

优化抛出的错误信息，增加常见 错误码链接。

缺陷修复

修复 Android 无法云端录制问题。

Version 1.0.7 @ 2022.07.22

新特性

- 补充 错误码、警告码 文档。
- 补充 类型定义 文档。
- 增加 onUserSubStreamAvailable 事件。

缺陷修复

修复 Android 美颜设置没变化问题。

Version 1.0.6 @ 2022.07.04

新特性

- 增加 switchRole 角色切换接口。

- 增加 switchCamera 前置、后置摄像头切换。
- 增加 setLocalRenderParams 设置本地画面的渲染参数，可设置的参数包括有：画面的旋转角度、填充模式以及左右镜像。
- 增加 muteLocalVideo 暂停/恢复发布本地的视频流。
- 增加截屏 snapshotVideo 接口。
- 增加 mute audio 接口，
  - muteLocalAudio 静音/恢复本地的音频
  - muteRemoteAudio 静音/恢复某一个用户的声音
  - muteAllRemoteAudio 静音/恢复所有用户的声音。
- 增加启用或关闭音量大小提示接口 enableAudioVolumeEvaluation 。
- 增加接口 setSubStreamEncoderParam 用设置屏幕分享（即辅路）的视频编码参数。
- 增加屏幕分享接口 startScreenCapture 。
- 增加美颜相关接口 setBeautyLevel 、setBeautyStyle 。

Version 1.0.5 @ 2022.06.24

新特性

- 支持角色切换 switchRole 及对应的事件监听 onSwitchRole 。
- 支持静音某一个用户的声音 muteRemoteAudio 。
- 静音掉所有用户的声音 muteAllRemoteAudio 。

Version 1.0.4 @ 2022.06.03

缺陷修复

iOS 原生插件兼容到 iOS9.x。

Version 1.0.3 @ 2022.05.11

功能优化

解决 videoView 未创建时，调用 startRemoteView 报错问题。

Version 1.0.2 @ 2022.04.13

新特性

增加 setAudioRoute 切换音频路由接口，用于切换“听筒” 和 “免提”。

Version 1.0.1 @ 2022.04.02

缺陷修复

iOS 修复采用字符串房间号进房问题。

Version 1.0.0 @ 2022.04.01

新特性

发布 uni-app 音视频 SDK。提供基本的音视频通话，接口使用详见 官方文档 。

发布日志（Flutter）

最近更新时间：2024-09-03 18:25:12

Version 2.8.6 @ 2024.8.12

依赖更新

MacOS SDK 更新至 12.0.16292。

新特性

- MacOS：新增 enableFollowingDefaultAudioDevice 接口。
- MacOS：新增 startSystemAudioLoopback 接口，支持系统音频捕获，例如第三方音乐播放器。
- Android&iOS&Windows&MacOS：新增 startSpeedTestWithParams 接口和 onSpeedTestResult 回调。

缺陷修复

iOS：iOS上 onStatistics 回调参数解析字段与 Android 保持一致，receivedBytes 改为 receiveBytes 。

Version 2.8.5 @ 2024.8.2

依赖更新

Windows SDK 更新至 12.0.1.6002。

Version 2.8.4 @ 2024.8.1

依赖更新

Windows SDK 更新至 12.0.0.15124。

新特性

Windows：新增 enableFollowingDefaultAudioDevice 接口。

Version 2.8.3 @ 2024.7.22

依赖更新

- Android SDK 更新至 11.9.0.14466。
- iOS SDK 更新至 11.9.15963。

Version 2.8.2 @ 2024.6.20

新特性

- Android&iOS：新增 onAudioRouteChanged 回调。
- 添加新的音频路由如 TRTC_AUDIO_ROUTE_WIREDHEADSET，详情请见： TRTCCloudDef 。

缺陷修复

Windows：修复 onDeviceChange 回调无法收到的问题。

Version 2.8.1 @ 2024.6.14

新特性

- Android&iOS: 新增 setVoicePitch 接口
- 添加新的变声特效 Studio 2，详情请见 TXVoiceReverbType 。

Version 2.8.0 @ 2024.5.21

新特性

Windows: 新增 getScreenCaptureSources 和 selectScreenCaptureTarget 接口，支持 Windows 屏幕分享。

Version 2.7.9 @ 2024.5.20

依赖更新

- Android SDK 更新至 11.8.0.14188
- iOS SDK 更新至 11.8.15687

Version 2.7.8 @ 2024.4.18

依赖更新

- Windows SDK 更新至 11.7.0.14863。
- MacOS SDK 更新至 11.7.15304。
- Android SDK 更新至 11.7.0.13910。
- iOS SDK 更新至 11.7.15343。

新特性

Android&iOS：新增 createSubCloud 和 destroySubCloud API。

缺陷修复

Windows：修复了 onRecvCustomCmdMsg 回调数据解析错误问题。

Version 2.7.7 @ 2024.4.3

缺陷修复

Web：修复了 switchRole 调用无效的问题。

Version 2.7.6 @ 2024.2.29

缺陷修复

Windows：修复了 Window 库中的数据上传问题。

Version 2.7.5 @ 2024.2.29

新特性

Windows：更新 startSystemAudioLoopback 接口以支持收集特定应用程序的声音。

Version 2.7.4 @ 2024.2.29

新特性

Windows：新增 startSystemAudioLoopback 接口，支持系统音频捕获，如第三方音乐播放器。

Version 2.7.3 @ 2024.1.15

缺陷修复

Web：修复了由于引入 dart:ffi 导致的 Web 平台编译失败问题。

Version 2.7.2 @ 2024.1.11

依赖更新

Window：升级客户端 SDK 版本至 11.4.0。

Version 2.7.1 @ 2023.12.28

缺陷修复

macOS：修复了在 TexTure 渲染过程中偶尔出现的崩溃问题。

新特性

Android&iOS：新增 setAudioFrameListener API。

Version 2.7.0 @ 2023.12.13

新特性

Web：将 Web TRTC SDK 升级到最新版本（v5），以实现更稳定的功能。

缺陷修复

Web：修复了从 Android 和 iOS 设备进行屏幕共享时，无法在 Web 版本中查看的问题。

Version 2.6.0 @ 2023.11.21

缺陷修复

Web：修复了在调用 getCurrentDevice 和 getDevicesList API 时出现的异常问题。

Version 2.5.9 @ 2023.9.28

新特性

- Android 和 iOS：新增 startPublishMediaStream API。
- Android 和 iOS：新增 updatePublishMediaStream API。
- Android 和 iOS：新增 stopPublishMediaStream API。

Version 2.5.8 @ 2023.9.13

新特性

将文档跳转链接替换为国际站点。

Version 2.5.7 @ 2023.9.11

依赖更新

- Android SDK 更新至 11.4.0.13189。
- iOS SDK 更新至 11.4.14445。
- macOS SDK 更新至 11.4.14445。

Version 2.5.6 @ 2023.8.9

缺陷修复

Windows：优化 Dart 代码风格。

Version 2.5.5 @ 2023.8.2

依赖更新

- Android SDK 更新至 11.3.0.13200。
- iOS SDK 更新至 11.3.14354。
- macOS SDK 更新至 11.3.14333。

Version 2.5.4 @ 2023.7.10

依赖更新

- Android SDK 更新至 11.3.0.13176。
- iOS SDK 更新至 11.3.14333。

Version 2.5.3 @ 2023.6.27

依赖更新

- Android SDK 更新至 11.2.13145。
- iOS SDK 更新至 11.2.14217。

Version 2.5.2 @ 2023.6.16

缺陷修复

- Windows：修复了 startSpeedTest 函数返回过长数据事件且无响应的问题。
- Web：将 setAudioPlayoutVolume 和 getAudioPlayoutVolume 标记为不可用。

Version 2.5.1 @ 2023.6.2

新特性

- Windows&Mac&Web：恢复对 Windows&Mac&Web 平台的支持。
- iOS：新增 setSystemAudioLoopbackVolume API，支持在屏幕共享时调整系统音量。

缺陷修复

iOS：修复了在特定场景下 TRTCCloudVideoView 偶尔出现的内存泄漏问题。

依赖更新

Windows&Mac：将客户端 SDK 版本升级至 11.1.0。

Version 2.5.0 @ 2023.5.4

功能优化

暂时移除了 Web、MacOS 和 Windows 平台的支持。

Version 2.4.6 @ 2023.5.4

依赖更新

- Android SDK 更新至 11.1.0.13111。
- iOS SDK 更新至 11.1.14143。

Version 2.4.5 @ 2023.3.14

新特性

Android：新增 startSystemAudioLoopback 功能。

Version 2.4.4 @ 2023.3.6

功能优化

优化部分代码。

Version 2.4.2 @ 2023.1.9

依赖更新

Android SDK 更新至 10.9.0.24004。

Version 2.4.2 @ 2023.1.9

缺陷修复

修复 iOS 平台 snapshotVideo 为空的问题。

Version 2.4.1 @ 2022.12.1

依赖更新

- Android SDK 更新至 10.8.0.13065。
- iOS SDK 更新至 10.8.12025。

Version 2.4.0 @ 2022.10.31

功能优化

优化部分代码。

Version 2.3.9 @ 2022.10.18

依赖更新

- Android SDK 更新至 10.7.0.13053。
- iOS SDK 更新至 10.7.11936。

Version 2.3.8 @ 2022.9.20

功能优化

优化 Windows 平台，自动添加相关 DLL 文件。

Version 2.3.7 @ 2022.9.16

缺陷修复

修复 Web 平台 "Can't use 'Function' as a name" 的问题。

Version 2.3.6 @ 2022.9.5

功能优化

优化部分代码。

Version 2.3.5 @ 2022.8.23

依赖更新

- Android SDK 更新至 10.3.0.11196。
- iOS SDK 更新至 10.3.12231。

Version 2.3.4 @ 2022.7.21

新特性

- 更新 Windows、MacOS 和 Web 平台，支持纯视频模式。
- setMixTranscodingConfig：仅支持混合视频。

Version 2.3.2 @ 2022.7.14

依赖更新

Android 和 iOS SDK 更新至 10.3。

Version 2.3.1 @ 2022.6.23

功能优化

优化部分代码。

Version 2.3.0 @ 2022.6.20

依赖更新

Android 和 iOS SDK 更新至 10.1 版本的 LiteAVSDK_Professional。

新特性

支持第三方美颜。

Version 2.2.4 @ 2022.5.11

功能优化

优化 setMixTranscodingConfig 接口。

Version 2.2.3 @ 2022.5.7

依赖更新

- Android SDK 设置为 9.9.0.11820。
- iOS SDK 设置为 9.5.11346。

Version 2.2.2 @ 2022.5.5

功能优化

更新日志模块。

Version 2.2.1 @ 2022.4.21

功能优化

PlatformView 支持 'onTap' 事件。

Version 2.2.0 @ 2022.3.30

依赖更新

将 Android 和 iOS SDK 更新为 TXLiteAVSDK_Live。

Version 2.1.7 @ 2022.3.22

缺陷修复

修复 2.1.6 版本 iOS 视频渲染的问题。

Version 2.1.6 @ 2022.3.17

功能优化

优化 iOS 纹理。

Version 2.1.5 @ 2022.3.11

功能优化

更新 remoteView 调整参数顺序。

Version 2.1.4 @ 2022.3.10

功能优化

更新文档。

Version 2.1.3 @ 2022.3.10

功能优化

更新文档。

Version 2.1.2 @ 2022.3.7

依赖更新

- Android SDK 更新至 9.5.11207。
- iOS SDK 更新至 9.5.11207。

Version 2.1.1 @ 2022.2.15

缺陷修复

修复 onSpeedTest 回调数据错误的问题。

版权所有：腾讯云计算（北京）有限责任公司

Version 2.1.0 @ 2022.1.25

功能优化

优化初始化时机。

Version 2.0.9 @ 2022.1.13

缺陷修复

修复未找到 web 文件夹的问题。

依赖更新

Android 和 iOS SDK 更新至 9.5。

Version 2.0.7 @ 2022.1.10

功能优化

解决警告。

Version 2.0.6 @ 2022.1.10

功能优化

删除 web 文件夹。

Version 2.0.5 @ 2022.1.10

功能优化

优化文档显示。

Version 2.0.1 @ 2022.1.7

功能优化

将视频纹理渲染封装成 PlatformView。

Version 2.0.0 @ 2022.1.4

功能优化

支持流畅的 Web。

Version 1.3.1 @ 2022.1.4

功能优化

优化部分文档。

Version 1.3.0 @ 2021.11.22

功能优化

Android 视频 SurfaceView 渲染更改为 GLSurfaceView。

Version 1.2.9 @ 2021.11.15

依赖更新

将底层 Android SDK 版本更新为9.3.10765。

Version 1.2.8 @ 2021.11.15

功能优化

Android 的底层 GLSurfaceView 被 TextureView 替换，updateView 仅支持 TextureView。

Version 1.2.7 @ 2021.11.5

功能优化

屏幕共享支持指定大小的流。

Version 1.2.6 @ 2021.11.1

缺陷修复

修复前一版本导致的 iOS 视频渲染问题。

Version 1.2.5 @ 2021.10.27

新特性

Android 视频渲染支持混合集成模式。默认模式为虚拟显示模式。TRTCCloudVideoView 的视图模式传输给 TRTCCloudDef.TRTC_VideoView_Model_Hybrid。

Version 1.2.4 @ 2021.9.29

功能优化

- Fluent 的 Windows 端支持纹理渲染。
- 在线提供 Fluent 英文文档。

Version 1.2.3 @ 2021.9.28

新特性

Android支持 updateLocalView 和 updateRemoteView 接口。

Version 1.2.2 @ 2021.9.10

缺陷修复

修复 Android 纹理渲染多人切换视频内存增长问题。

Version 1.2.1 @ 2021.9.10

功能优化

优化部分功能。

Version 1.2.0 @ 2021.9.10

新特性

指定美颜、设备和音效管理模块的返回值。

Version 1.1.9 @ 2021.8.19

缺陷修复

修复 iOS 和 MacOS 截图失败等问题。

Version 1.1.8 @ 2021.8.12

功能优化

Android 纹理渲染兼容 meglhelper。

Version 1.1.7 @ 2021.8.10

缺陷修复

修复 Android 缺少 businessInfo 字段的问题。

Version 1.1.6 @ 2021.8.3

功能优化

优化部分功能。

Version 1.1.5 @ 2021.8.3

缺陷修复

修复在 iOS 和 MacOS 的发布模式下播放音乐时出现特殊字符串参数导致的崩溃问题。

Version 1.1.4 @ 2021.8.3

缺陷修复

修复 iOS 和 MacOS 下特殊字符串参数导致的崩溃问题。

Version 1.1.3 @ 2021.7.27

缺陷修复

- 修复 iOS 和 MacOS 下 onspeedtest 回调没有网络质量数据的问题。
- 修复 Android 纹理渲染 meglcore 为空的问题。

Version 1.1.2 @ 2021.7.23

缺陷修复

修复 iOS 和 MacOS 不支持辅助流渲染的问题。

Version 1.1.1 @ 2021.7.21

新特性

新的纹理渲染形式。

Version 1.1.0 @ 2021.7.14

缺陷修复

修复 Android 停止远程视图后再次启动远程视图视频无法渲染的问题。

Version 1.0.9 @ 2021.6.30

新特性

Android 和 iOS 支持本地录制 startLocalRecording 。

Version 1.0.8 @ 2021.6.28

新特性

支持 Windows 和 macOS。目前仅支持音频相关接口，视频渲染暂不支持。

Version 1.0.5 @ 2021.6.9

缺陷修复

修复 Android 视频视图销毁后出现的平台异常错误。

Version 1.0.4 @ 2021.6.2

新特性

iOS 添加 updateLocalView 和 updateRemoteView 接口。

Version 1.0.3 @ 2021.6.1

缺陷修复

修复 Android 关闭麦克风后设置音频路由无效的问题。

Version 1.0.2 @ 2021.5.27

功能优化

修改文档注释。

Version 1.0.1 @ 2021.4.28

缺陷修复

修复 Android 房间 ID 超过 2147483647 时无法进入房间的问题。值范围支持：1 – 4294967294。

Version 1.0.0 @ 2021.4.23

功能优化

升级 Flutter 2.0，支持零安全。

跑通 API-Example
iOS

最近更新时间：2025-03-10 17:58:32

本文介绍了如何快速运行腾讯云 TRTC iOS SDK Demo。

环境准备

- Xcode 11.0及以上版本。
- 确保您的项目已设置有效的开发者签名。
- Qt Creator 4.13.3 (Mac) 及以上版本。

操作步骤

步骤1. 下载 Demo

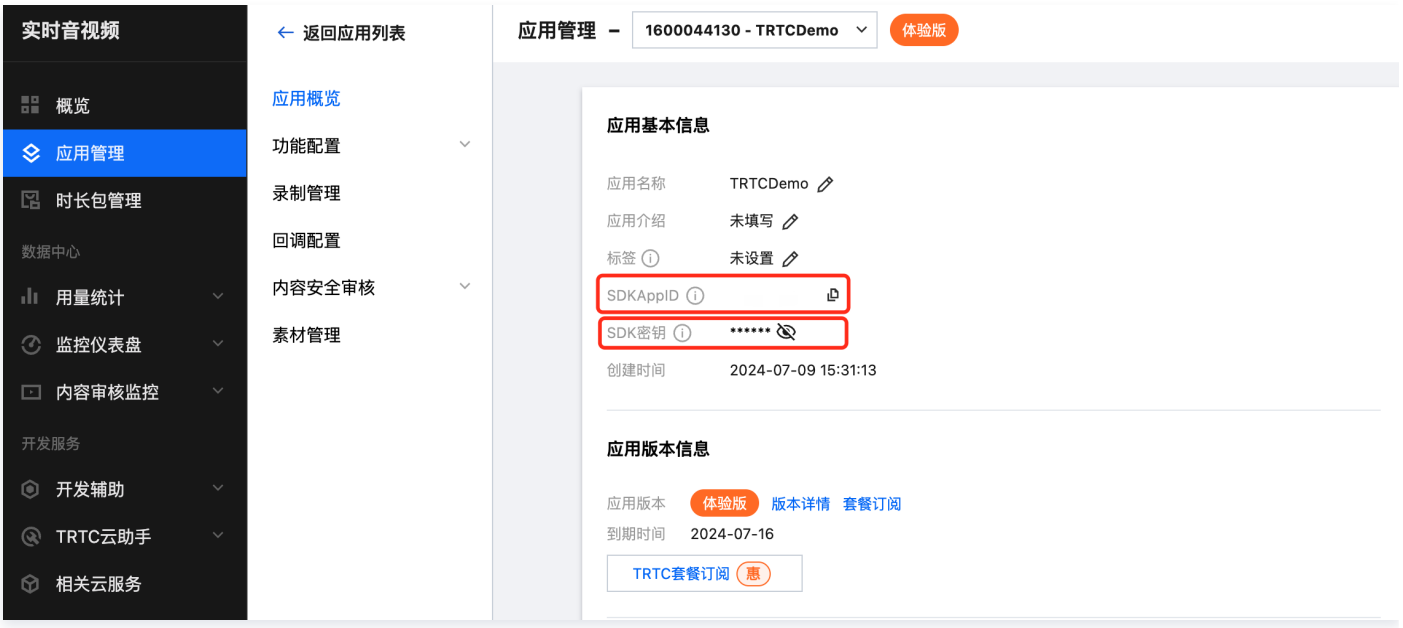
您可以在 github 下载 iOS 平台的示例代码，或者在终端运行以下命令：

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

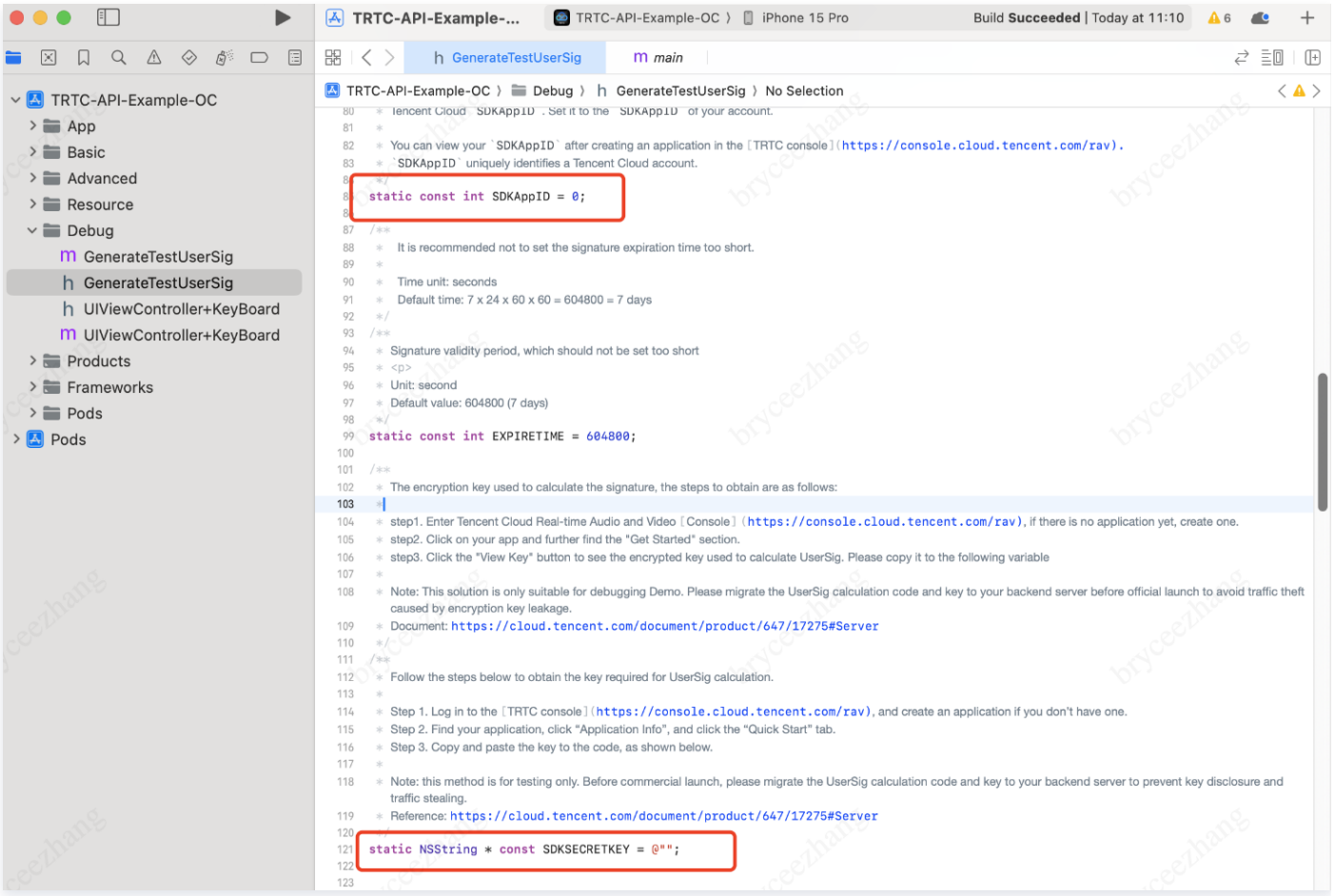
在终端窗口中进入目标工程目录后执行 `pod install` 命令即可，导入 iOS 平台 SDK 中其他步骤已经在示例代码中操作过。

步骤2. 配置 Demo

1. 登录 实时音视频控制台，单击**创建应用**。如果您已经完成创建，可以跳过该操作。
2. 在创建应用成功后，您可以在**应用管理**中获取到您的 **SDKAppID** 和 **SDK 密钥**。



3. 您需要将 `TRTC-API-Example-XX/Debug` 目录下 `GenerateTestUserSig.h` 或 `GenerateTestUserSig.swift` 中的 **SDKAPPID** 和 **SDKSECRETKEY** 替换为您在**步骤2**中获取到的值。



注意：

- 本文使用的生成 UserSig 的方案是在本地配置 SDKSecretKey，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，因此**该方法仅适合本地跑通 Demo 和功能调试**。正确的 UserSig 签发方式是在您的服务端集成 服务端生成 UserSig，当用户进房时：
  - 发起 http 请求到您的服务端。
  - 服务端生成一个 UserSig。
  - 将其返回给用户供进房使用。
- 若要将 Demo 部署到公网体验，需要通过 HTTPS 协议，即 `https://域名/xxx` 访问，原因可参见文档 页面访问协议限制说明。

步骤3. 跑通 Demo

使用 XCode（11.0及以上的版本）打开源码目录下的 `TRTC-API-Example-OC.xcworkspace` 工程，编译并运行 TRTC-API-Example 工程即可。

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。
- 如果您是开发者，也欢迎您加入我们的 RTC Engine 技术交流平台 zhiliao，进行技术交流和产品沟通。

Mac

最近更新时间：2024-07-09 16:21:31

本文介绍了如何快速运行腾讯云 TRTC Mac SDK Demo。

环境准备

- Xcode 11.0及以上版本。
- 确保您的项目已设置有效的开发者签名。
- Qt Creator 4.13.3 (Mac) 及以上版本。

操作步骤

步骤1. 下载 Demo

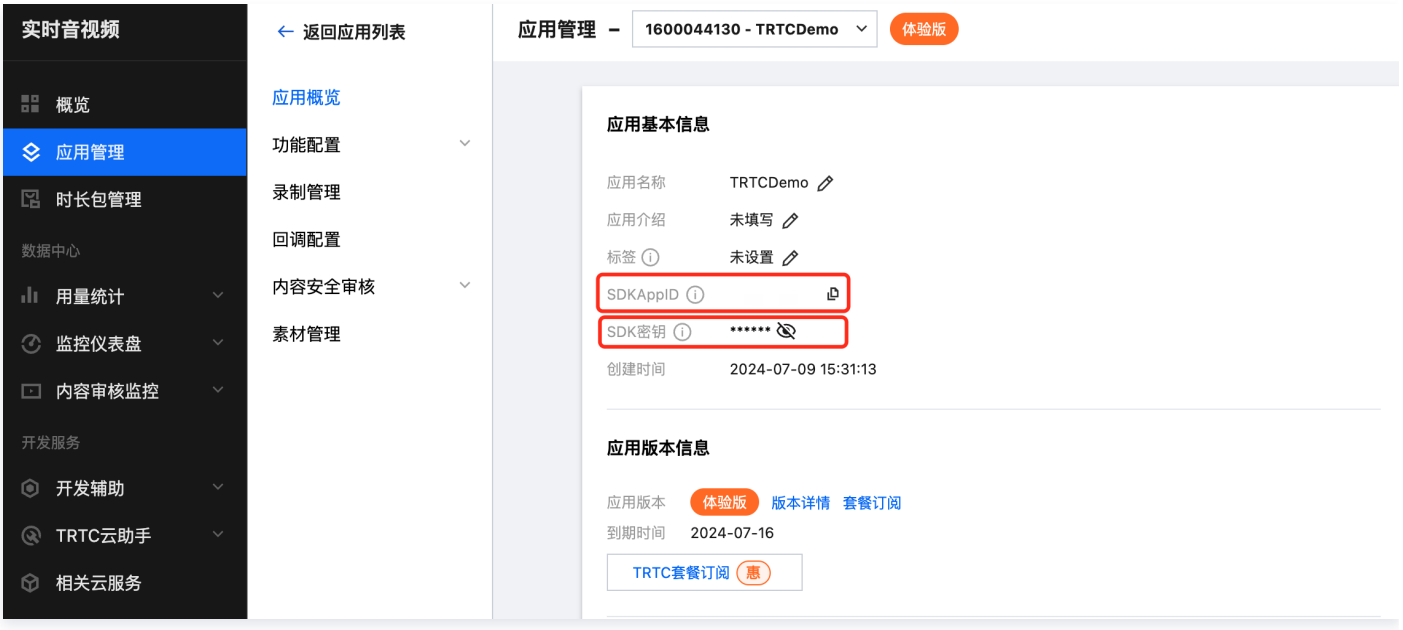
您可以在 github 下载 macOS 平台的示例代码，或者在终端运行以下命令：

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

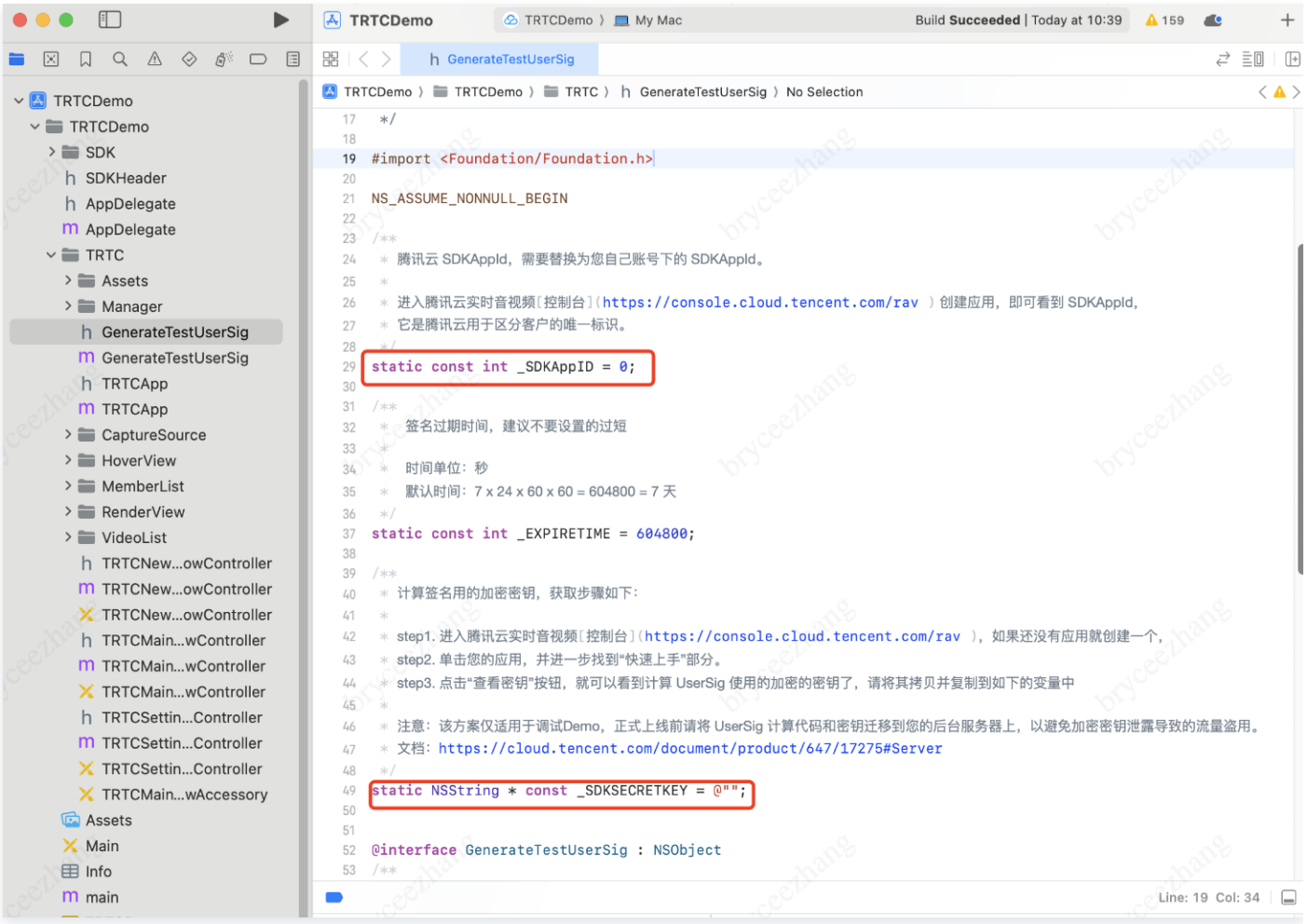
在终端窗口中进入目标工程目录后执行 `pod init` 命令即可，导入macOS的SDK 中其他步骤已经在示例代码中操作过。

步骤2. 配置 Demo

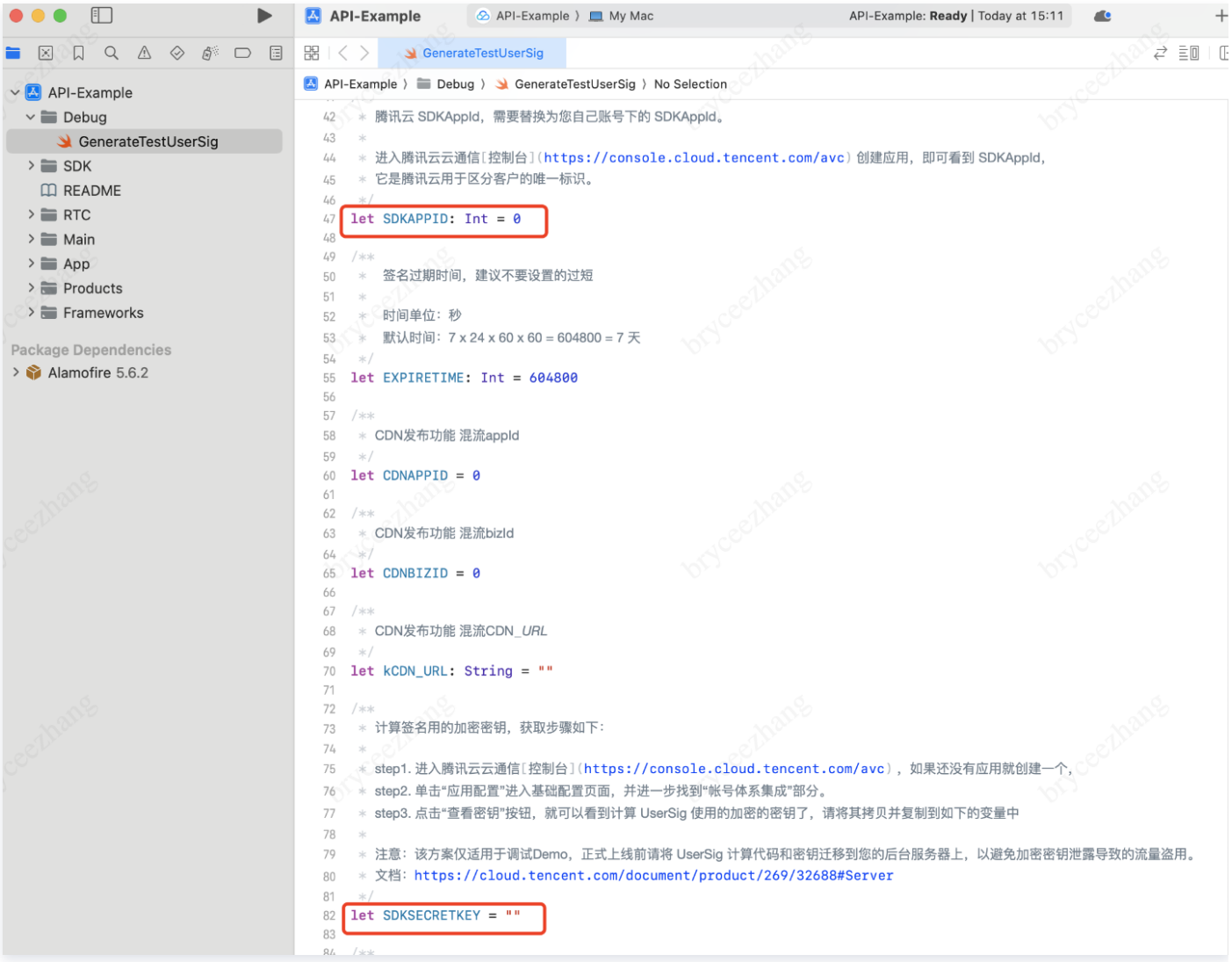
1. 登录 实时音视频控制台，单击**创建应用**。如果您已经完成创建，可以跳过该操作。
2. 在创建应用成功后，您可以在**应用管理**中获取到您的 **SDKAppID** 和 **SDK 密钥**。



3. 如果您选择的是 **OCDemo**，将 `TRTCDemo/TRTC` 目录下 `GenerateTestUserSig.h` 中的 **SDKAppID** 和 **SDKSecretKey** 修改为在**步骤2**中获得的值。



如果您选择的是 SwiftDemo 将 `API-Example/Debug` 目录下 `GenerateTestUserSig.swift` 中的 SDKAppID 和 SDKSecretKey 修改为在步骤2中获得的值。



⚠ 注意：

- **本文使用的生成 UserSig 的方案是在本地配置 SDKSecretKey，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，因此该方法仅适合本地跑通 Demo 和功能调试。正确的 UserSig 签发方式是在您的服务端集成 服务端生成 UserSig，当用户进房时：**
  - **发起 http 请求到您的服务端。**
  - **服务端生成一个 UserSig。**
  - **将其返回给用户供进房使用。**
- **若要将 Demo 部署到公网体验，需要通过 HTTPS 协议，即 `https://域名/xxx` 访问，原因可参见文档 页面访问协议限制说明 。**

步骤3. 跑通 Demo

使用 XCode（11.0及以上的版本）打开源码目录下的 `TRTCDemo.xcworkspace/API-Example.xcworkspace` 工程，编译并运行 TRTC-API-Example 工程即可。

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 。
- 如果您是开发者，也欢迎您加入我们的 TUICallKit 技术交流平台 zhiliao ，进行技术交流和产品沟通。

Android

最近更新时间：2024-07-09 16:21:31

本文介绍了如何快速运行腾讯云 TRTC Android SDK Demo。

环境准备

- 最低兼容 Android 4.1（SDK API Level 16），建议使用 Android 5.0（SDK API Level 21）及以上版本。
- Android Studio 3.5 及以上版本。
- App 要求 Android 4.1 及以上设备。

操作步骤

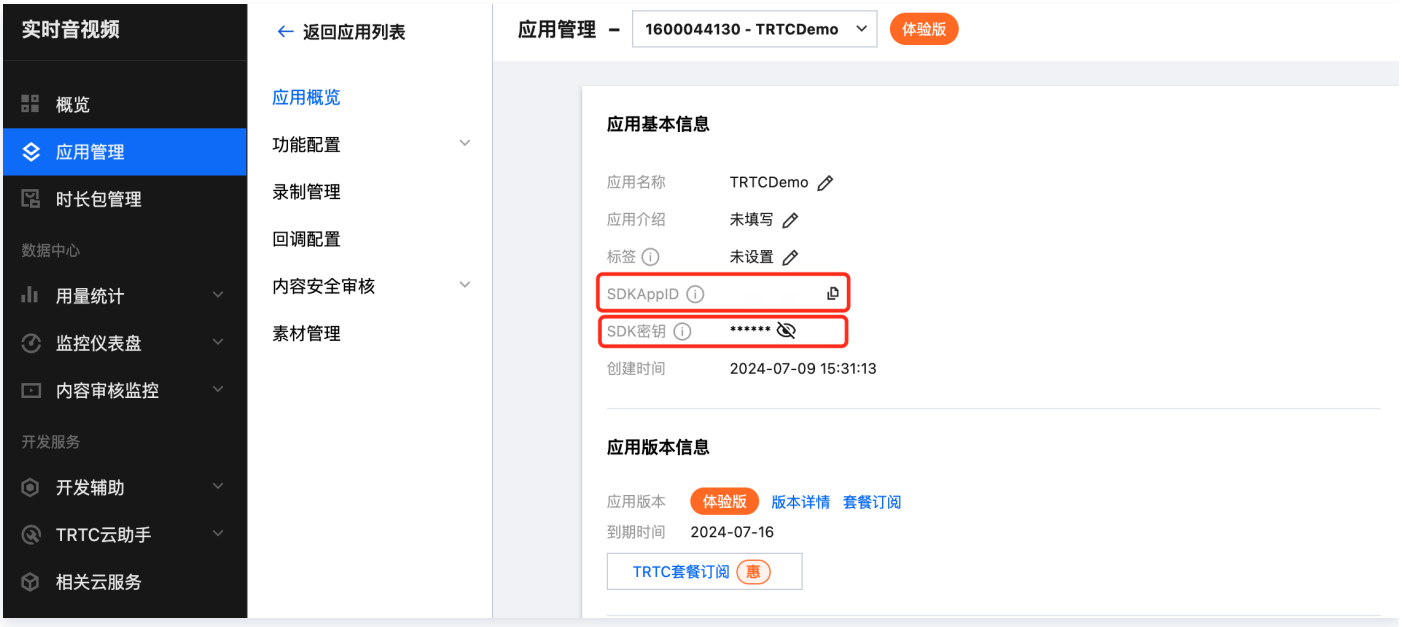
步骤1. 下载 Demo

您可以在 github下载 Andorid 平台的示例代码，或者在终端运行以下命令：

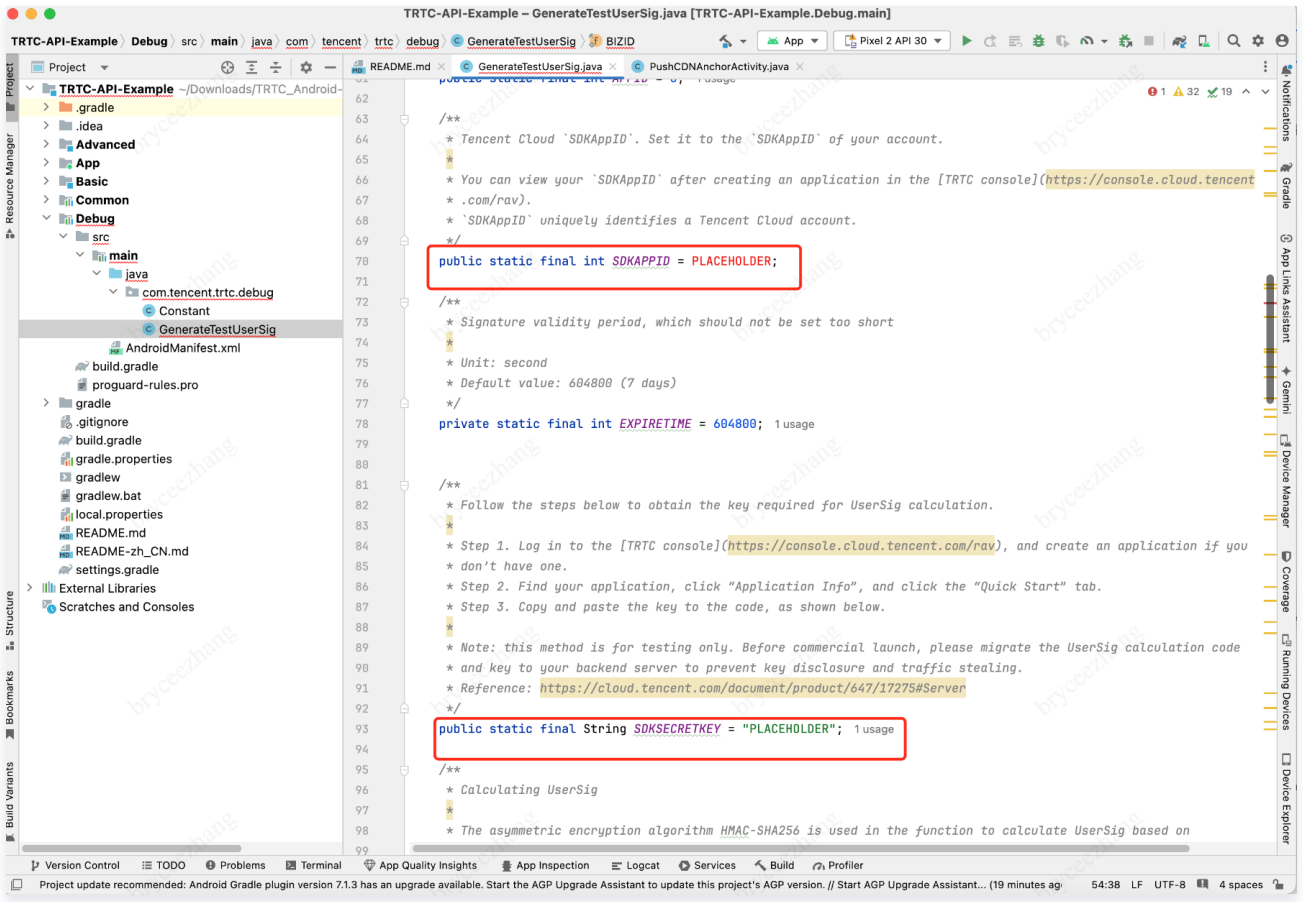
```
git clone https://github.com/Tencent-RTC/TRTC_Android.git
```

步骤2. 配置 Demo

1. 登录 实时音视频控制台，单击**创建应用**。如果您已经完成创建，可以跳过该操作。
2. 在创建应用成功后，您可以在**应用管理**中获取到您的 **SDKAppID** 和 **SDK密钥**。



3. 您需要将 `TRTC-API-Example/Debug/src/main/java/com.tencent.trtc.debug` 目录下 `GenerateTestUserSig` 中的 **SDKAPPID** 和**SDKSECRETKEY** 替换为您在**步骤2**中获取到的值。



> ⚠ **注意：**
>
> - 本文使用的生成 UserSig 的方案是在本地配置 SDKSecretKey，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，因此**该方法仅适合本地跑通 Demo 和功能调试**。正确的 UserSig 签发方式是在您的服务端集成 服务端生成 UserSig，当用户进房时：
>   - 发起 http 请求到您的服务端
>   - 服务端生成一个 UserSig
>   - 将其返回给用户供进房使用
> - 若要将 Demo 部署到公网体验，需要通过 HTTPS 协议，即 `https://域名/xxx` 访问，原因可参见文档 页面访问协议限制说明。

步骤3. 跑通 Demo

配置完成后，使用 Android Studio（3.5及以上的版本）打开源码工程 `TRTC-API-Example`，单击**运行**即可体验。

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。
- 如果您是开发者，也欢迎您加入我们的 RTC Engine 技术交流平台 zhiliao，进行技术交流和产品沟通。

Windows C#

最近更新时间：2024-12-13 21:04:52

本文主要介绍如何快速运行腾讯云 TRTC Demo（Windows C#）。

Windows（C#）开发环境

- Microsoft Visual Studio 2015及以上版本，推荐使用 Microsoft Visual Studio 2019。
- .Net Framework 4.0及以上版本，推荐使用 .Net Framework 4.0。

前提条件

您已 注册腾讯云 账号，并完成 实名认证 。

操作步骤

步骤1：创建新的应用

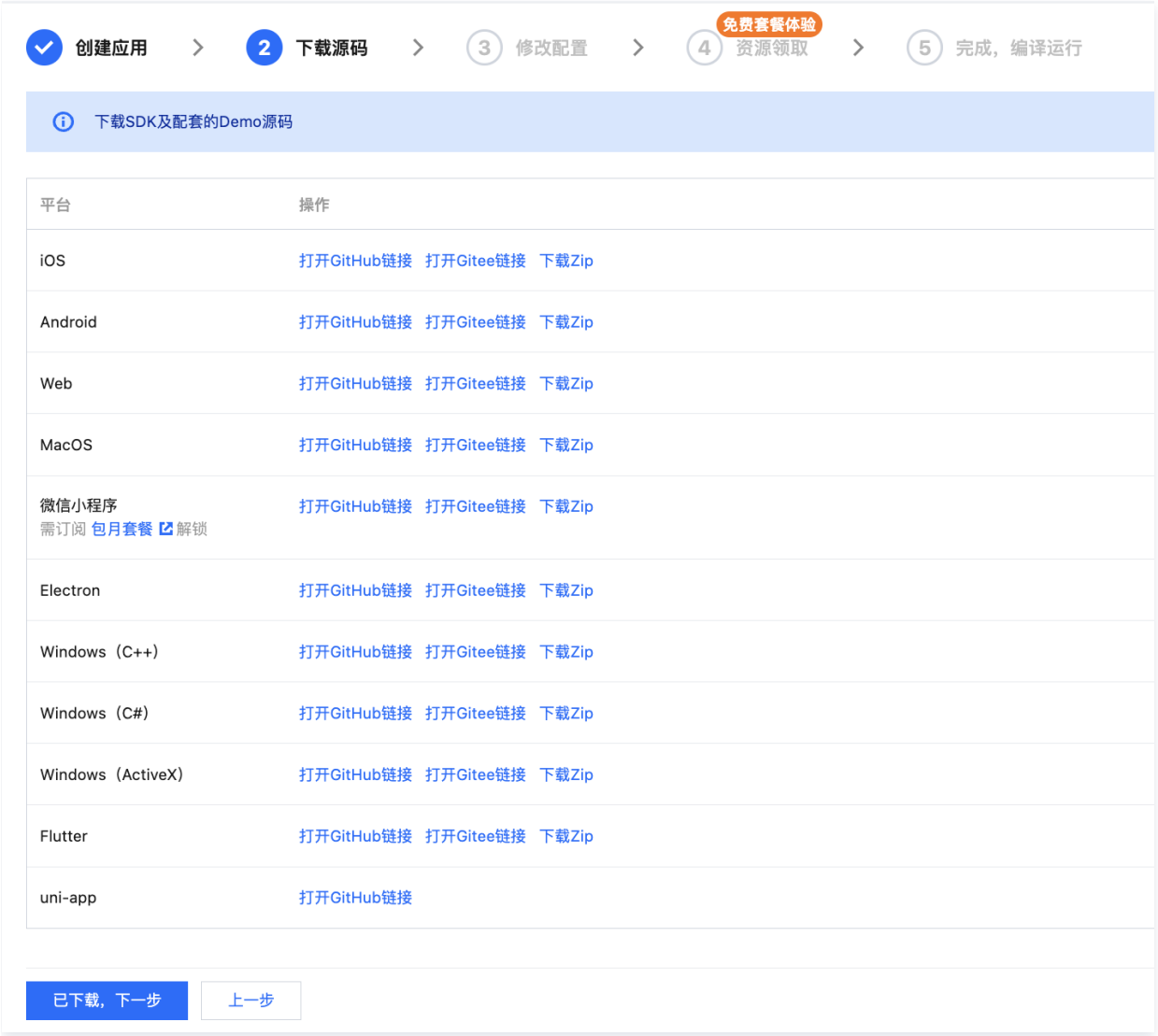
1. 登录实时音视频控制台，选择**开发辅助** > 快速跑通 Demo 。
2. 单击**新建应用**输入应用名称，例如 `TestTRTC` ；若您已创建应用可单击**选择已有应用**。
3. 根据实际业务需求添加或编辑标签，单击**创建**。

说明

- 应用名称只能包含数字、中英文字符和下划线，长度不能超过15个字符。
- 标签用于标识和组织您在腾讯云的各种资源。例如：企业可能有多个业务部门，每个部门有1个或多个 TRTC 应用，这时，企业可以通过给 TRTC 应用添加标签来标记部门信息。标签并非必选项，您可根据实际业务需求添加或编辑。

步骤2：下载 SDK 和 Demo 源码

1. 根据实际业务需求下载 SDK 及配套的 Demo 源码。
2. 下载完成后，单击**已下载，下一步**。



步骤3：配置 Demo 工程文件

1. 进入修改配置页，根据您下载的源码包，选择相应的开发环境。
2. 找到并打开定义 SDKAppID 和密钥信息的文件，如下：

适用平台	文件相对路径
Windows(C#)	TRTC-API-Example-CSharp/GenerateTestUserSig.cs

3. 设置 `GenerateTestUserSig.cs` 文件中的相关参数：
   - SDKAPPID：默认为0，请设置为实际的 SDKAppID。
   - SDKSECRETKEY：默认为空字符串，请设置为实际的密钥信息。

```
/// </summary>

namespace TRTCCSharpDemo
{
    class GenerateTestUserSig
    {
        /// <summary>
        /// 腾讯云 SDKAppId，需要替换为您自己账号下的 SDKAppId。
        ///
        /// 进入腾讯云云通信[控制台](https://console.cloud.tencent.com/avc) 创建应用，即可看到 SDKAppId，
        /// </summary>
        /// <remarks>
        /// 它是腾讯云用于区分客户的唯一标识。
        /// </remarks>
        public const int SDKAPPID = 0;    将 SDKAPPID 粘贴到此处

        /// <summary>
        /// 计算签名用的加密密钥
        ///
        /// 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
        /// </summary>
        /// <remarks>
        /// 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#GetFromServer
        /// </remarks>
        public const string SDKSECRETKEY = @"";    将 密钥 粘贴到此处

```

4. 粘贴完成后，单击**已复制粘贴，下一步**即创建成功。
5. 编译完成后，单击**回到控制台概览**即可。

> **注意**
>
> - 本文提到的生成 UserSig 的方案是在客户端代码中配置 SDKSECRETKEY，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，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 。

步骤4：编译运行

使用 Visual Studio（建议 VS2017）打开源码目录下的 `TRTC-API-Example-CSharp\TRTC-API-Example-CSharp.csproj` 工程文件，推荐选择 Release/X86 构建平台，编译并运行 Demo 工程即可。

常见问题

1. 查看密钥时只能获取公钥和私钥信息，要如何获取密钥?

TRTC SDK 6.6 版本（2019年08月）开始启用新的签名算法 HMAC-SHA256。在此之前已创建的应用，需要先升级签名算法才能获取新的加密密钥。如不升级，您也可以继续使用 老版本算法 ECDSA-SHA256 ，如已升级，您按需切换为新旧算法。

升级/切换操作：

1. 登录 实时音视频控制台 。
2. 在左侧导航栏选择**应用管理**，单击目标应用所在行的**应用信息**。
3. 选择**快速上手**页签，单击**第二步 获取签发 UserSig 的密钥**区域的 **HMAC-SHA256**。

- 切换回老版本算法 ECDSA-SHA256：

- 切换为新版本算法 HMAC-SHA256：

2. 两台设备同时运行 Demo，为什么看不到彼此的画面?

请确保两台设备在运行 Demo 时使用的是不同的 UserID，TRTC 不支持同一个 UserID（除非 SDKAppID 不同）在两个设备同时使用。

3. 防火墙有什么限制?

由于 SDK 使用 UDP 协议进行音视频传输，所以对 UDP 有拦截的办公网络下无法使用，如遇到类似问题，请参见 应对公司防火墙限制 。

Windows C++

最近更新时间：2024-07-09 16:21:31

本文介绍了如何快速运行腾讯云 TRTC Windows SDK Demo。

环境准备

- Microsoft Visual Studio 2017 及以上版本，推荐使　Microsoft Visual Studio 2019。
- 下载并安装 QT 开发环境（QT 5.14.x版本）。
- 下载并安装 .vsix 插件　件，根据官　文件命名找对应插件版本安装即可。
- 打开 VS 并在工具栏中找到 **Extension > QT VS Tools > Qt Options > Qt Versions**, 添加下载好的 Qt 编译器 msvc .
- 将 `SDK/CPlusPlus/Win64/lib` 下的所有的 `.dll`　件拷　到　程　录下的 `debug/release`　件夹下。

> **注意：**
> `debug/release`　件夹均是在 VS 上的环境配置完后　动　成。如果是32位程序，则需要拷贝 `SDK/CPlusPlus/Win64/lib` 下的所有 `.dll` 到 `debug` / `release`　件夹下。

操作步骤

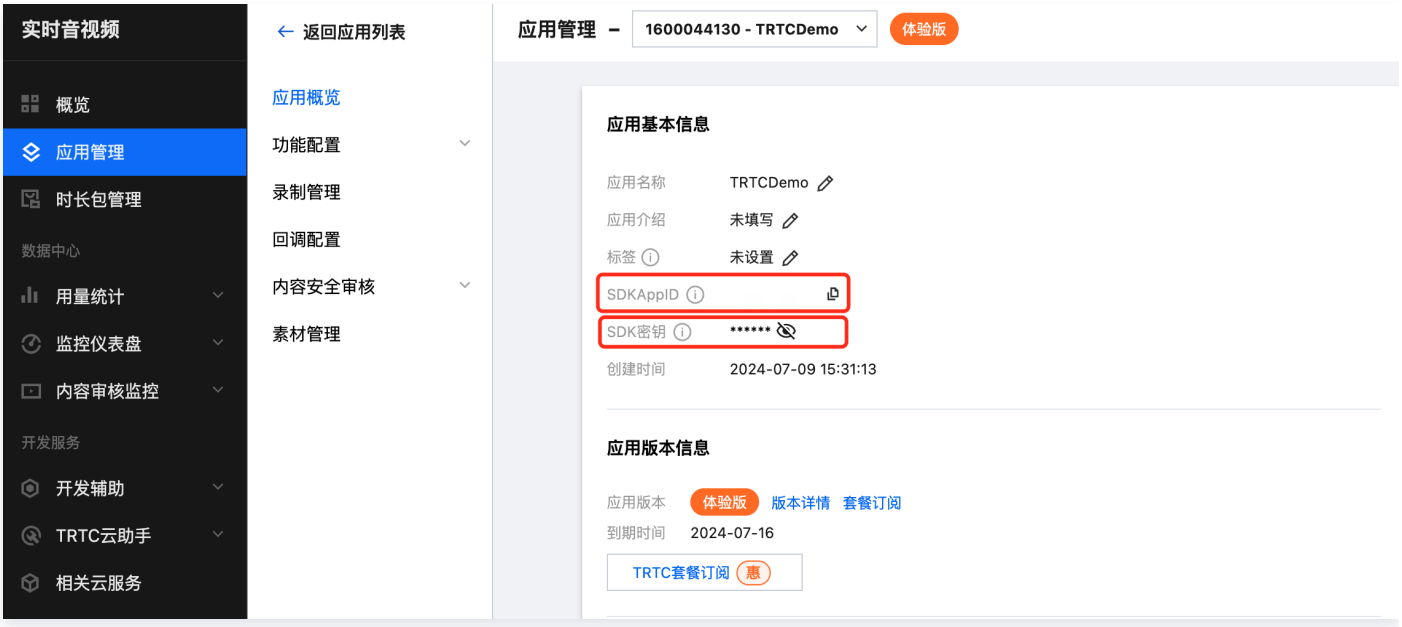
步骤1. 下载 Demo

您可以在 github 下载 TRTC_Windows-C++ 平台的示例代码，或者在终端运行以下命令：

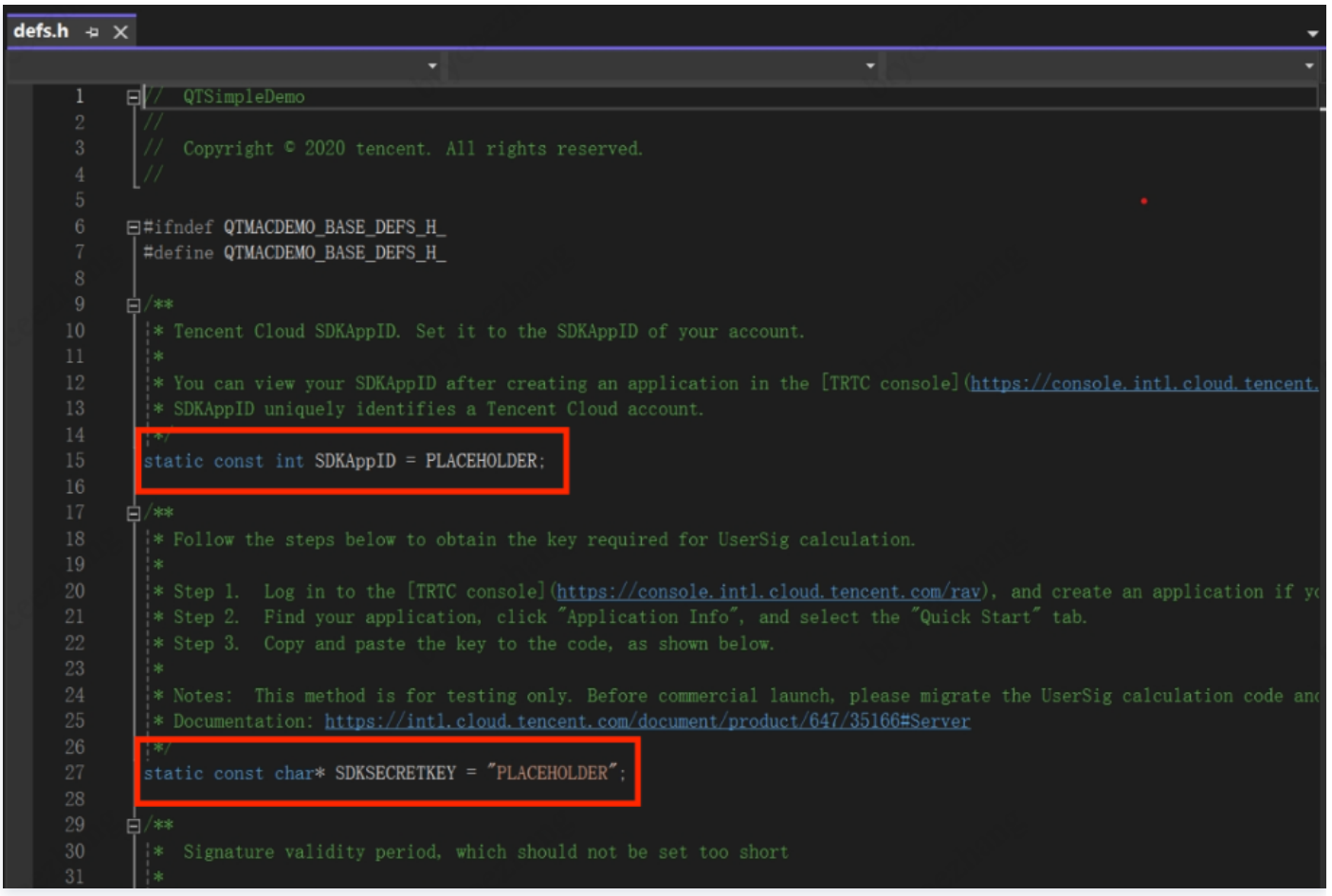
```
git clone https://github.com/Tencent-RTC/TRTC_Windows.git
```

步骤2. 配置 Demo

1. 登录 实时音视频控制台，单击**创建应用**。如果您已经完成创建，可以跳过该操作。
2. 在创建应用成功后，您可以在**应用管理**中获取到您的 **SDKAppID** 和 **SDK密钥**。



3. 将 `TRTC-API-Example-C++/TRTC-API-Example-Qt/src/Util/` 目录下 `defs.h` 中的 SDKAPPID 和 SDKSECRETKEY 替换为步骤2中获取的值。



> ⚠ **注意：**
>
> - 本文使用的生成 UserSig 的方案是在本地配置 SDKSECRETKEY，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，因此**该方法仅适合本地跑通 Demo 和功能调试**。正确的 UserSig 签发方式是在您的服务端集成 服务端生成 UserSig，当用户进房时：
>   - 发起 http 请求到您的服务端
>   - 服务端生成一个 UserSig
>   - 将其返回给用户供进房使用
> - 若要将 Demo 部署到公网体验，需要通过 HTTPS 协议，即 `https://域名/xxx` 访问，原因可参见文档 页面访问协议限制说明。

步骤3. 跑通 Demo

使用 Microsoft Visual Studio（推荐使Microsoft Visual Studio 2019）打开TRTC-API-Example-Qt目录下的源码工程 `QTDemo.sln`，并配置 Qt 环境，（推荐使用 QT 5.14版本）。单击运行即可体验。

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。
- 如果您是开发者，也欢迎您加入我们的 RTC Engine 技术交流平台 zhiliao，进行技术交流和产品沟通。

Windows ActiveX

最近更新时间：2024-03-19 09:31:22

本文主要介绍如何快速运行腾讯云 TRTC Demo（Windows ActiveX版本）。

Windows（ActiveX）运行环境

- Windows 7及以上操作系统。
- IE 9及以上版本，推荐使用 IE 11版本。

前提条件

您已 注册腾讯云 账号，并完成 实名认证 。

操作步骤

步骤1：创建新的应用

1. 登录实时音视频控制台，选择**开发辅助** > 快速跑通 Demo 。
2. 单击**新建应用**输入应用名称，例如 `TestTRTC` ；若您已创建应用可单击**选择已有应用**。
3. 根据实际业务需求添加或编辑标签，单击**创建**。

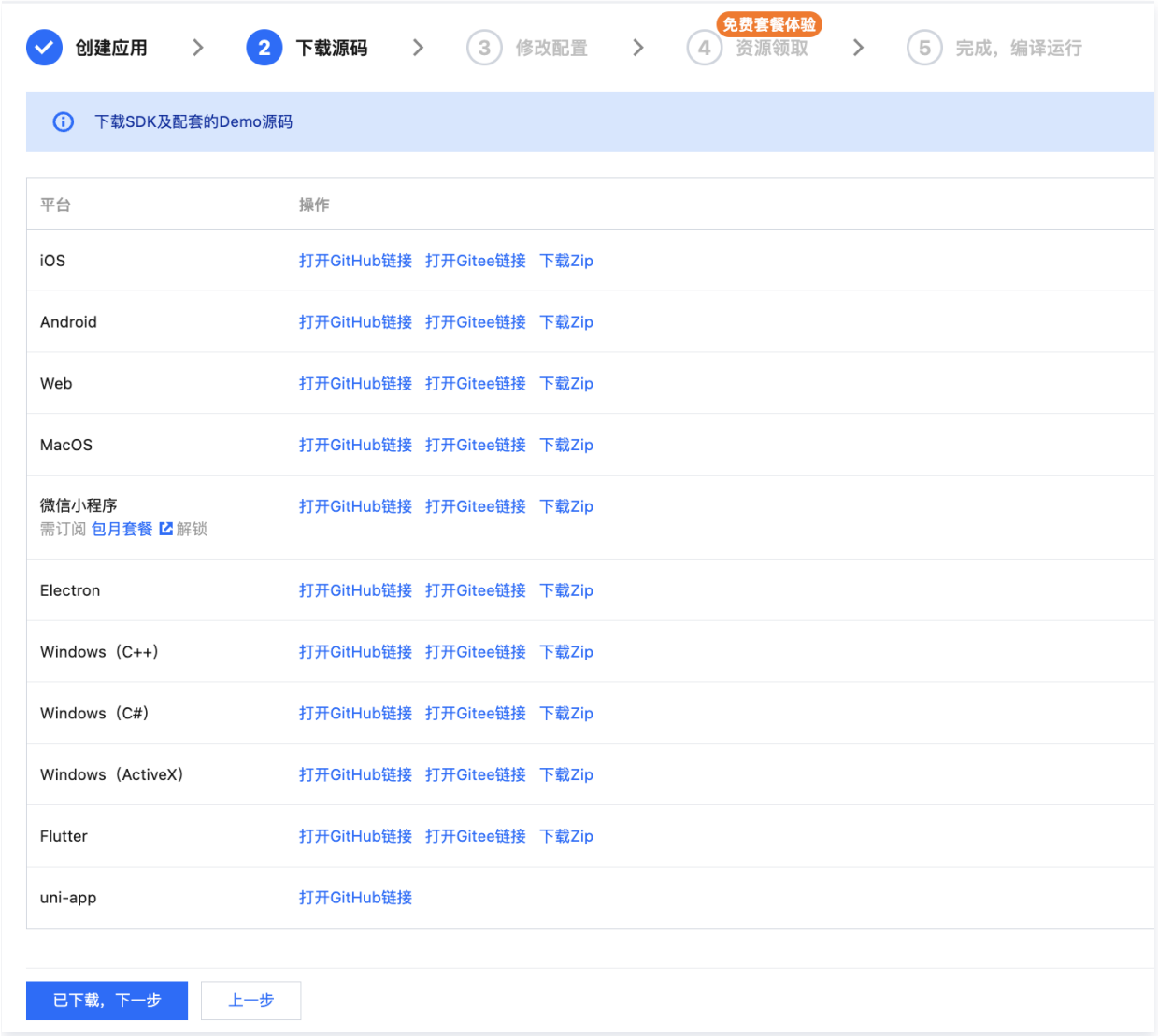
说明

- 应用名称只能包含数字、中英文字符和下划线，长度不能超过15个字符。
- 标签用于标识和组织您在腾讯云的各种资源。例如：企业可能有多个业务部门，每个部门有1个或多个 TRTC 应用，这时，企业可以通过给 TRTC 应用添加标签来标记部门信息。标签并非必选项，您可根据实际业务需求添加或编辑。

步骤2：下载 Windows ActiveX 和 Demo 源码

1. 根据实际业务需求下载 Windows（Active X）的 ZIP 包或到 GitHub/Gitee 下载 ActiveX 的 cab 包及配套的 Demo 源码。
2. 下载完成后，单击**已下载，下一步**。

版权所有：腾讯云计算（北京）有限责任公司



步骤3：部署 ActiveX 插件

1. 将下载好的 ActiveX 压缩包解压，并将其中的 SDK 文件夹（里面包含 LiteAVActiveXSDK.cab）和 TRTC-API-Example-ActiveX 文件夹（里面包含 Demo 所需的 HTML 和 JS 文件）放到 Web 服务器的指定目录下。
2. 在 `TRTC-API-Example-ActiveX\js` 中找到并打开定义 SDKAppID 和密钥信息的文件，如下：

适用平台	文件相对路径
Windows(ActiveX)	TRTC-API-Example-ActiveX/js/GenerateTestUserSig.js

3. 设置 `GenerateTestUserSig.js` 文件中的相关参数：
   - SDKAPPID：默认为0，请设置为实际的 SDKAppID。
   - SDKSECRETKEY：默认为空字符串，请设置为实际的密钥信息。

```
function genTestUserSig(userID) {
  /**
   * 腾讯云 SDKAppId，需要替换为您自己账号下的 SDKAppId。
   *
   * 进入腾讯云实时音视频[控制台](https://console.cloud.tencent.com/rav ) 创建应用，即可看到 SDKAppId，
   * 它是腾讯云用于区分客户的唯一标识。
   */
  const SDKAPPID = 0;
  /**
   * 签名过期时间，建议不要设置的过短
   * <p>
   * 时间单位：秒
   * 默认时间：7 x 24 x 60 x 60 = 604800 = 7 天
   */
  const EXPIRETIME = 604800;

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

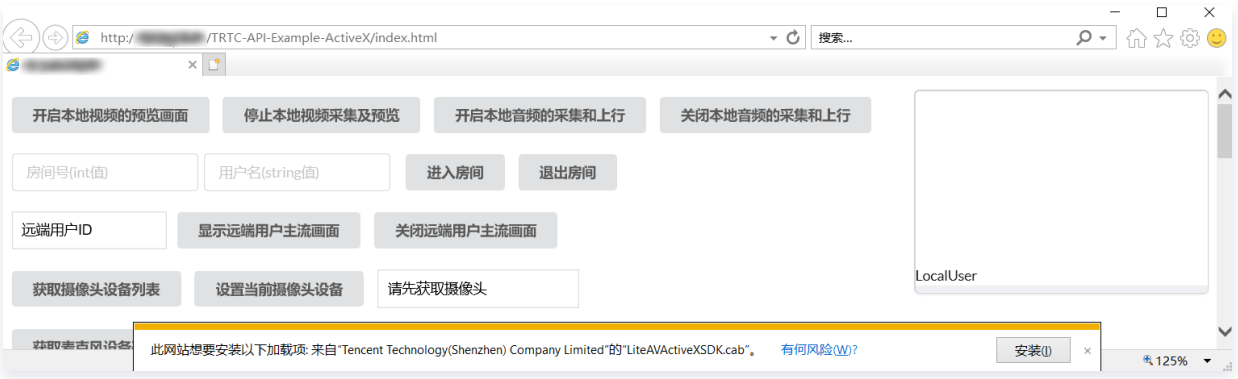
将 SDKAPPID 粘贴到此处

将 密钥 粘贴到此处

4. 将页面中已生成的SDKAppID和密钥粘贴完成后，单击**已复制粘贴，下一步**即创建成功。
5. 完成后，单击**回到控制台概览**即可。

步骤4：运行 Demo

1. 完成以上的下载和部署后，打开 IE 浏览器 并访问 Web 服务器的地址（如 `http://xx.xx.xx.xx/TRTC-API-Example-ActiveX/index.html` ），IE 浏览器弹出 **此网站想要安装以下加载项：来自"Tencent Technology(Shenzhen) CompanyLimited"的"LiteAVActiveXSDK.cab"。** 如下所示：



2. 单击**安装**会弹出如下的下载安装页面：

3. 安装完成后，可输出房间号和用户名进行音视频会话。

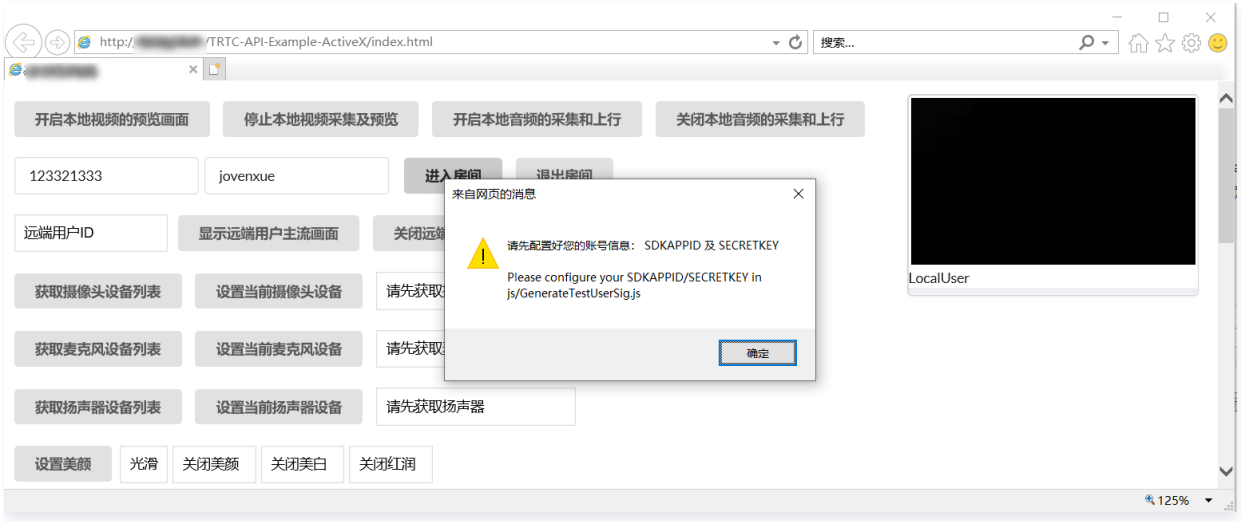
常见问题

1. 单击进入房间后，页面没有任何反应，也没有日志输出?

如果单击 **进入房间** 后，页面没有任何反应，也没有任何日志输出。请检查是否关闭了下载 LiteAVActiveXSDK.cab 的提示框，导致 LiteAVActiveXSDK.cab 没有正确下载并安装到本地（如 步骤4 中截图所示）。此时可以尝试刷新或重新加载网站首页，并在弹出 `此网站想要安装以下加载项：来自"Tencent Technology(Shenzhen) CompanyLimited"的"LiteAVActiveXSDK.cab"。` 时单击**安装**确保 LiteAVActiveXSDK.cab 正常下载安装到本地。

2. 单击进入房间按钮后，弹出配置信息的提示框?

单击进入房间后，弹出如下窗口：



请参照 步骤3 进行 SDKAPPID 和 SDKSECRETKEY 的配置填写。

Web&H5

最近更新时间：2024-10-14 15:43:21

本文主要介绍如何快速运行腾讯云 TRTC Web SDK Demo。

> **说明：**
> - 本教程基于 5.x TRTC Web SDK 实现，若您使用 4.x 版本 SDK，可参见 此教程 。
> - 5.x 与 4.x 版本的区别 。

准备工作

运行 TRTC Web SDK Demo 之前需要了解的事项。

支持的平台

TRTC Web SDK 基于 WebRTC 实现，目前支持桌面端和移动端的主流浏览器，详细支持度表格请参见 支持的平台 。
如果您的应用场景不在支持的表格里，可以打开 TRTC Web SDK 能力检测页面 检测当前环境是否支持 WebRTC 所有能力，例如 WebView 等环境。

URL 域名协议限制

由于浏览器安全策略的限制，使用 WebRTC 能力对页面的访问协议有严格的要求，请参照以下表格进行开发和部署应用。

应用场景	协议	接收（播放）	发送（上麦）	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	-

防火墙限制

在使用 TRTC Web SDK 时，用户可能因防火墙限制导致无法正常进行音视频通话，请参见 应对防火墙限制相关 将相应端口及域名添加至防火墙白名单中。

前提条件

您已 注册腾讯云 账号，并完成 实名认证 。

操作步骤

步骤1：创建新的应用

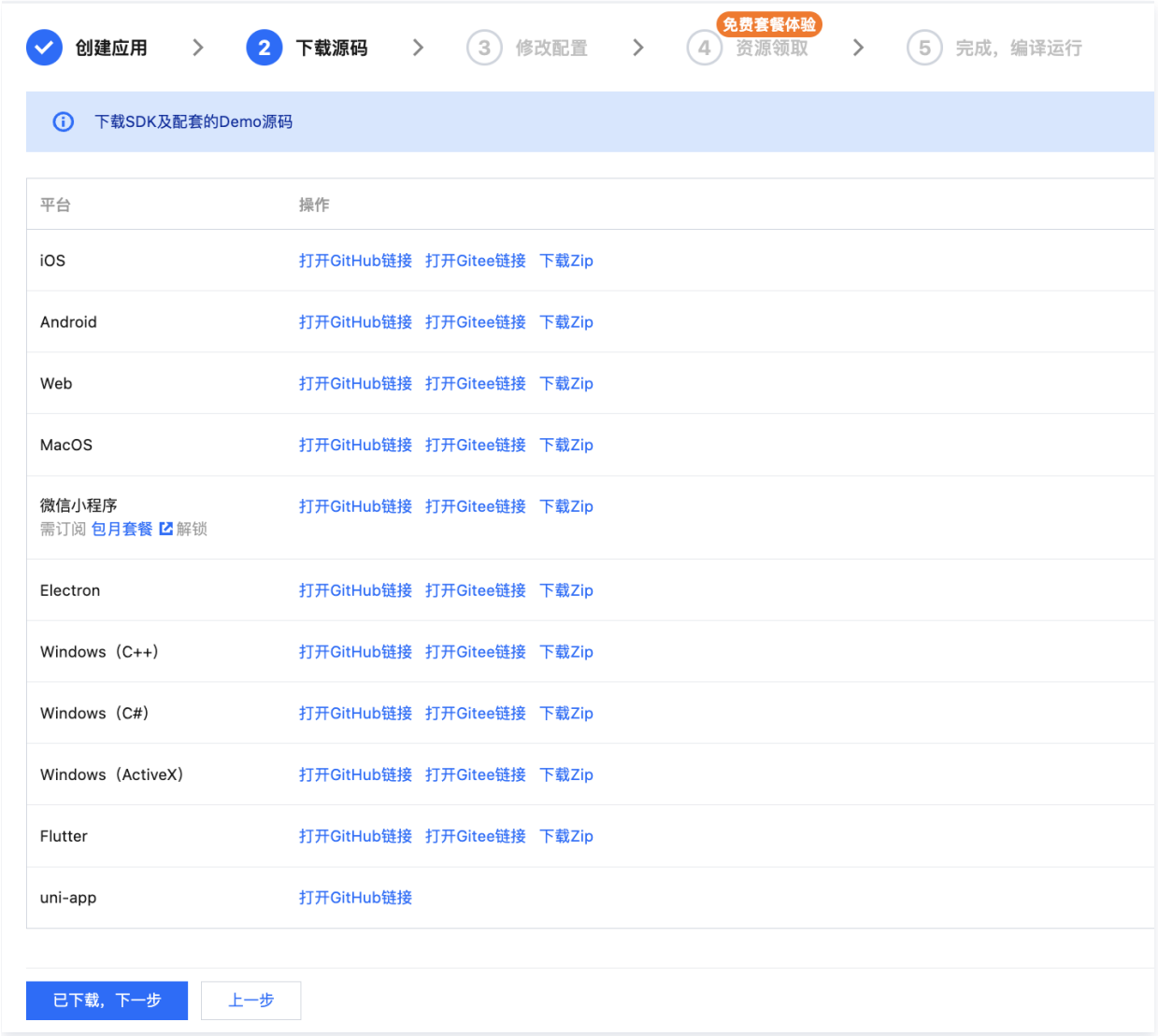
1. 登录实时音视频控制台，选择**开发辅助** > 快速跑通 Demo 。
2. 单击**新建应用**输入应用名称，例如 `TestTRTC` ；若您已创建应用可单击**选择已有应用**。
3. 根据实际业务需求添加或编辑标签，单击**创建**。

说明

- 应用名称只能包含数字、中英文字符和下划线，长度不能超过15个字符。
- 标签用于标识和组织您在腾讯云的各种资源。例如：企业可能有多个业务部门，每个部门有1个或多个 TRTC 应用，这时，企业可以通过给 TRTC 应用添加标签来标记部门信息。标签并非必选项，您可根据实际业务需求添加或编辑。

步骤2：下载 SDK 和 Demo 源码

1. 下载 Web 端 SDK 及配套的 Demo 源码。
2. 下载完成后，单击**已下载，下一步**。



步骤3：获取 SDKAppId 和 密钥（SDKSecretKey）

1. 进入修改配置页，获取 SDKAppID 和 SDKSecretKey。
2. 复制 SDKAppID 和 SDKSecretKey 完成后，单击**已复制粘贴，下一步**即创建成功。

步骤4：运行 Demo

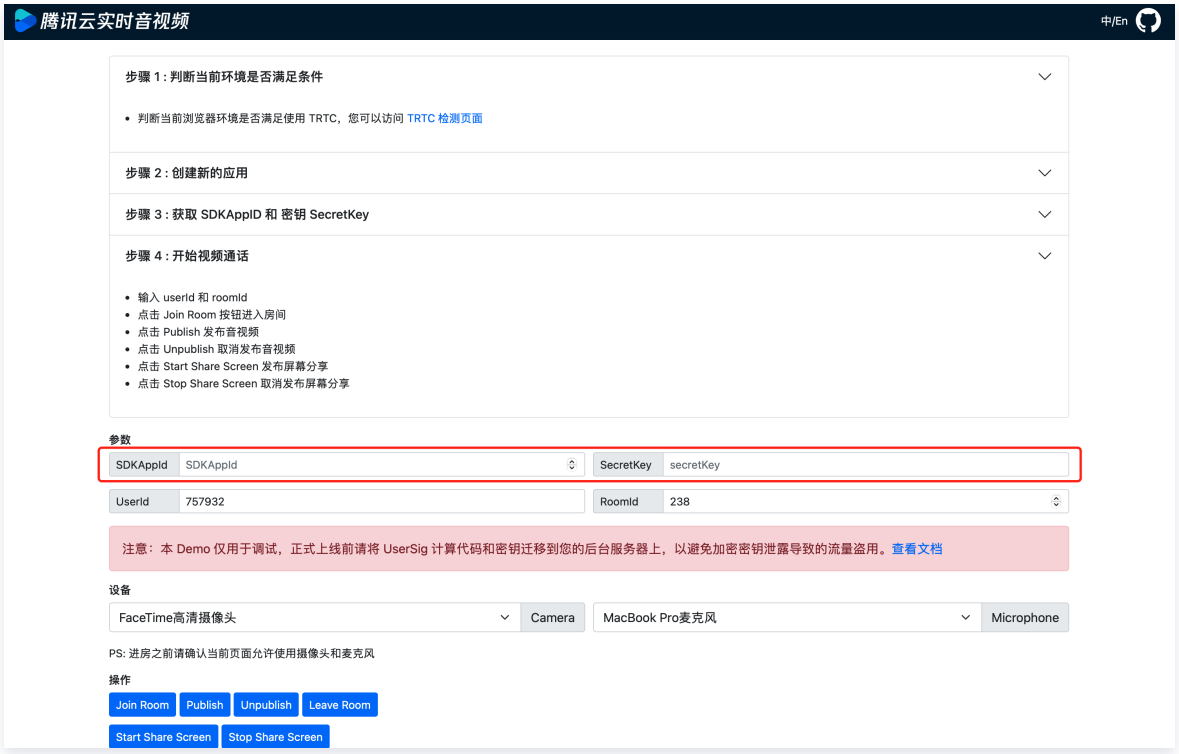
TRTC Web 目前提供以下几种基础 Demo，您可以选择您熟悉的项目框架进行运行体验：

1. `quick-demo-js` 为 TRTC Web 快速运行 Demo (原生 JS 版本)，集成了 TRTC Web SDK 的基础音视频通话、设备选择等功能，使用原生 Js 开发，可直接在浏览器中运行。快速体验可访问 quick-demo-js 在线体验地址，源码地址为 Github 。
2. `quick-demo-vue2-js` 为 TRTC Web 快速运行 Demo (Vue2 版本)，集成了 TRTC Web SDK 的基础音视频通话、设备选择等功能，使用 Vue2 + JavaScript 开发，需要您安装 Node 环境，按下方说明运行体验。快速体验可访问 quick-demo-vue2-js 在线体验地址，源码地址为 Github 。

3. `quick-demo-vue3-ts` 为 TRTC Web 快速运行 Demo (Vue3 版本)，集成了 TRTC Web SDK 的基础音视频通话、设备选择等功能，使用 Vue3 + TypeScript 开发，需要您安装 Node 环境，按下方说明运行体验。快速体验可访问 quick-demo-vue3-ts 在线体验地址，源码地址为 Github 。

quick-demo-js

1. 在下载的源码中找到并使用浏览器打开 `TRTC_Web/quick-demo-js/index.html` 文件。
2. 在浏览器打开的页面中填写 步骤3 获取的 SDKAppId 和 SDKSecretKey。



3. 功能体验
   - 点击 **Join Room** 进入房间。
   - 点击 **Publish** 发布音视频。
   - 点击 **Unpublish** 取消发布音视频。
   - 点击 **Start Share Screen** 发布屏幕分享。
   - 点击 **Stop Share Screen** 取消发布屏幕分享。
4. 加入房间后您可以通过分享邀请链接与被邀请人一起体验 TRTC Web 语音及视频互通功能。

quick-demo-vue2-js

1. 在下载的源码中找到并进入到 `TRTC_Web/quick-demo-vue2-js/` 目录下。

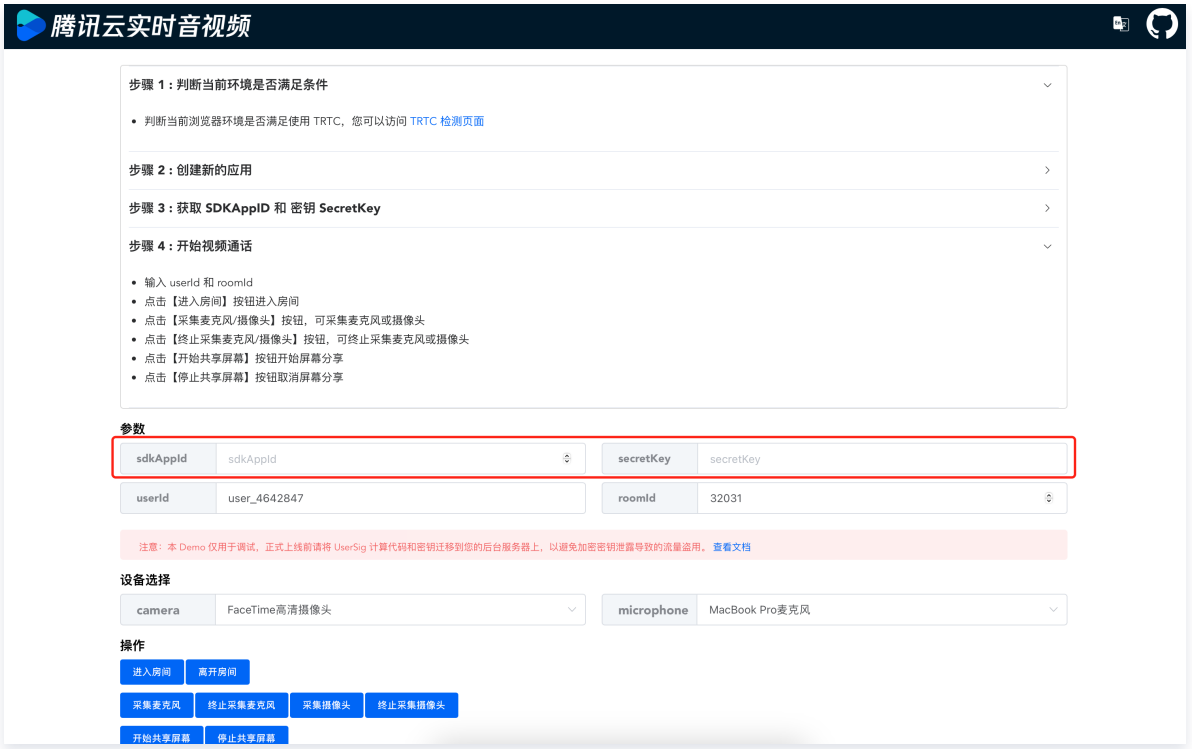
```
cd TRTC_Web/quick-demo-vue2-js/
```

2. 本地运行 Demo，在终端执行以下命令，将会自动安装依赖并运行 demo

```
npm start
```

默认浏览器会自动打开 http://localhost:8080/ 地址。

3. 在浏览器打开的页面中填写 步骤3 获取的 SDKAppId 和 SDKSecretKey。



4. 功能体验
   - 点击【 进入房间 】按钮进入房间。
   - 点击【 采集麦克风/摄像头 】按钮，可采集麦克风或摄像头。
   - 点击【 终止采集麦克风/摄像头 】按钮，可终止采集麦克风或摄像头。
   - 点击【 开始共享屏幕 】按钮开始屏幕分享。
   - 点击【 停止共享屏幕 】按钮取消屏幕分享。
5. 加入房间后您可以通过分享邀请链接与被邀请人一起体验 TRTC Web 语音及视频互通功能。

quick-demo-vue3-ts

1. 在下载的源码中找到并进入到 `TRTC_Web/quick-demo-vue3-ts/` 目录下。

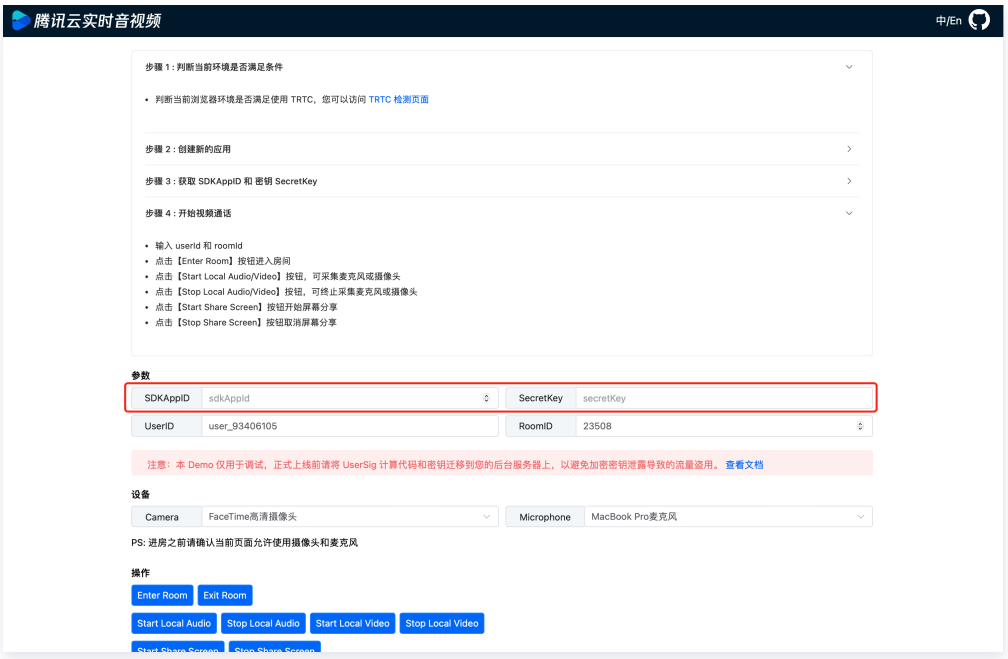
```
cd TRTC_Web/quick-demo-vue3-ts
```

2. 本地运行 Demo。

```
npm start
```

默认浏览器会自动打开 `http://localhost:3000/` 地址。

3. 在浏览器打开的页面中填写 步骤3 获取的 SDKAppId 和 SDKSecretKey。



4. 功能体验
   - 输入 userId 和 roomId。
   - 点击 **Enter Room** 按钮进入房间。
   - 点击**Start Local Audio/Video** 按钮，可采集麦克风或摄像头。
   - 点击 **Stop Local Audio/Video** 按钮，可终止采集麦克风或摄像头。
   - 点击 **Start Share Screen** 按钮开始屏幕分享。
   - 点击 **Stop Share Screen** 按钮取消屏幕分享。
5. 加入房间后您可以通过分享邀请链接与被邀请人一起体验 TRTC Web 语音及视频互通功。

⚠ 注意：

- 本文使用的生成 UserSig 的方案是在本地配置 SDKSECRETKEY，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资源，因此**该方法仅适合本地跑通 Demo 和功能调试**。正确的 UserSig 签发方式是在您的服务端集成 服务端生成 UserSig，当用户进房时：
  - 发起 http 请求到您的服务端
  - 服务端生成一个 UserSig
  - 将其返回给用户供进房使用
- 本地体验 Demo 可以直接在本地搭建静态服务（本地计算机需要接入互联网），通过 `http://localhost:端口` 访问，打开两个页面即可进行通话。部署到公网体验，需要通过 HTTPS 协议，即 `https://域名/xxx` 访问，原因可参见文档 页面访问协议限制说明 。

常见问题

参见 Web 端常见问题 。

Electron

最近更新时间：2024-12-13 16:58:33

本文主要介绍如何快速运行腾讯云 TRTC-API-Example（Electron）。

前提条件

您已 注册腾讯云 账号，并完成 实名认证 。

操作步骤

步骤1：创建新的应用

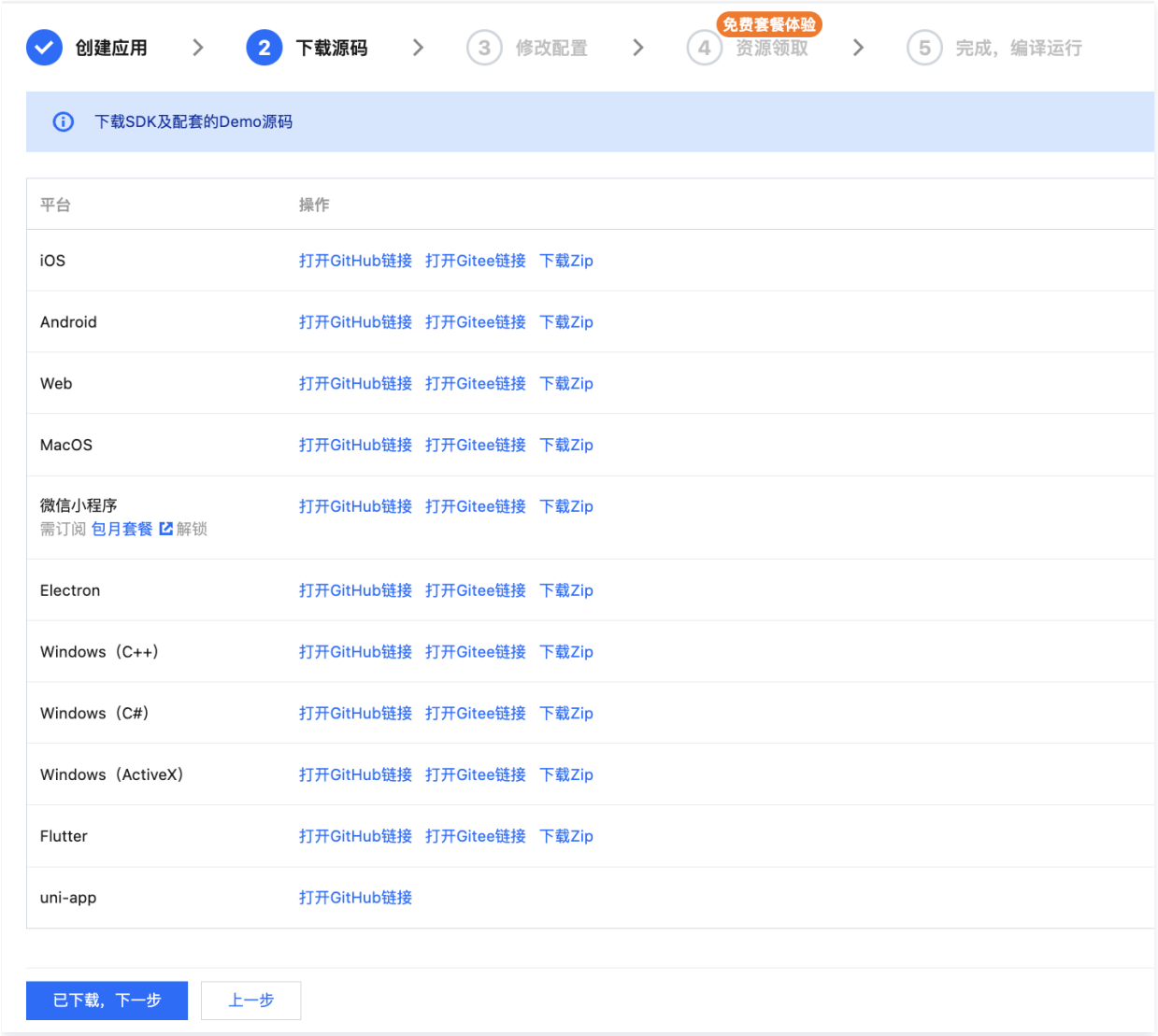
1. 登录实时音视频控制台，选择**开发辅助** > 快速跑通 Demo 。
2. 单击**新建应用**输入应用名称，例如 `TestTRTC` ；若您已创建应用可单击**选择已有应用**。
3. 根据实际业务需求添加或编辑标签，单击**创建**。

说明

- 应用名称只能包含数字、中英文字符和下划线，长度不能超过15个字符。
- 标签用于标识和组织您在腾讯云的各种资源。例如：企业可能有多个业务部门，每个部门有1个或多个 TRTC 应用，这时，企业可以通过给 TRTC 应用添加标签来标记部门信息。标签并非必选项，您可根据实际业务需求添加或编辑。

步骤2：下载 SDK 和 TRTC-API-Example 源码

1. 下载 Electron 端 SDK 及配套的 Demo 源码。
2. 下载完成后，单击**已下载，下一步**。



步骤3：配置 TRTC-API-Example 工程文件

1. 找到并打开 `TRTC-API-Example/assets/debug/gen-test-user-sig.js` 文件。
2. 设置 `gen-test-user-sig.js` 文件中的相关参数：
   - SDKAPPID：默认为 0，请设置为实际的 SDKAppID。
   - SDKSECRETKEY：默认为 ''，请设置为实际的密钥信息。

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

  const SDKAPPID = 0;    将 SDKAPPID 粘贴到此处

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


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

  const SDKSECRETKEY = '';    将 密钥 粘贴到此处

```

> **注意**
>
> - 本文提到的生成 UserSig 的方案是在客户端代码中配置 SDKSECRETKEY，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，因此**该方法仅适合本地跑通 TRTC-API-Example 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 。

步骤4：编译运行

```
cd TRTC-API-Example
npm install
cd src/app/render/main-page
npm install

cd ../../..
npm run start
```

常见问题

1. 查看密钥时只能获取公钥和私钥信息，该如何获取密钥?

TRTC SDK 6.6 版本（2019年08月）开始启用新的签名算法 HMAC-SHA256。在此之前已创建的应用，需要先升级签名算法才能获取新的加密密钥。如不升级，您也可以继续使用 老版本算法 ECDSA-SHA256 ，如已升级，您按需切换为新旧算法。

升级/切换操作：

1. 登录 实时音视频控制台 。
2. 在左侧导航栏选择**应用管理**，单击目标应用所在行的**应用信息**。
3. 选择**快速上手**页签，单击**第二步 获取签发UserSig的密钥**区域的 **HMAC-SHA256**。

- 切换回老版本算法 ECDSA-SHA256：

- 切换为新版本算法 HMAC-SHA256：

2. 防火墙有什么限制?

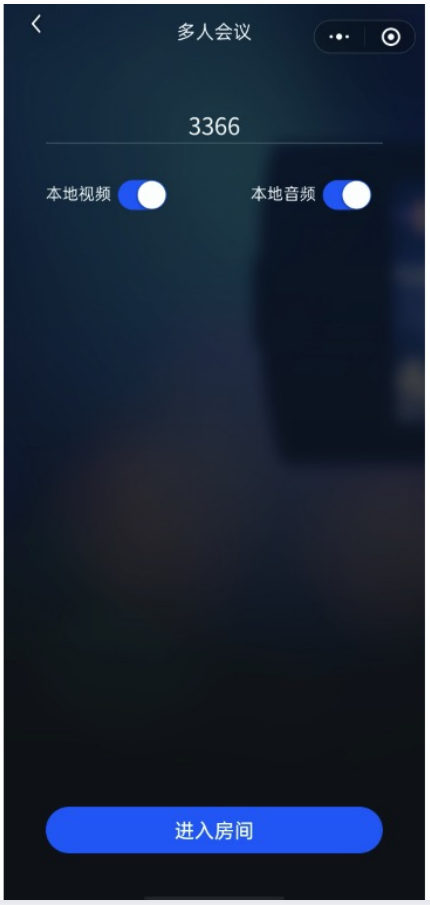
由于 SDK 使用 UDP 协议进行音视频传输，所以在对 UDP 有拦截的办公网络下无法使用。如遇到类似问题，请参见 应对公司防火墙限制 排查并解决。

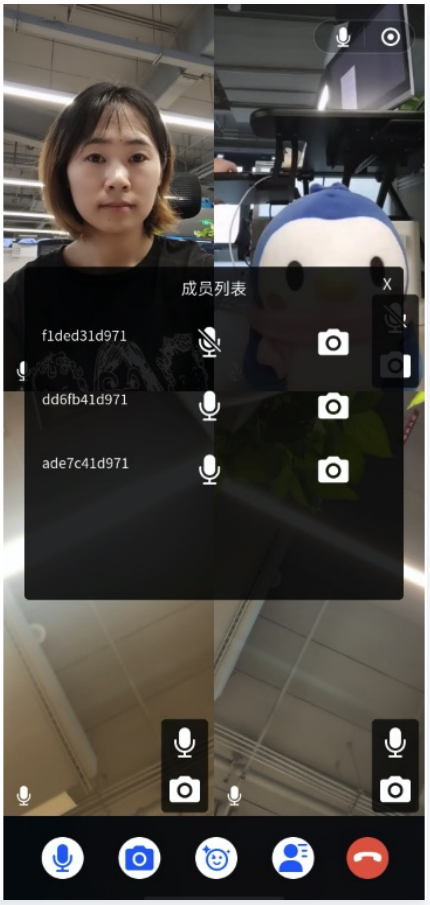
小程序

最近更新时间：2024-12-04 16:01:22

本文主要介绍如何快速跑通微信小程序版本的 TRTC Demo，您可以从 Github 上的 WXMini 目录下获取相关代码。Demo 中前三个功能项演示了三个不同的应用场景：

- **语音聊天室**：纯语音交互，支持多人互动语音聊天，以及混音、混响等声音特效功能。适合在线狼人杀、在线语音直播等社交类场景。
- **双人通话**：1v1视频通话，配合 Web IM SDK 可以实现在线问诊，在线客服等需要面对面交流的沟通场景。
- **多人会议**：支持多路视频通话和大小画面等围绕视频会议相关的高级功能，适用于远程培训、在线教育等场景。





环境要求

- 微信 App iOS 最低版本要求：7.0.9。
- 微信 App Android 最低版本要求：7.0.8。
- 小程序基础库最低版本要求：2.10.0。
- 由于微信开发者工具不支持原生组件（即 <live-pusher> 和 <live-player> 标签），需要在真机上进行运行体验。
- 由于小程序测试号不具备 <live-pusher> 和 <live-player> 的使用权限，需要申请常规小程序账号进行开发。
- 请使用原生小程序开发环境。如果想在 uniapp 开发环境开发音视频通话，可以参考使用 TUICallKit 组件 。

前提条件

> **说明：**
> - 自2023年02月15日起，如需使用微信小程序 SDK 音视频通话和互动直播，需开通订阅 TRTC 包月套餐 以解锁小程序SDK能力，详见包月套餐 功能与计费说明 。

- **2023年02月15日前创建过 TRTC 应用的腾讯云账号下的所有应用 SdkAppId，作为缓冲期可免费使用微信小程序 SDK 能力至2023年04月01日。**

1. 您已 注册腾讯云 账号，并完成 实名认证 。
2. **开通小程序类目与推拉流标签权限（如不开通则无法正常使用）。**
   出于政策和合规的考虑，微信暂未放开所有小程序对实时音视频功能（即 <live-pusher> 和 <live-player> 标签）的支持：

- 小程序推拉流标签不支持个人小程序，只支持企业类小程序。另外您申请小程序标签公司主体无需与您接入 TRTC SDK 的腾讯云账号的主体一致。
- 小程序推拉流标签使用权限暂时只开放给以下有限 类目 。

申请开通

暂只针对国内主体如下类目的小程序开放，需要先通过类目审核，再在小程序管理后台，「开发」-「接口设置」中自助开通该组件权限。

一级类目/主体类型	二级类目	小程序内容场景
社交	直播	涉及娱乐性质，如明星直播、生活趣事直播、宠物直播等。选择该类目后首次提交代码审核，需经当地互联网主管机关审核确认，预计审核时长7天左右
教育	在线视频课程	网课、在线培训、讲座等教育类直播
医疗	互联网医院，公立医疗机构，其他私立医疗机构	问诊、大型健康讲座等直播
金融	银行、信托、公募基金、私募基金、证券/期货、证券、期货投资咨询、保险、征信业务、新三板信息服务平台、股票信息服务平台（港股/美股）、消费金融、融资担保	金融产品视频客服理赔、金融产品推广直播等
汽车	汽车预售服务	汽车预售、推广直播
政府主体账号	/	政府相关工作推广直播、领导讲话直播等
IT科技	多方通信；音视频设备	为多方提供电话会议/视频会议等服务；智能家居场景下控制摄像头
房地产服务	房地产营销	房地产营销直播服务、在线音视频带看等
商业服务	公证	在线业务办理等

- 符合类目要求的小程序，需要在 微信公众平台 > **开发 > 开发管理 > 接口设置**中自助开通该组件权限，如下图所示：

操作步骤

步骤1：创建新的应用

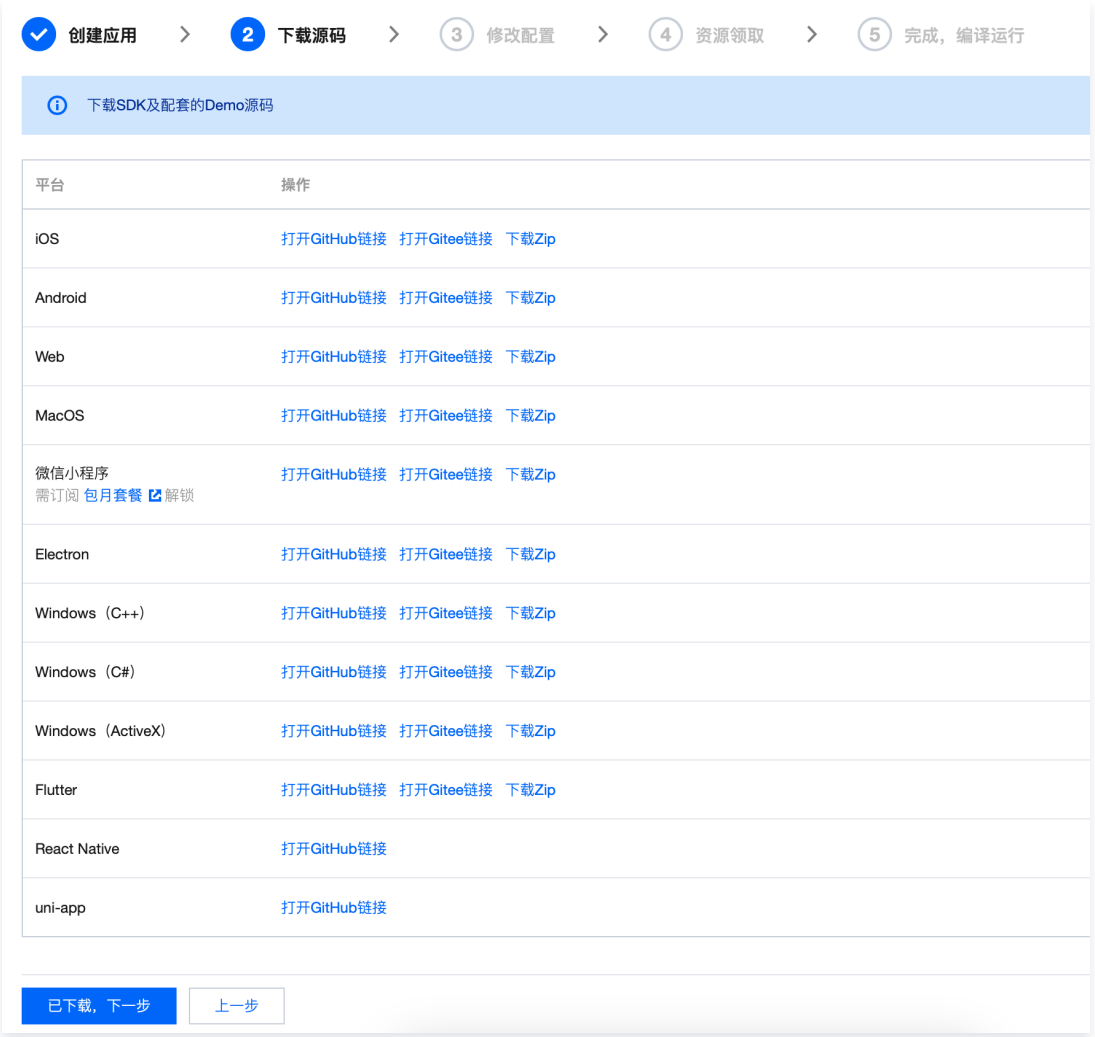
1. 登录实时音视频控制台，选择**开发辅助** > 快速跑通 Demo 。
2. 单击**新建应用**输入应用名称，例如 `TestTRTC` ；若您已创建应用可单击**选择已有应用**。
3. 根据实际业务需求添加或编辑标签，单击**创建**。

说明

- 应用名称只能包含数字、中英文字符和下划线，长度不能超过15个字符。
- 标签用于标识和组织您在腾讯云的各种资源。例如：企业可能有多个业务部门，每个部门有1个或多个 TRTC 应用，这时，企业可以通过给 TRTC 应用添加标签来标记部门信息。标签并非必选项，您可根据实际业务需求添加或编辑。

步骤2：下载 SDK 和 Demo 源码

1. 根据实际业务需求下载 SDK 及配套的 Demo 源码。
2. 下载完成后，单击**已下载，下一步**。



步骤3：配置 Demo 工程文件

1. 进入修改配置页，根据您下载的源码包，选择相应的开发环境。
2. 找到并打开 `./debug/GenerateTestUserSig.js` 文件。
3. 设置 `GenerateTestUserSig.js` 文件中的相关参数：
   - SDKAPPID：默认为0，请设置为实际的 SDKAppID。
   - SECRETKEY：默认为空字符串，请设置为实际的密钥信息。

```
function genTestUserSig(userID) {
  /**
   * 腾讯云 SDKAppId，需要替换为您自己账号下的 SDKAppId。
   *
   * 进入腾讯云实时音视频[控制台](https://console.cloud.tencent.com/rav ) 创建应用，即可看到 SDKAppId，
   * 它是腾讯云用于区分客户的唯一标识。
   */
  var SDKAPPID    = 0;
  /**
   * 签名过期时间，建议不要设置的过短
   * <p>
   * 时间单位：秒
   * 默认时间：7 x 24 x 60 x 60 = 604800 = 7 天
   */
  var EXPIRETIME = 604800;

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

将 SDKAppID 粘贴到此处

将 密钥 粘贴到此处

4. 粘贴完成后，单击**已复制粘贴，下一步**即创建成功。
5. 编译完成后，单击**回到控制台概览**即可。

> **注意：**
> - 本文提到的生成 UserSig 的方案是在客户端代码中配置 SECRETKEY，该方法中 SECRETKEY 很容易被反编译逆向破解，一旦您的密钥泄露，攻击者就可以盗用您的腾讯云流量，因此**该方法仅适合本地跑通 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 。

步骤4：编译运行

1. 打开微信开发者工具，选择**小程序**，单击新建图标，选择**导入项目**。
2. 填写您微信小程序的 AppID，单击**导入**。

> **注意：**
> 此处应输入您微信小程序的 AppID，而非 SDKAppID。

3. 单击**预览**，生成二维码，通过手机微信扫码二维码即可进入小程序。

注意

- 小程序 <live-player> 和 <live-pusher> 标签需要在手机微信上才能使用，微信开发者工具上无法使用。
- 为了小程序能够使用腾讯云房间管理服务，您需要在手机微信上开启调试功能：手机微信扫码二维码后，单击右上角“...” > **打开调试**。

上线/部署到正式环境

- 请申请域名并做备案。
- 请将服务端代码部署到申请的服务器上。
- 请将以下域名配置到小程序控制台 request 合法域名里：

域名	说明
https://cloud.tencent.com	推流域名
https://yun.tim.qq.com	业务域名

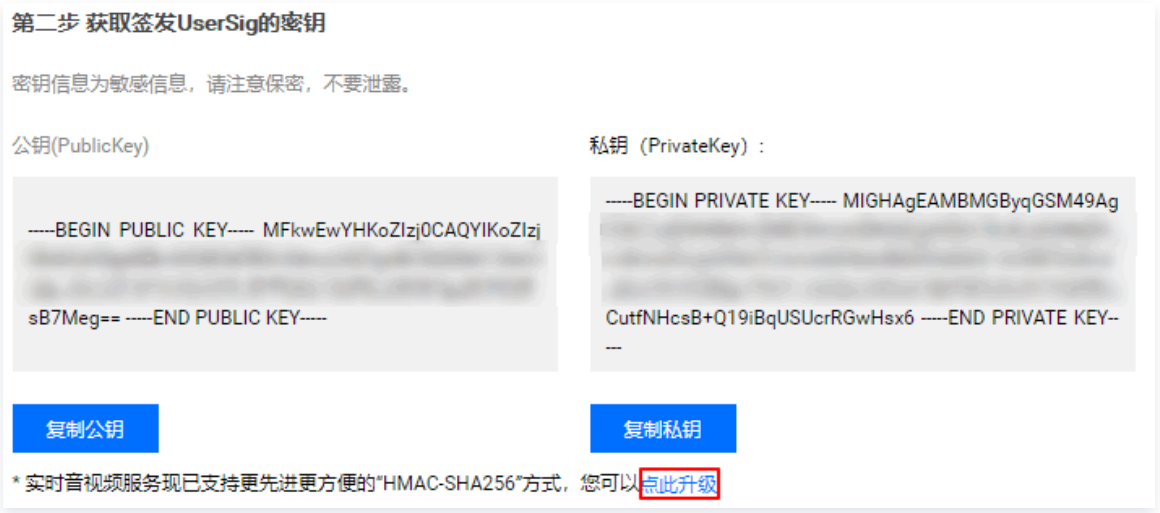
常见问题

1. 查看密钥时只能获取公钥和私钥信息，该如何获取密钥?

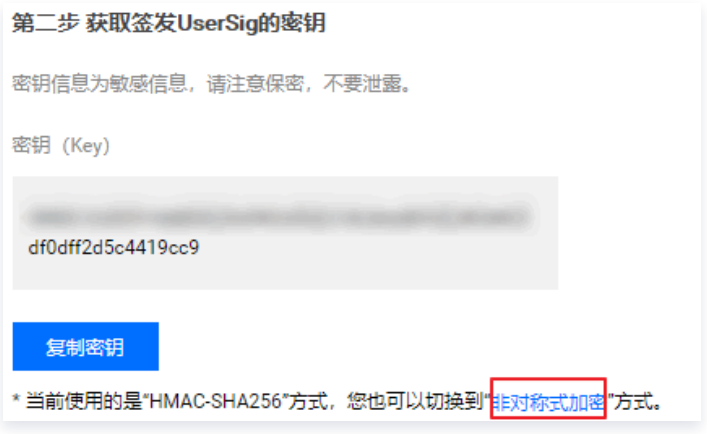
TRTC SDK 6.6 版本（2019年08月）开始启用新的签名算法 HMAC-SHA256。在此之前已创建的应用，需要先升级签名算法才能获取新的加密密钥。如不升级，您也可以继续使用 老版本算法 ECDSA-SHA256 ，如已升级，您按需切换为新旧算法。
升级/切换操作：

1. 登录 实时音视频控制台 。
2. 在左侧导航栏选择**应用管理**，单击目标应用所在行的**应用信息**。
3. 选择**快速上手**页签，单击**第二步 获取签发 UserSig 的密钥**区域的**点此升级**、**非对称式加密**或 HMAC-SHA256。

- 升级：



- 切换回老版本算法 ECDSA-SHA256：



- 切换为新版本算法 HMAC-SHA256：

2. 防火墙有什么限制?

由于 SDK 使用 UDP 协议进行音视频传输，所以对 UDP 有拦截的办公网络下无法使用，如遇到类似问题，请参见 应对公司防火墙限制 。

技术咨询

了解更多详情您可 QQ 咨询：941036374（技术交流群）。

Flutter

最近更新时间：2024-03-19 09:31:22

本文主要介绍如何快速运行腾讯云 TRTC Demo（Flutter）。

环境要求

- Flutter 2.0 及以上版本。
- **Android 端开发：**
  - Android Studio 3.5及以上版本。
  - App 要求 Android 4.1及以上版本设备。
- **iOS 端开发：**
  - Xcode 11.0及以上版本。
  - osx 系统版本要求 10.11 及以上版本
  - 请确保您的项目已设置有效的开发者签名。

前提条件

您已 注册腾讯云 账号，并完成实名认证。

操作步骤

步骤1：创建新的应用

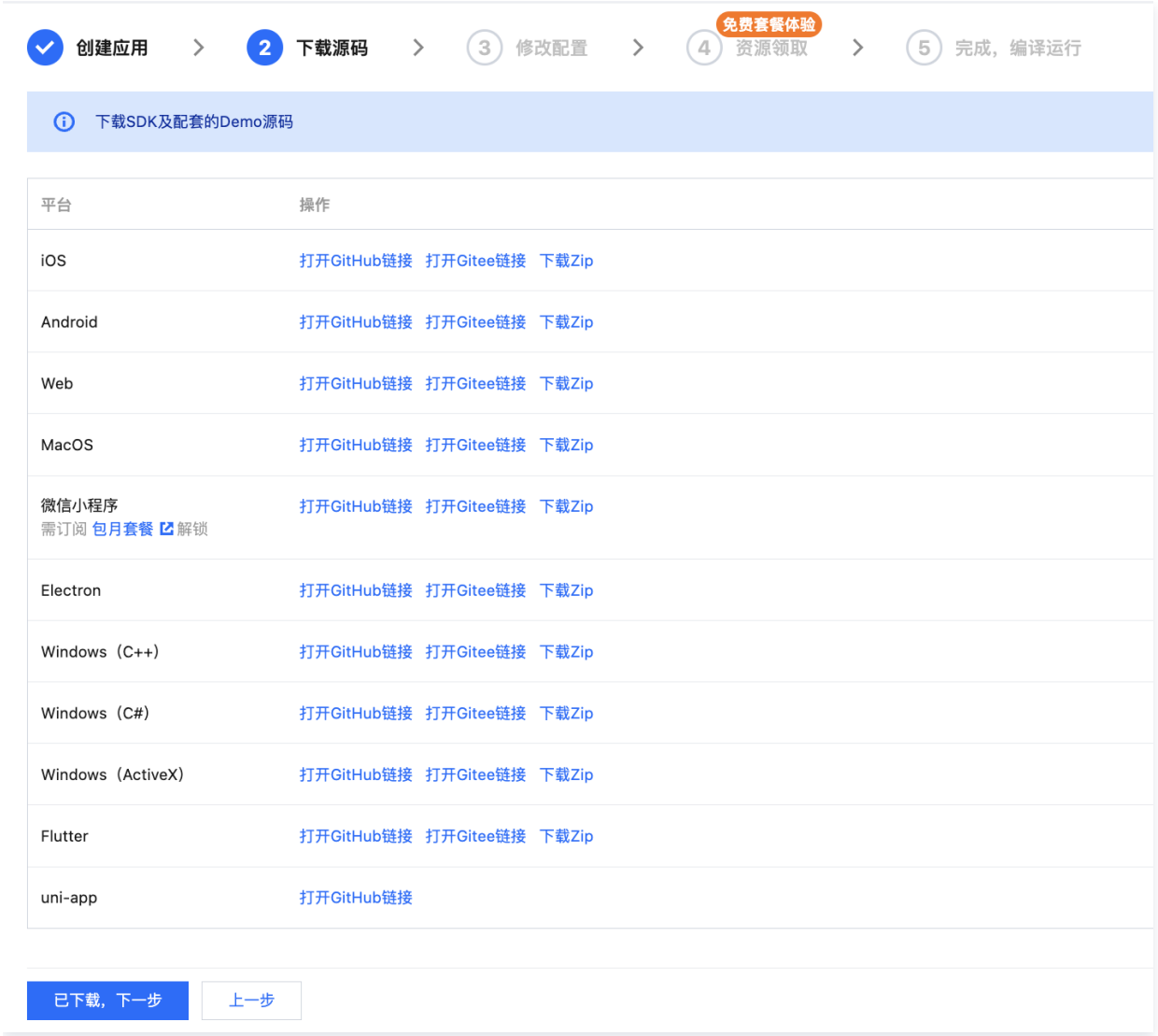
1. 登录实时音视频控制台，选择**开发辅助** > 快速跑通 Demo 。
2. 单击**新建应用**输入应用名称，例如 `TestTRTC`；若您已创建应用可单击**选择已有应用**。
3. 根据实际业务需求添加或编辑标签，单击**创建**。

说明：

- 应用名称只能包含数字、中英文字符和下划线，长度不能超过15个字符。
- 标签用于标识和组织您在腾讯云的各种资源。例如：企业可能有多个业务部门，每个部门有1个或多个 TRTC 应用，这时，企业可以通过给 TRTC 应用添加标签来标记部门信息。标签并非必选项，您可根据实际业务需求添加或编辑。

步骤2：下载 SDK 和 Demo 源码

1. 根据实际业务需求下载 SDK 及配套的 Demo 源码 。
2. 下载完成后，单击**已下载，下一步**。



步骤3：配置 Demo 工程文件

1. 进入修改配置页，根据您下载的源码包，选择相应的开发环境。
2. 找到并打开 `TRTC-Simple-Demo/lib/debug/GenerateTestUserSig.dart` 文件。
3. 设置 `GenerateTestUserSig.dart` 文件中的相关参数：
   - SDKAPPID：默认为0，请设置为实际的 SDKAppID。
   - SDKSECRETKEY：默认为空字符串，请设置为实际的密钥信息。

```
class GenerateTestUserSig {
    /**
     * Tencent Cloud `SDKAppID`. Set it to the `SDKAppID` of your account.
     *
     * You can view your `SDKAppID` after creating an application in the [TRTC console](https://console.cloud.tencent.com/rav).
     * `SDKAppID` uniquely identifies a Tencent Cloud account.
     */
    static int sdkAppId = 0;  将 SDKAPPID 粘贴到此处

    /**
     * Signature validity period, which should not be set too short
     * <p>
     * Unit: second
     * Default value: 604800 (7 days)
     */
    static int expireTime = 604800;

    /**
     * Follow the steps below to obtain the key required for UserSig calculation.
     *
     * Step 1. Log in to the [TRTC console](https://console.cloud.tencent.com/rav), and create an application if you don't have one.
     * Step 2. Find your application, click "Application Info", and click the "Quick Start" tab.
     * Step 3. Copy and paste the key to the code, as shown below.
     *
     * Note: this method is for testing only. Before commercial launch, please migrate the UserSig calculation code and key to your backend server to prevent key disclosure an
     * Reference: https://cloud.tencent.com/document/product/647/17275#Server
     */
    static String sdkSecretKey = '';  将 密钥 粘贴到此处

```

4. 粘贴完成后，单击**已复制粘贴，下一步**即创建成功。
5. 编译完成后，单击**回到控制台概览**即可。

> **说明：**
> - 本文提到的生成 UserSig 的方案是在客户端代码中配置 SDKSECRETKEY，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，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 。

步骤4：编译运行

1. 执行 `flutter pub get` 。
2. 编译运行调试：

Android 端

1. 执行 `flutter run` 。
2. 使用 Android Studio（3.5及以上的版本）打开源码工程，单击**运行**即可。

iOS 端

1. 到 iOS 目录 `pod install` 。
2. 使用 XCode（11.0及以上的版本）打开源码目录下的 `/ios` 工程，编译并运行 Demo 工程即可。

常见问题

如何查看 TRTC 日志?

TRTC 的日志默认压缩加密，后缀为 `.xlog` 。地址如下：

- **iOS 端**：sandbox 的 `Documents/log` 。
- **Android 端**：
  - 6.7及之前的版本：`/sdcard/log/tencent/liteav` 。
  - 6.8之后的版本：`/sdcard/Android/data/包名/files/log/tencent/liteav/` 。

iOS 无法显示视频（Android 没问题）？

请确认在您的 info.plist 中，`io.flutter.embedded_views_preview` 是否为 YES。

Android Manifest merge failed 编译失败?

请打开 `/example/android/app/src/main/AndroidManifest.xml` 文件。

1. 将 `xmlns:tools="http://schemas.android.com/tools"` 加入到 manifest 中。
2. 将 `tools:replace="android:label"` 加入到 application 中。

```
android > app > src > main > AndroidManifest.xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.example.mlp">
    <!-- io.flutter.app.FlutterApplication is an android.app.Application that
         calls FlutterMain.startInitialization(this); in its onCreate method.
         In most cases you can leave this as-is, but you if you want to provide
         additional functionality it is fine to subclass or reimplement
         FlutterApplication and put your custom class here. -->
    <application
        tools:replace="android:label"
        android:name="io.flutter.app.FlutterApplication"
        android:label="mlp"
        android:icon="@mipmap/ic_launcher">
```

说明：
更多常见问题，请参见 Flutter 相关问题 。

uni-app

最近更新时间：2024-03-13 10:07:01

本文主要介绍如何快速运行腾讯云 TRTC Demo（uni-app）。

- uni-app 插件
- API 文档
- GitHub

环境要求

- 建议使用最新的 HBuilderX 编辑器 。
- iOS 9.0 或以上版本且支持音视频的 iOS 设备，暂不支持模拟器。
- Android 版本不低于 4.1 且支持音视频的 Android 设备，暂不支持模拟器。如果为真机，请开启“允许调试”选项。
- iOS/Android 设备已经连接到 Internet。

前提条件

1. 您已 注册腾讯云 账号，并完成实名认证。
2. DCloud 开发者中心注册 之后登录 HBuilderX 编辑器。

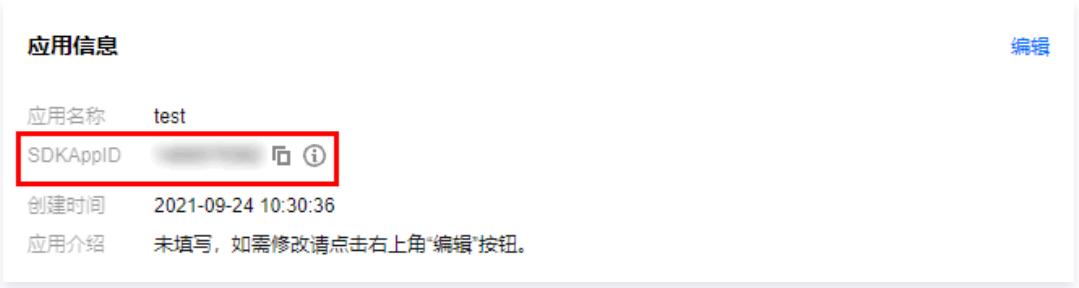
操作步骤

步骤1：创建实时音视频 TRTC 应用

1. 注册腾讯云账号 并开通 实时音视频 。
2. 在 实时音视频控制台 单击**应用管理 > 创建应用**创建新应用。

步骤2：获取 TRTC 密钥信息

1. 在**应用管理 > 应用信息**中获取 SDKAppID 信息。



2. 在**应用管理 > 快速上手**中获取应用的 secretKey 信息。

步骤3：下载 Api-Example

- 从 GitHub 获取

```
git clone https://github.com/LiteAVSDK/TRTC_UniApp.git
```

- 进入 Api-Example 目录

```
  cd TRTC_UniApp/Api-Example/
```

- 安装依赖

```
npm install
```

步骤4：配置 Api-Example 工程文件

1. 根据您下载的源码包，进入修改配置页。
2. 找到并打开 `./debug/GenerateTestUserSig.js` 文件。
3. 设置 GenerateTestUserSig.js 文件中的相关参数：
    - SDKAPPID：默认为0，请设置为实际的 SDKAppID。
    - SECRETKEY：默认为空字符串，请设置为实际的密钥信息。

```
import LibGenerateTestUserSig from './lib-generate-test-usersig-es.min.js';
/**…

const SDKAPPID = 0;   将 SDKAppID 粘贴到此处
/**
 * 签名过期时间，建议不要设置的过短
 * <p>
 * 时间单位：秒
 * 默认时间：7 x 24 x 60 x 60 = 604800 = 7 天
 */

const EXPIRETIME = 604800;
/**
 * 计算签名用的加密密钥，获取步骤如下：
 *
 * step1. 进入腾讯云实时音视频[控制台](https://console.cloud.tencent.com/rav )，如果还没有应用就创建一个，
 * step2. 单击“应用配置”进入基础配置页面，并进一步找到“帐号体系集成”部分。
 * step3. 点击“查看密钥”按钮，就可以看到计算 UserSig 使用的加密的密钥了，请将其拷贝并复制到如下的变量中
 *
 * 注意：该方案仅适用于调试Demo，正式上线前请将 UserSig 计算代码和密钥迁移到您的后台服务器上，以避免加密密钥泄露导致的流量盗用。
 * 文档：https://cloud.tencent.com/document/product/647/17275#Server
 */

const SECRETKEY = '';   将 密钥 粘贴到此处
/*
 * Module:   GenerateTestUserSig
 *
 * Function: 用于生成测试用的 UserSig，UserSig 是腾讯云为其云服务设计的一种安全保护签名。
 *           其计算方法是对 SDKAppID、UserID 和 EXPIRETIME 进行加密，加密算法为 HMAC-SHA256。
 *
 * Attention: 请不要将如下代码发布到您的线上正式版本的 App 中，原因如下：
 *
 *            本文件中的代码虽然能够正确计算出 UserSig，但仅适合快速调通 SDK 的基本功能，不适合线上产品，
 *            这是因为客户端代码中的 SECRETKEY 很容易被反编译逆向破解，尤其是 Web 端的代码被破解的难度几乎为零。
 *            一旦您的密钥泄露，攻击者就可以计算出正确的 UserSig 来盗用您的腾讯云流量。
 *
 *            正确的做法是将 UserSig 的计算代码和加密密钥放在您的业务服务器上，然后由 App 按需向您的服务器获取实时算出的 UserSig。
 *            由于破解服务器的成本要高于破解客户端 App，所以服务器计算的方案能够更好地保护您的加密密钥。
```

> **注意：**
> - 本文提到的生成 UserSig 的方案是在客户端代码中配置 SECRETKEY，该方法中 SECRETKEY 很容易被反编译逆向破解，一旦您的密钥泄露，攻击者就可以盗用您的腾讯云流量，因此该方法仅适合本地跑通 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 。

步骤5：运行

1. 制作自定义调试基座

> **说明**
> uni-app 官方教程请参见 什么是自定义调试基座及使用说明 。

1.1 选择**运行** > **运行到手机或模拟器** > **制作自定义调试基座**菜单。

1.2 在弹出的界面中，参考 `uni-app` 教程，填写相关信息，并单击**打包**进行云打包。

2. **切换运行基座为自定义调试基座**

在自定义调试基座选择**运行 > 运行到手机或模拟器 > 运行基座选择 > 自定义调试基座**菜单。

3. **运行**
   在**运行 > 运行到手机或模拟器**菜单，选择自己的设备，并运行。

常见问题

1.【官方】腾讯云实时音视频 SDK 和【官方】腾讯云原生音视频插件的区别?

- 【官方】腾讯云实时音视频 SDK 就是 `TRTC SDK`。【官方】腾讯云原生音视频插件 就是 `TUICalling`。
- `TRTC SDK` 是音视频基础能力，无 UI 和逻辑。适用于比较复杂的场景，或者定制化比较强的场景。用现有的 `TUICalling` 无法满足的需求，就可以使用 `TRTC SDK` 自己搭建。
- `TUICalling` 插件包含 UI 和逻辑，专用于 1V1、多人音视频通话场景，接入简单。

2.【官方】腾讯云实时音视频 SDK 支持小程序吗?

建议请参见 TRTCSimpleDemoUniApp，用uni-app 开发 微信小程序仍需开微信小程序通推拉流标签使用权限，目前暂时只开放给以下有限 类目。

申请开通

暂只针对国内主体如下类目的小程序开放，需要先通过类目审核，再在小程序管理后台，「开发」-「接口设置」中自助开通该组件权限。

一级类目/主体类型	二级类目	小程序内容场景
社交	直播	涉及娱乐性质，如明星直播、生活趣事直播、宠物直播等。选择该类目后首次提交代码审核，需经当地互联网主管机关审核确认，预计审核时长7天左右
教育	在线视频课程	网课、在线培训、讲座等教育类直播
医疗	互联网医院，公立医疗机构，其他私立医疗机构	问诊、大型健康讲座等直播
金融	银行、信托、公募基金、私募基金、证券/期货、证券、期货投资咨询、保险、征信业务、新三板信息服务平台、股票信息服务平台（港股/美股）、消费金融、融资担保	金融产品视频客服理赔、金融产品推广直播等
汽车	汽车预售服务	汽车预售、推广直播
政府主体账号	/	政府相关工作推广直播、领导讲话直播等
IT科技	多方通信；音视频设备	为多方提供电话会议/视频会议等服务；智能家居场景下控制摄像头
房地产服务	房地产营销	房地产营销直播服务、在线音视频带看等
商业服务	公证	在线业务办理等

3. 实例里面 app.vue 中的 aegis-weex-sdk 有什么用?

Demo 里的 `aegis-weex-sdk` 主要是用来对 Demo 做性能监控统计。

4. 创建实例 this.trtcCloud = TrtcCloud.createInstance(); 和 this.trtcCloud = new TrtcCloud(); 有什么区别?

创建 trtcCloud 单例，只能通过 `TrtcCloud.createInstance()` 实例化一个 TrtcCloud 对象。

5. startRemoteView 在什么时候调用?

建议在 onUserVideoAvailable 事件回调中调用 startRemoteView 。

6. 本地预览如何自定义样式?

建议通过 view 包裹 `trtc-local-view、trtc-remote-view` 。然后设置 view 的样式即可。

7. startLocalPreview 不传 viewId 可以吗?

startLocalPreview 可以不传 viewId。但是页面需要有 trtc-local-view 组件，并且它的 viewId 绑定的是本地用户 ID。

8. startLocalPreview 报 -2 错误?

- 错误原因：没有实例化本地预览的 view。
- 解决方案：
  - 首先必须是 .nvue 文件，详见 。
  - 页面必须使用 `trtc-local-view` 并且绑定了 viewId。

9. 【官方】腾讯云实时音视频 SDK 和 livepusher 推流模块同时使用，打包冲突

uni-app 官网文档介绍 第三方 SDK 使用 LiteAVSDK（weex_livepusher-release.aar）做直播推流，和 【官方】腾讯云实时音视频 SDK 使用的 LiteAVSDK_TRTC 类冲突。和 uni-app 技术沟通后，对方反馈不清楚 SDK 属于哪个版本类型，因此 uni-app 他们无法对 livepusher 进行升级。

快速接入
Web & H5

最近更新时间：2025-03-04 11:42:03

本文将介绍如何快速完成 Web&H5 RTC Engine 的接入，实现一个基本的音视频通话。

下载

npm

```
npm install trtc-sdk-v5 --save
```

yarn

```
yarn add trtc-sdk-v5
```

或者手动下载：

1. 下载 trtc.js 。
2. 将 trtc.js 复制到您的项目中。

用法

导入 TRTC SDK

```
import TRTC from 'trtc-sdk-v5';
```

如果您手动下载了 trtc.js，您应该使用 <script> 标签来导入 TRTC SDK。

```
<script src="/your_path/trtc.js"></script>
```

创建 TRTC 实例

调用 TRTC.create() 方法来创建一个 trtc 实例。

```
const trtc = TRTC.create();
```

进入房间

调用 trtc.enterRoom() 进入房间。 此方法通常在“开始通话”按钮的点击回调函数中调用。

参数	类型	描述
sdkAppId	`number`	您在 TRTC 控制台 中创建的音视频应用程序的 sdkAppId。
userId	`string`	您指定的用户 ID。
userSig	`string`	用户签名，请参见 UserSig 。

roomId	`number`	您指定的房间 ID，通常是一个唯一的房间 ID。

有关更详细的参数描述，请参考接口文档 trtc.enterRoom() 。

```
try {
  await trtc.enterRoom({ sdkAppId, userId, userSig, roomId: 8888 });
  console.log('enter room successfully');
} catch (error) {
  console.error('failed to enter room ' + error);
}
```

打开/关闭麦克风

调用 trtc.startLocalAudio() 打开麦克风并将音频发布到房间。

```
await trtc.startLocalAudio();
```

调用 trtc.stopLocalAudio 关闭麦克风并取消发布。

```
await trtc.stopLocalAudio();
```

打开/关闭摄像头

调用 trtc.startLocalVideo() 打开摄像头并将视频发布到房间。

```
// 要预览摄像头图像，您需要在 DOM 中放置一个 HTML 元素，
// 这可以是一个 div 标签，假设其 ID 为 local-video。
const view = 'local-video';
await trtc.startLocalVideo({ view });
```

调用 trtc.stopLocalVideo 关闭摄像头并取消发布

```
await trtc.stopLocalVideo();
```

播放远端音频

默认情况下，SDK 会自动播放远端音频，因此您无需手动调用任何 API 来播放远端音频。

> ⚠ **自动播放策略限制：**
> 如果用户在进入房间之前没有与页面进行交互，自动音频播放可能会因自动播放策略限制而失败。请参见 处理自动播放限制 。

以下代码片段展示了如何关闭自动播放音频，并手动控制远端音频播放。

```
trtc.on(TRTC.EVENT.REMOTE_AUDIO_AVAILABLE, event => {
    // 当你需要播放远端音频时调用该api
    trtc.muteRemoteAudio(event.userId, false);
    // 停止远端音频
    trtc.muteRemoteAudio(event.userId, true);
})
// 设置 autoReceiveAudio = false 以关闭自动音频播放。
```

```
await trtc.enterRoom({ ..., autoReceiveAudio: false });
```

播放远端视频

1. 在进入房间之前，监听 TRTC.EVENT.REMOTE_VIDEO_AVAILABLE 事件，以接收所有远端用户视频发布事件。
2. 当您收到该事件时，调用 trtc.startRemoteVideo() 来播放远端视频流。

```
trtc.on(TRTC.EVENT.REMOTE_VIDEO_AVAILABLE, ({ userId, streamType }) => {
  // 要播放视频图像，您需要在 DOM 中放置一个 HTMLElement，
  // 这可以是一个 div 标签，假设它的 id 是 ${userId}_${streamType}。
  const view = `${userId}_${streamType}`;
  trtc.startRemoteVideo({ userId, streamType, view });
});
```

退出房间

调用 trtc.exitRoom() 来退出房间。

```
await trtc.exitRoom();
// 在成功退出后，如果您不再需要使用 trtc 实例，可以调用 trtc.destroy 方法
// 及时销毁实例并释放相关资源。
// 被销毁的 trtc 实例无法再次使用，需要创建一个新的实例。
trtc.destroy();
```

处理被移出房间的场景

除了用户主动退出房间之外，用户也有可能因为如下原因被移出房间，此时 SDK 会抛出 KICKED_OUT 事件，这时不需要调用 `trtc.exitRoom()` 退房，SDK 自动进入退房状态。

1. `kick`：两个相同 userId 的用户进入相同房间，前一个进房的用户会被移出。同名用户同时进入同一房间是不允许的行为，可能会导致双方音视频通话异常，应避免出现这种情况。
2. `banned`：通过服务端的 RemoveUser | RemoveUserByStrRoomId 接口将某个用户移出某个 TRTC 房间。该用户会收到被移事件，reason 为 `banned`。
3. `room-disband`：通过服务端的 DismissRoom | DismissRoomByStrRoomId 接口将某个 TRTC 房间解散，解散房间之后，该房间的所有用户都会收到被移事件，reason 为 `room-disband`。

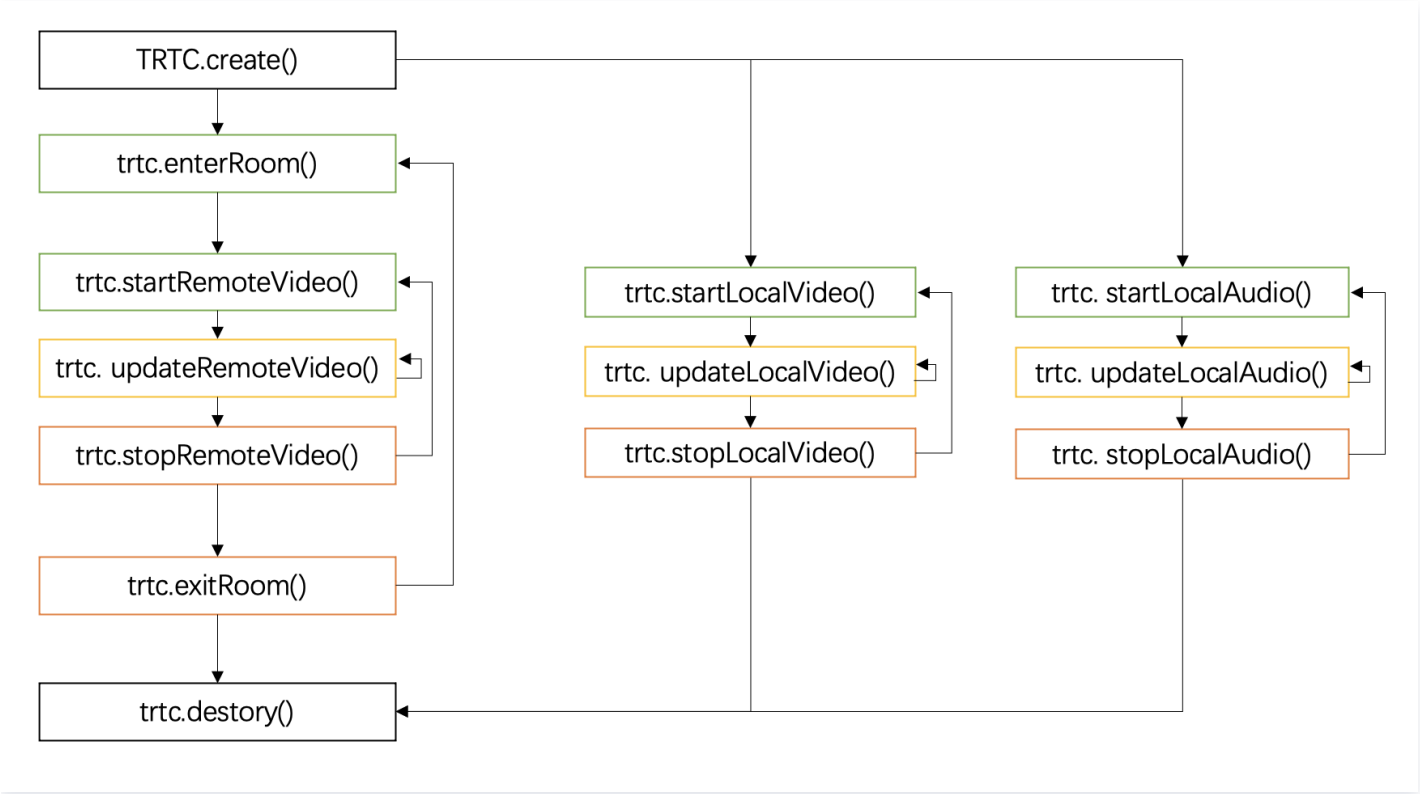
```
trtc.on(TRTC.EVENT.KICKED_OUT, error => {
  console.error(`kicked out, reason:${error.reason}, message:${error.message}`);
});
```

API Overview

- TRTC 是 TRTC SDK 的主要入口，提供创建 trtc 实例(TRTC.create)，TRTC.getCameraList，TRTC.getMicrophoneList，TRTC.isSupported 等。
- trtc 实例提供实时音视频通话的核心功能。
  - 进入房间 trtc.enterRoom
  - 退出房间 trtc.exitRoom
  - 打开/关闭麦克风 trtc.startLocalAudio/trtc.stopLocalAudio
  - 打开/关闭摄像头 trtc.startLocalVideo/trtc.stopLocalVideo
  - 播放/停止远端音频 trtc.muteRemoteAudio

- 播放/停止远端视频 trtc.startRemoteVideo / trtc.stopRemoteVideo

API 生命周期



联系我们

如果您是开发者，欢迎您加入我们的 RTC Engine 技术交流平台 zhiliao，进行技术交流和产品沟通。

Android

最近更新时间：2025-05-16 14:27:12

本文将介绍如何快速完成 Android RTC Engine 的接入，实现一个基本的音视频通话。

环境准备

- Android Studio 3.5+。
- Android 8.1（SDK API 27）及以上系统。

接入指引

步骤1. 导入 SDK

1. 在 `app/build.gradle` 中 **dependencies** 添加对 **TRTC SDK** 的依赖。

```
dependencies {
    // ...项目其他依赖
    implementation 'com.tencent.liteav:LiteAVSDK_TRTC:latest.release' // 添加 TRTC SDK 依赖
}
```

2. 在 `app/build.gradle` 中 **defaultConfig** 指定项目的 **CPU** 架构，以支持 **armeabi-v7a/arm64-v8a** 架构的设备。

```
android {
    defaultConfig {
        // ...其他默认配置
        ndk {
            abiFilters "armeabi-v7a", "arm64-v8a" // 支持 armeabi-v7a 和 arm64-v8a 架构
        }
    }
}
```

完成以上配置后，单击 `Sync Now` 后 SDK 将自动集成到目标工程中。

步骤2. 配置工程

1. 进入 `AndroidManifest.xml` 文件，添加 **TRTC SDK** 所需权限。

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-feature android:name="android.hardware.camera.autofocus" />
```

> ⚠ **注意：**
> 请勿设置 `android:hardwareAccelerated="false"`，关闭硬件加速导致视频流无法渲染。

2. 进入 `proguard-rules.pro` 文件，将 **TRTC SDK** 相关类及其成员加入不混淆名单。

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

步骤3. 创建 TRTC 实例

音视频功能的正常使用需要请求**摄像头**和**麦克风**权限，推荐在成功请求后创建 TRTC 实例。

```
private TRTCCloud mCloud; // 声明成员变量

// 创建 TRTC 实例并设置监听
mCloud = TRTCCloud.sharedInstance(getApplicationContext());
mCloud.addListener(new TRTCCloudListener() {
    @Override
    public void onError(int errCode, String errMsg, Bundle extraInfo) {
        super.onError(errCode, errMsg, extraInfo);
        String notificaiton = "Error Code: " + errCode + ", Error Message: " + errMsg + ",
extraInfo: " + extraInfo;
        Toast.makeText(getApplicationContext(), notificaiton, Toast.LENGTH_LONG).show();
    }

    @Override
    public void onEnterRoom(long result) {
        super.onEnterRoom(result);
        if(result > 0) {
            Toast.makeText(getApplicationContext(), "进房成功！", Toast.LENGTH_LONG).show();
        } else {
            Toast.makeText(getApplicationContext(), "进房失败！", Toast.LENGTH_LONG).show();
        }
    }
});
```

步骤4. 进入房间

1. 在进入房间之前需要请求**摄像头**和**麦克风**权限。

```
private static final int REQUEST_CAMERA_AND_MICROPHONE = 1; //
ActivityCompat.requestPermissions() 方法所需 requestCode 参数

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    // 请求摄像头和麦克风权限
    requestCameraAndMicrophonePermission();
}

private void requestCameraAndMicrophonePermission() {
    String[] permissions = {android.Manifest.permission.CAMERA,
android.Manifest.permission.RECORD_AUDIO};
    ActivityCompat.requestPermissions(this, permissions, REQUEST_CAMERA_AND_MICROPHONE);
}

@Override
```

```
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions,
@NonNull int[] grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    if (requestCode == REQUEST_CAMERA_AND_MICROPHONE) {
        boolean allPermissionsGranted = true;
        for (int grantResult : grantResults) {
            if (grantResult != PackageManager.PERMISSION_GRANTED) {
                allPermissionsGranted = false;
                break;
            }
        }
        if (allPermissionsGranted) {
            // 申请设备权限成功，创建 TRTC 实例并使用音视频所支持的功能
        } else {
            // 申请设备权限失败，可做出相应提示
            Toast.makeText(getApplicationContext(), "获取权限失败！",
Toast.LENGTH_LONG).show();
        }
    }
}
```

2. 设置进房参数 `TRTCParams` 并调用 `enterRoom` 即可成功进入房间，该方法通常在点击**开始通话**按钮后调用。

参数	类型	描述
sdkAppId	number	您在 实时音视频控制台 中创建的音频和视频应用程序的 sdkAppId。
userId	string	您指定的用户 ID。
userSig	string	用户签名，请参见 UserSig 。
roomId	number	您指定的房间 ID，通常是唯一的房间 ID。

有关更详细的参数描述，请参考接口文档 enterRoom 。

```
// 将以下 trtcParams 参数修改为自己的参数
TRTCCloudDef.TRTCParams trtcParams = new TRTCCloudDef.TRTCParams();
trtcParams.sdkAppId = 1400000123;
trtcParams.userId = "denny";
trtcParams.userSig = "xxx";
trtcParams.roomId = 123321;

// 对于多人视频通话场景，推荐使用 TRTC_APP_SCENE_LIVE
mCloud.enterRoom(trtcParams, TRTCCloudDef.TRTC_APP_SCENE_LIVE);
```

步骤5. 打开/关闭摄像头

1. 在布局文件中添加 `TXCloudVideoView` 视图组件，用于显示视频流内容。

```
<com.tencent.rtmp.ui.TXCloudVideoView
  android:id="@+id/txcvv_main_local"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  app:layout_constraintBottom_toBottomOf="parent"
```

```
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

2. 设置本地预览的渲染参数 `setLocalRenderParams` ，并调用 `startLocalPreview` 进行本地预览，成功调用 `enterRoom` 后开始推流。

```
// 设置本地预览渲染参数
TRTCCloudDef.TRTCRenderParams trtcRenderParams = new TRTCCloudDef.TRTCRenderParams();
trtcRenderParams.fillMode   = TRTCCloudDef.TRTC_VIDEO_RENDER_MODE_FILL; // 渲染模式为填充
trtcRenderParams.mirrorType = TRTCCloudDef.TRTC_VIDEO_MIRROR_TYPE_AUTO; // 镜像类型为自动
mCloud.setLocalRenderParams(trtcRenderParams);

// 对采集内容进行本地预览
TXCloudVideoView cameraVideo = findViewById(R.id.txcvv_main_local);
mCloud.startLocalPreview(true, cameraVideo);
```

调用 `stopLocalPreview` 关闭摄像头预览并停止推送本地视频信息。

```
mCloud.stopLocalPreview();
```

3. 添加**“切换前后摄像头”**、**“设置对焦模式”**、**“闪光灯”**等设备扩展功能的使用。

```
// 默认开启前置摄像头，切换为后置摄像头
TXDeviceManager manager = mCloud.getDeviceManager();
if(manager.isFrontCamera()) {
    manager.switchCamera(false);
}

// 切换为前置摄像头
TXDeviceManager manager = mCloud.getDeviceManager();
manager.switchCamera(true);
```

```
// 若设备支持自动识别人脸位置，开启自动对焦功能
TXDeviceManager manager = mCloud.getDeviceManager();
if (manager.isAutoFocusEnabled()) {
    manager.enableCameraAutoFocus(true);
}

// 关闭自动对焦功能
TXDeviceManager manager = mCloud.getDeviceManager();
manager.enableCameraAutoFocus(false);
```

```
// 切换后置摄像头时可开启闪光灯
TXDeviceManager manager = mCloud.getDeviceManager();
manager.enableCameraTorch(true);

// 关闭闪光灯
TXDeviceManager manager = mCloud.getDeviceManager();
```

```
manager.enableCameraTorch(false);
```

步骤6. 打开/关闭麦克风

调用 `startLocalAudio` 开启麦克风采集，请根据您的需求**选择以下其中一个声音质量参数** `Quality` 。

```
// 开启麦克风采集，设置当前场景为：语音模式
// 具有高的噪声抑制能力，有强有弱的网络阻力
mCloud.startLocalAudio(TRTCCloudDef.TRTC_AUDIO_QUALITY_SPEECH );
```

```
// 开启麦克风采集，设置当前场景为：音乐模式
// 为获得高保真度，低音质损失，建议配合专业声卡使用
mCloud.startLocalAudio(TRTCCloudDef.TRTC_AUDIO_QUALITY_MUSIC);
```

调用 `stopLocalAudio` 关闭麦克风采集并停止推送本地音频信息。

```
mCloud.stopLocalAudio();
```

步骤7. 播放/停止视频流

1. 在进入房间之前对 onUserVideoAvailable 进行监听，当您收到 `onUserVideoAvailable(userId, true)` 通知时，表示该路画面已经有可播放的视频帧到达。
2. 调用 `startRemoteView` 对远端采集画面进行播放。

```
// 播放远端画面
TXCloudVideoView cameraVideo = findViewById(R.id.txcvv_main_local);
mCloud.startRemoteView("denny", TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG, cameraVideo); //
以高清大画面播放远端采集视频内容
```

调用 `stopRemoteView` 停止播放远端画面。

```
// 停止播放
mCloud.stopRemoteView("denny", TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG); // 仅停止播放
denny 画面
mCloud.stopAllRemoteView(); // 停止播放所有视频
```

步骤8. 播放/停止音频流

默认情况下，SDK 将自动播放远程音频，因此您不需要调用任何 API 来手动播放它。
但当您不喜欢自动播放音频时，可以调用 `muteRemoteAudio` 选择播放或停止远端声音。

```
// 静音
mCloud.muteRemoteAudio("denny", true); // 仅静音 denny

mCloud.muteAllRemoteAudio(true); // 静音所有远端用户
```

```
// 取消静音
mCloud.muteRemoteAudio("denny", false); // 仅取消静音 denny
```

```
mCloud.muteAllRemoteAudio(false); // 取消静音所有远端用户
```

步骤9. 退出房间

调用 `exitRoom` 退出当前的房间，TRTC SDK 会在退房结束后通过 `onExitRoom` 回调事件通知您。

```
// 退出当前房间
mCloud.exitRoom();

// 监听 `onExitRoom` 回调
@Override
public void onExitRoom(int reason) {
    if (reason == 0) {
        Log.d(TAG, "Exit current room by calling the 'exitRoom' api of sdk ...");
    } else if (reason == 1) {
        Log.d(TAG, "Kicked out of the current room by server through the restful api...");
    } else if (reason == 2) {
        Log.d(TAG, "Current room is dissolved by server through the restful api...");
    }
}
```

常见问题

- 您可以在 API 概览 查看所有函数列表及其描述。
- 如果您的接入和使用中遇到问题，请参见 常见问题 。

联系我们

如果您是开发者，欢迎您加入我们的 RTC Engine 技术交流平台 zhiliao，进行技术交流和产品沟通。

iOS

最近更新时间：2025-06-17 12:12:12

本文将介绍如何快速完成 Objective-C 版本的 iOS RTC Engine 的接入，实现一个基本的音视频通话。

环境准备

- Xcode 13.0+。
- iOS 9.0 以上的 iPhone 或者 iPad 真机。
- 项目已配置有效的开发者签名。

接入指引

步骤1. 导入 SDK

1. 在终端窗口执行以下命令安装 **CocoaPods** ，如果您已经完成 **CocoaPods** 的安装，可以跳过该步。

```
sudo gem install cocoapods
```

2. 在终端窗口中进入**项目根目录**，执行以下命令为项目创建 **Podfile** 文件。

```
pod init
```

3. 编辑并保存 **Podfile** 内容如下。

```
platform :ios, '8.0'

# 将 App 修改为您项目的名称
target 'App' do
pod 'TXLiteAVSDK_TRTC', :podspec =>
'https://liteav.sdk.qcloud.com/pod/liteavsdkspec/TXLiteAVSDK_TRTC.podspec'
end
```

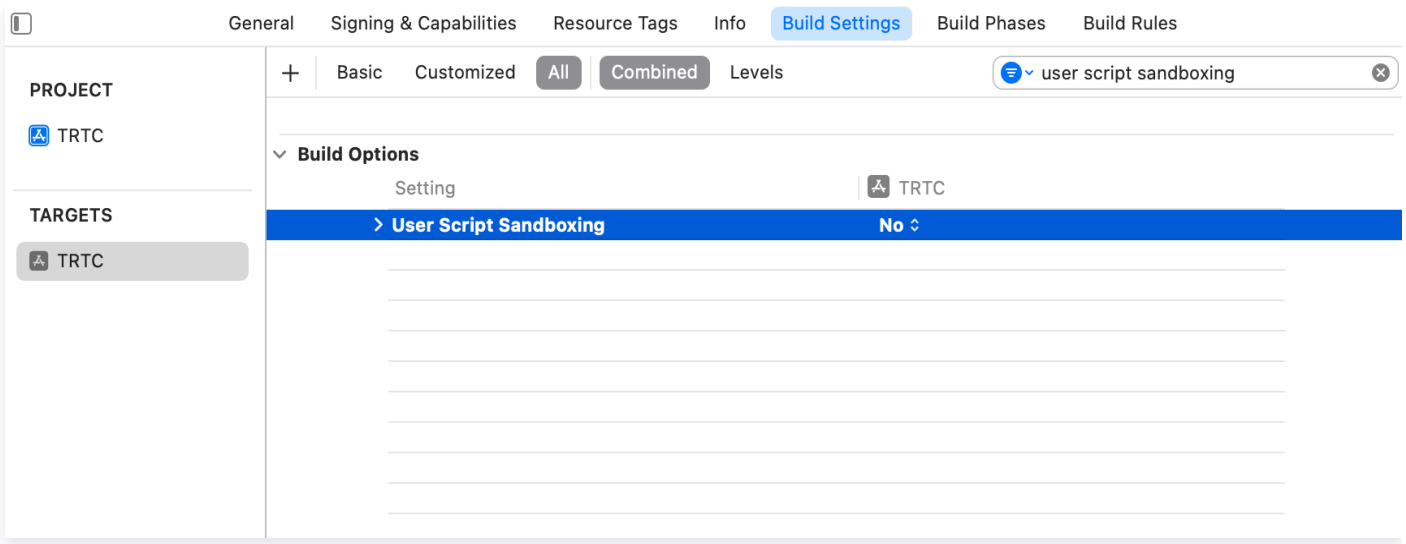
4. 在终端窗口中执行以下命令更新本地库文件，完成对 **TRTC SDK** 的下载。

```
pod install
```

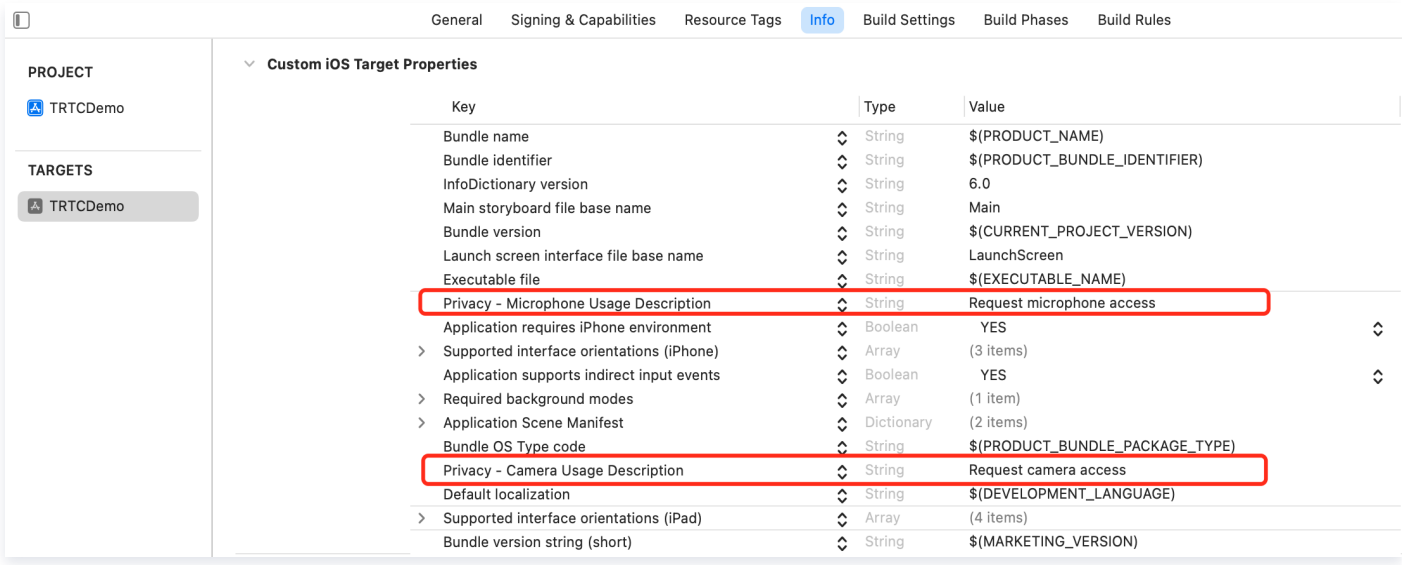
> **说明：**
> `pod install` 执行完成，会生成新的 **.xcworkspace** 工程文件，双击打开**.xcworkspace** 文件进行后续工作。

步骤2. 配置项目

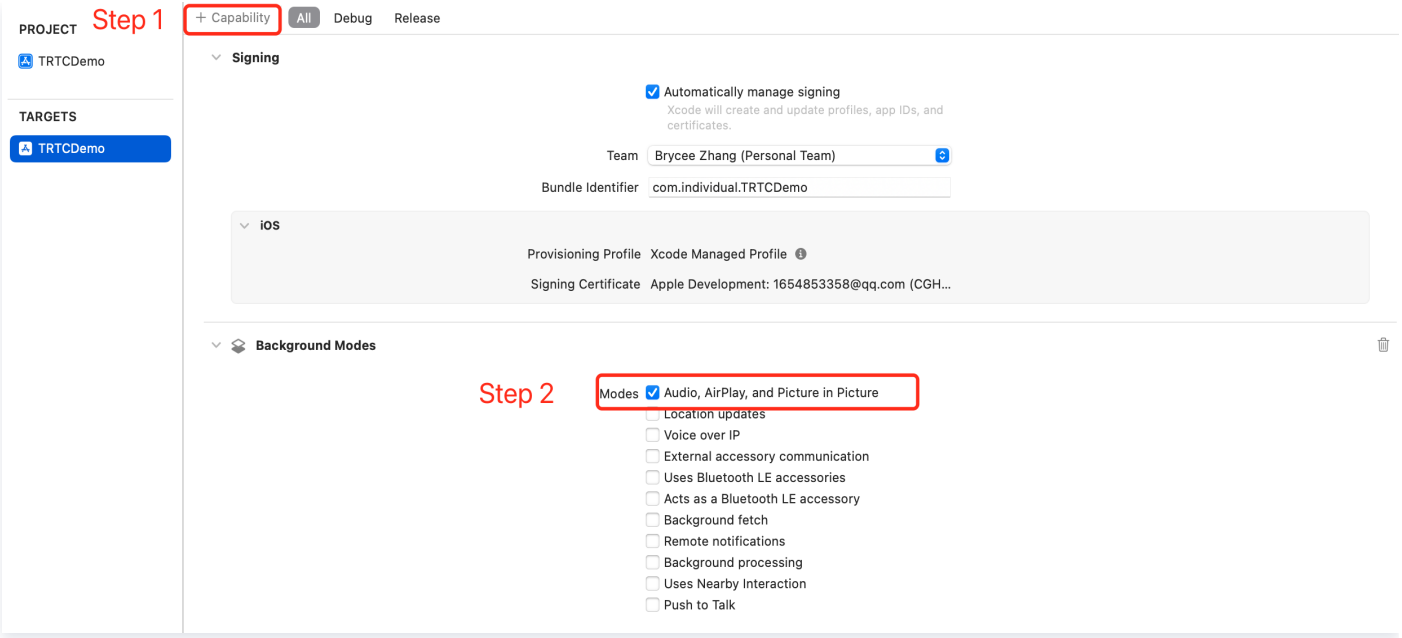
1. 打开 `.xcworkspace` 工程文件后，在 **Xcode** 的导航栏中点击左侧的 **Project Navigator**，点击您的项目名称并确保在编辑区域选择正确的 **TARGETS**。
2. 在 `Build Settings` 选项卡中搜索 **User Script Sandboxing**，将其值置为 `No` ，以允许用户脚本访问更广泛的系统资源和文件。



3. 在 `Info.plist` 选项卡中添加 **Privacy-Microphone Usage Description** 和 **Privacy-Microphone Usage Description**，并填入 **Microphone/Camera** 使用的目标提示语，获取麦克风和摄像头的使用权限。



4. 在 `Signing & Capabilities` 选项卡中添加 **Background Modes**，并勾选 **Audio，AirPlay and Picture in Picture**，以允许应用在后台运行音频、AirPlay和画中画功能。



步骤3. 创建 TRTC 实例

1. 在 `AppDelegate.h` 文件中引入 TRTC SDK。

```
@import TXLiteAVSDK_TRTC; // 引入 TRTC SDK 模块
```

2. 在 `AppDelegate.h` 文件中声明 TRTCCloud 属性和 TRTCCloudDelegate 接口。

```
@interface AppDelegate : UIResponder <UIApplicationDelegate, TRTCCloudDelegate> // 添加 TRTCCloudDelegate 接口声明

@property (nonatomic, strong) TRTCCloud *trtcCloud; // 添加 TRTCCloud 属性
```

3. 进入 `AppDelegate.m` 文件后，在 `didFinishLaunchingWithOptions` 方法中调用 `sharedInstance` 创建 TRTC 实例，并设置事件监听。

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    // 创建 TRTC 实例并设置监听
    _trtcCloud = [TRTCCloud sharedInstance];
    _trtcCloud.delegate = self;

    return YES;
}

// 监听并处理 `onError` 事件
- (void)onError:(TXLiteAVError)errCode
         errMsg:(nullable NSString *)errMsg
        extInfo:(nullable NSDictionary *)extInfo{
    // 处理 `Error` 事件，建议对 OnError 的信息通过 'toastTip:' 进行提示
    NSString *errCodeStr = [NSString stringWithFormat:@"%ld", (long)errCode];
    NSString *extInfoStr = extInfo ? [extInfo description] : @"";
    NSString *notification = [NSString stringWithFormat:@"%@, %@, %@", errCodeStr, errMsg, extInfoStr];
```

```
    [self toastTip:notification];
}

// 实现 'toastTip:'
- (void)toastTip:(NSString *)tip {
    UIAlertController *alert = [UIAlertController alertControllerWithTitle:@"Tip"
                                                                    message:tip

preferredStyle:UIAlertControllerStyleAlert];
    UIAlertAction *okAction = [UIAlertAction actionWithTitle:@"Confirm"
                                                       style:UIAlertActionStyleDefault
                                                     handler:nil];
    [alert addAction:okAction];
    [self.window.rootViewController presentViewController:alert animated:YES
completion:nil];

    dispatch_after(dispatch_time(DISPATCH_TIME_NOW, (int64_t)(2.0 * NSEC_PER_SEC)),
dispatch_get_main_queue(), ^{
        [alert dismissViewControllerAnimated:YES completion:nil];
    });
}
```

步骤4. 进入房间

设置进房参数 `TRTCParams` 并调用 `enterRoom` 即可成功进入房间，该方法通常在点击**开始通话按钮**后调用。

参数	类型	描述
sdkAppId	number	您在 TRTC 控制台 中创建的音频和视频应用程序的sdkAppId。
userId	string	您指定的用户 ID。
userSig	string	用户签名，请参见 UserSig 。
roomId	number	您指定的房间 ID，通常是唯一的房间 ID。

有关更详细的参数描述，请参考接口文档 enterRoom 。

```
#import "AppDelegate.h" // 导入 "AppDelegate.h" 文件

- (void)enterRoom {
    // 将以下 trtcParams 参数修改为自己的参数
    TRTCParams *trtcParams = [[TRTCParams alloc] init];
    trtcParams.sdkAppId = 1400000123;
    trtcParams.userId = @"denny";
    trtcParams.userSig = @"";
    trtcParams.roomId = 123321;

    // 对于多人视频通话场景，推荐使用 TRTC_APP_SCENE_LIVE
    AppDelegate *appDelegate = (AppDelegate *)[[UIApplication sharedApplication] delegate];
    [appDelegate.trtcCloud enterRoom:trtcParams appScene:TRTCAppSceneLIVE];
}

// 监听 `onEnterRoom` 事件
```

```
- (void)onEnterRoom:(NSInteger)result {
    // 处理 onEnterRoom 事件
}
```

步骤5. 打开/关闭摄像头

1. 在 `ViewController.h` 文件中声明 UIWindow 和 UIView 属性。

```
@property (strong, nonatomic) UIWindow *window; // 添加 UIWindow 属性
@property (nonatomic, strong) UIView *localCameraVideoView; // 添加 UIView 属性
```

2. 设置本地预览的渲染参数 `setLocalRenderParams` ，并调用 `startLocalPreview` 进行本地预览，成功调用 `enterRoom` 后开始推流。

```
- (void)startLocalPreview {
    // 设置本地预览渲染参数
    TRTCRenderParams *trtcRenderParams = [[TRTCRenderParams alloc] init];
    trtcRenderParams.fillMode   = TRTCVideoFillMode_Fill;
    trtcRenderParams.mirrorType = TRTCVideoMirrorTypeAuto;

    AppDelegate *appDelegate = (AppDelegate *)[[UIApplication sharedApplication]
delegate];
    [appDelegate.trtcCloud setLocalRenderParams:trtcRenderParams];

    // 对采集内容进行本地预览
    [appDelegate.trtcCloud startLocalPreview:YES view:self.localCameraVideoView];
}
```

调用 `stopLocalPreview` 关闭摄像头预览并停止推送本地视频信息。

```
AppDelegate *appDelegate = (AppDelegate *)[[UIApplication sharedApplication] delegate];
[appDelegate.trtcCloud stopLocalPreview];
```

3. 添加“切换前后摄像头”、“设置对焦模式”、“闪光灯”等设备扩展功能的使用。

```
// 默认开启前置摄像头，切换为后置摄像头
AppDelegate *appDelegate = (AppDelegate *)[[UIApplication sharedApplication] delegate];
TXDeviceManager * deviceManager = [appDelegate.trtcCloud getDeviceManager];
if([deviceManager isFrontCamera]) {
    [deviceManager switchCamera:false];
}

// 切换为前置摄像头
AppDelegate *appDelegate = (AppDelegate *)[[UIApplication sharedApplication] delegate];
TXDeviceManager * deviceManager = [appDelegate.trtcCloud getDeviceManager];
[deviceManager switchCamera:true];
```

```
// 若设备支持自动识别人脸位置，开启自动对焦功能
AppDelegate *appDelegate = (AppDelegate *)[[UIApplication sharedApplication] delegate];
TXDeviceManager * deviceManager = [appDelegate.trtcCloud getDeviceManager];
```

```
if([deviceManager isAutoFocusEnabled]) {
    [deviceManager enableCameraAutoFocus:true];
}

// 关闭自动对焦功能
AppDelegate *appDelegate = (AppDelegate *)[[UIApplication sharedApplication] delegate];
TXDeviceManager * deviceManager = [appDelegate.trtcCloud getDeviceManager];
[deviceManager enableCameraAutoFocus:false];
```

```
// 切换后置摄像头时可开启闪光灯
AppDelegate *appDelegate = (AppDelegate *)[[UIApplication sharedApplication] delegate];
TXDeviceManager * deviceManager = [appDelegate.trtcCloud getDeviceManager];
if(![deviceManager isFrontCamera]) {
    [deviceManager enableCameraTorch:true];
}

// 关闭闪光灯
AppDelegate *appDelegate = (AppDelegate *)[[UIApplication sharedApplication] delegate];
TXDeviceManager * deviceManager = [appDelegate.trtcCloud getDeviceManager];
[deviceManager enableCameraTorch:false];
```

4. 在 `ViewController.m` 中初始化 **localCameraVideoView**，并将其添加到当前视图的视图层次结构中。

```
#import "AppDelegate.h" // 引入 "AppDelegate.h" 文件

@interface ViewController ()

@end

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];
    // Do any additional setup after loading the view.

    // 初始化 localCameraVideoView
    self.localCameraVideoView = [[UIView alloc] initWithFrame:self.view.bounds];
    self.localCameraVideoView.backgroundColor = [UIColor blackColor];

    [self viewDidAppear];
}

- (void)viewDidAppear {
    AppDelegate *appDelegate = (AppDelegate *)[[UIApplication sharedApplication]
delegate];

    // 将 localCameraVideoView 添加到当前视图的视图层次结构中
    [self.view addSubview:self.localCameraVideoView];

    // 开启本地预览
    [appDelegate.trtcCloud startLocalPreview:YES view:self.localCameraVideoView];
}
```

```
@end
```

步骤6. 打开/关闭麦克风

调用 `startLocalAudio` 开启麦克风采集，请根据您的需求选择以下其中一个声音质量参数 `Quality` 。

```
// 开启麦克风采集，设置当前场景为：语音模式
// 具有高的噪声抑制能力，有强有弱的网络阻力
AppDelegate *appDelegate = (AppDelegate *)[[UIApplication sharedApplication] delegate];
[appDelegate.trtcCloud startLocalAudio:TRTCAudioQualitySpeech];
```

```
// 开启麦克风采集，设置当前场景为：音乐模式
// 为获得高保真度，低音质损失，建议配合专业声卡使用
AppDelegate *appDelegate = (AppDelegate *)[[UIApplication sharedApplication] delegate];
[appDelegate.trtcCloud startLocalAudio:TRTCAudioQualityMusic];
```

调用 `stopLocalAudio` 关闭麦克风采集并停止推送本地音频信息。

```
AppDelegate *appDelegate = (AppDelegate *)[[UIApplication sharedApplication] delegate];
[appDelegate.trtcCloud stopLocalAudio];
```

步骤7. 播放/停止视频流

1. 在进入房间之前对 onUserVideoAvailable 进行监听，当您收到 `onUserVideoAvailable(userId, YES)` 通知时，表示该路画面已经有可播放的视频帧到达。
2. 调用 `startRemoteView` 进行本地预览，成功调用 `enterRoom` 后开始推流。

```
- (void)startRemoteView {
    // 播放远端画面
    AppDelegate *appDelegate = (AppDelegate *)[[UIApplication sharedApplication]
delegate];
    [appDelegate.trtcCloud startRemoteView:@"denny" streamType:TRTCVideoStreamTypeBig
view:self.localCameraVideoView];
}
```

调用 `stopRemoteView` 停止播放远端画面。

```
// 停止播放
AppDelegate *appDelegate = (AppDelegate *)[[UIApplication sharedApplication] delegate];
[appDelegate.trtcCloud stopRemoteView:@"denny"]; // 仅停止播放 denny 画面
[appDelegate.trtcCloud stopAllRemoteView]; // 停止播放所有视频
```

步骤8. 播放/停止音频流

调用 `muteRemoteAudio` 选择播放或停止远端声音。

```
// 静音
AppDelegate *appDelegate = (AppDelegate *)[[UIApplication sharedApplication] delegate];
[appDelegate.trtcCloud muteRemoteAudio:@"denny" mute:YES]; // 仅静音 denny
```

```
[appDelegate.trtcCloud muteAllRemoteAudio:YES]; // 静音所有远端用户
```

```
// 取消静音
AppDelegate *appDelegate = (AppDelegate *)[[UIApplication sharedApplication] delegate];
[appDelegate.trtcCloud muteRemoteAudio:@"denny" mute:NO]; // 仅取消静音 denny

[appDelegate.trtcCloud muteAllRemoteAudio:NO]; // 取消静音所有远端用户
```

步骤9. 退出房间

调用 `exitRoom` 退出当前的房间，TRTC SDK 会在退房结束后通过 `onExitRoom` 回调事件通知您。

```
// 退出当前房间
AppDelegate *appDelegate = (AppDelegate *)[[UIApplication sharedApplication] delegate];
[appDelegate.trtcCloud exitRoom];

// 监听 `onExitRoom` 回调
- (void)onExitRoom:(NSInteger)reason {
    if (reason == 0) {
        NSLog(@"Exit current room by calling the 'exitRoom' api of sdk ...");
    } else if (reason == 1) {
        NSLog(@"Kicked out of the current room by server through the restful api...");
    } else if (reason == 2) {
        NSLog(@"Current room is dissolved by server through the restful api...");
    }
}
```

常见问题

- 您可以在 API 概览 查看所有函数列表及其描述。
- 如果您的接入和使用中遇到问题，请参见 常见问题 。

联系我们

如果您是开发者，欢迎您加入我们的 RTC Engine 技术交流平台 zhiliao，进行技术交流和产品沟通。

微信小程序

最近更新时间：2025-03-04 11:42:03

本文主要介绍如何快速地将腾讯云 TRTC Wechat SDK 集成到您的项目中。

> **说明：**
> - 自2023年02月15日起，如需使用微信小程序 SDK 音视频通话和互动直播，需开通订阅 TRTC 包月套餐 以解锁小程序SDK能力，详见包月套餐 功能与计费说明 。
> - **2023年02月15日前创建过 TRTC 应用的腾讯云账号下的所有应用（SdkAppId），作为缓冲期可免费使用微信小程序 SDK 能力至2023年04月01日。**

准备工作

集成 TRTC Wechat SDK 之前需要了解的事项。

环境要求

- 微信 App iOS 最低版本要求：7.0.9 。
- 微信 App Android 最低版本要求：7.0.8 。
- 小程序基础库最低版本要求：2.10.0 。
- 由于小程序测试号不具备 <live-pusher> 和 <live-player> 的使用权限，请使用企业小程序账号申请相关权限进行开发。
- 由于微信开发者工具不支持原生组件（即 <live-pusher> 和 <live-player> 标签），需要在真机上进行运行体验。
- 请使用原生小程序开发环境。如果想在 uniapp 开发环境开发音视频通话，可以参考使用 TUICallKit 组件 。

前提条件

1. 您已 注册腾讯云 账号，并完成 实名认证 。
2. **开通小程序类目与推拉流标签权限（如不开通则无法正常使用）。**
3. 出于政策和合规的考虑，微信暂未放开所有小程序对实时音视频功能（即 <live-pusher> 和 <live-player> 标签）的支持：
   - 小程序推拉流标签不支持个人小程序，只支持企业类小程序。
   - 小程序推拉流标签使用权限暂时只开放给有限 类目 。
   - 符合类目要求的小程序，需要在 微信公众平台 > **开发** > **开发管理** > **接口设置** 中自助开通该组件权限，如下图所示：

配置域名添加

在 微信公众平台 > 开发 > 开发管理 > 开发设置 > 服务器域名中设置 request合法域名和 socket合法域名，如下图所示：

- request 合法域名：

```
https://official.opensso.tencent-cloud.com
https://yun.tim.qq.com
https://cloud.tencent.com
https://webim.tim.qq.com
https://query.tencent-cloud.com
https://web.sdk.qcloud.com
```

- socket 合法域名：

```
wss://wss.im.qcloud.com
wss://wss.tim.qq.com
```

开始集成

SDK 提供了 ES Module 类型的模块。

1 导入 SDK 到项目中

1. 您需要在项目中使用 trtc-wx 包。

```
npm i trtc-wx-sdk --save
```

2. 在项目脚本里引入模块。此处可引入静态文件，也可通过 小程序构建npm 直接引入。

```
import TRTC from './static/trtc-wx'; // 静态文件引入
import TRTC from 'trtc-wx-sdk'; // 小程序构建npm引入
```

2 进入房间

2.1 初始化 TRTC 实例

```
import TRTC from 'trtc-wx-sdk';
// 生命周期函数--监听页面加载
onLoad(options) {
```

```
    this.TRTC = new TRTC(this)
},
```

2.2 绑定事件回调

参见 事件表 。

```
<live-pusher
        bindstatechange="_pusherStateChangeHandler"
        bindnetstatus="_pusherNetStatusHandler"
        binderror="_pusherErrorHandler"
        bindbgmstart="_pusherBGMStartHandler"
        bindbgmprogress="_pusherBGMProgressHandler"
        bindbgmcomplete="_pusherBGMCompleteHandler"
        bindaudiovolumenotify="_pusherAudioVolumeNotify"
/>
```

```
// 生命周期函数--监听页面加载
onLoad(options) {
    this.TRTC = new TRTC(this)
    this.bindTRTCRoomEvent()
},
// 事件监听
bindTRTCRoomEvent() {
    const TRTC_EVENT = this.TRTC.EVENT
    // 初始化事件订阅
    this.TRTC.on(TRTC_EVENT.LOCAL_JOIN, (event) => {
        console.log('* room LOCAL_JOIN', event)
    })
    this.TRTC.on(TRTC_EVENT.LOCAL_LEAVE, (event) => {
        console.log('* room LOCAL_LEAVE', event)
    })
    this.TRTC.on(TRTC_EVENT.ERROR, (event) => {
        console.log('* room ERROR', event)
    })
    // 远端用户加入
    this.TRTC.on(TRTC_EVENT.REMOTE_USER_JOIN, (event) => {
        console.error('* room REMOTE_USER_JOIN', event)
    })
    // 远端用户退出
    this.TRTC.on(TRTC_EVENT.REMOTE_USER_LEAVE, (event) => {
        console.error('* room REMOTE_USER_LEAVE', event)
    })
    // 远端用户推送视频
    this.TRTC.on(TRTC_EVENT.REMOTE_VIDEO_ADD, (event) => {
        console.log('* room REMOTE_VIDEO_ADD',  event)
    })
    // 远端用户取消推送视频
    this.TRTC.on(TRTC_EVENT.REMOTE_VIDEO_REMOVE, (event) => {
        console.log('* room REMOTE_VIDEO_REMOVE', event)
    })
    // 远端用户推送音频
```

```
    this.TRTC.on(TRTC_EVENT.REMOTE_AUDIO_ADD, (event) => {
        console.log('* room REMOTE_AUDIO_ADD', event)
    })
    // 远端用户取消推送音频
    this.TRTC.on(TRTC_EVENT.REMOTE_AUDIO_REMOVE, (event) => {
        console.log('* room REMOTE_AUDIO_REMOVE', event)
    })
    // 远端用户音量更新
    this.TRTC.on(TRTC_EVENT.REMOTE_AUDIO_VOLUME_UPDATE, (event) => {
        console.log('* room REMOTE_AUDIO_VOLUME_UPDATE', event)
    })
    // 本地用户音量更新
    this.TRTC.on(TRTC_EVENT.LOCAL_AUDIO_VOLUME_UPDATE, (event) => {
        console.log('* room LOCAL_AUDIO_VOLUME_UPDATE', event)
    })
},
// 请保持跟 wxml 中绑定的事件名称一致
_pusherStateChangeHandler(event) {
    this.TRTC.pusherEventHandler(event)
},
_pusherNetStatusHandler(event) {
    this.TRTC.pusherNetStatusHandler(event)
},
_pusherErrorHandler(event) {
    this.TRTC.pusherErrorHandler(event)
},
_pusherBGMStartHandler(event) {
    this.TRTC.pusherBGMStartHandler(event)
},
_pusherBGMProgressHandler(event) {
    this.TRTC.pusherBGMProgressHandler(event)
},
_pusherBGMCompleteHandler(event) {
    this.TRTC.pusherBGMCompleteHandler(event)
},
_pusherAudioVolumeNotify(event) {
    this.TRTC.pusherAudioVolumeNotify(event)
},
_playerStateChange(event) {
    this.TRTC.playerEventHandler(event)
},
_playerFullscreenChange(event) {
    this.TRTC.playerFullscreenChange(event)
},
_playerNetStatus(event) {
    this.TRTC.playerNetStatus(event)
},
_playerAudioVolumeNotify(event) {
    this.TRTC.playerAudioVolumeNotify(event)
}
```

2.3 进入音视频通话房间

需要先调用 createpusher 初始化 pusher 实例并配置初始参数，再调用 enterroom 获取新的 pusher 实例，通过 setData 赋值给 live-pusher 标签，之后调用 start 开始进入音视频通话房间。

> ⚠ **注意：**
> 进房成功后，将会产生音视频通话用量和对应费用，包括观众观看用量或若仅主播单人在房时的音频用量，详细计费请见 音视频通话计费说明 。

```
// 生命周期函数--监听页面加载
onLoad(options) {
    this.TRTC = new TRTC(this)
    this.bindTRTCRoomEvent()
    this.init()
    this.enterRoom(options)
},
init() {
  const pusherConfig = {
    beautyLevel: 9,
  }
  this.setData({
    pusher: this.TRTC.createPusher(pusherConfig)
  })
},
enterRoom(options) {
  const { roomID, sdkAppID, userID, userSig } = options
  this.setData({
    pusher: this.TRTC.enterRoom({
      roomID,
      sdkAppID,
      userID,
      userSig,
    }),
  }, () => {
     this.TRTC.getPusherInstance().start() // 开始推流并进入trtc房间
  })
},
```

```
 <live-pusher
        url="{{pusher.url}}"
        mode="{{pusher.mode}}"
        enable-camera="{{pusher.enableCamera}}"
        enable-mic="{{pusher.enableMic}}"
/>
```

3 订阅音视频流

在远端用户进入的回调中，更新 playerList，通过循环遍历。

```
// 远端用户推送视频
this.TRTC.on(TRTC_EVENT.REMOTE_VIDEO_ADD, (event) => {
  console.log('* room REMOTE_VIDEO_ADD',  event)
  const { player } = event.data
```

```
    // 开始播放远端的视频流，默认是不播放的
    this.setPlayerAttributesHandler(player, { muteVideo: false })
  })
  // 远端用户推送音频
  this.TRTC.on(TRTC_EVENT.REMOTE_AUDIO_ADD, (event) => {
    console.log('* room REMOTE_AUDIO_ADD', event)
    const { player } = event.data
    this.setPlayerAttributesHandler(player, { muteAudio: false })
  })
```

```
// 设置某个 player 属性
setPlayerAttributesHandler(player, options) {
  this.setData({
    playerList: this.TRTC.setPlayerAttributes(player.streamID, options),
  })
},
```

```
<view wx:for="{{playerList}}" wx:key="id">
  <live-player
    id="{{item.id}}"
    src= "{{item.src}}"
    data-userid="{{item.userID}}"
    data-streamid="{{item.streamID}}"
    data-streamtype="{{item.streamType}}"
  />
</view>
```

4 发布音视频流

在进入房间后，调用 getPusherInstance().start() 或者开启自动推流模式即可开始推流。

```
 enterRoom(options) {
    this.setData({
      pusher: this.TRTC.enterRoom(options),
    }, () => {
      this.TRTC.getPusherInstance().start() // 开始推流
    })
  },
```

```
 <live-pusher
    url="{{pusher.url}}"
    autopush="{{true}}"
/>
```

5 退出房间

5.1 主动退出当前房间

通话结束时调用 exitRoom 方法退出音视频通话房间，整个音视频通话会话结束。

```
exitRoom() {
    const result = this.TRTC.exitRoom()
    // 状态机重置，会返回更新后的pusher和playerList
    this.setData({
        pusher: result.pusher,
        playerList: result.playerList,
    })
},
```

5.2 被动退出当前房间

除了用户主动退出房间，在以下情况下，用户会收到 KICKED_OUT 事件，表示当前用户被动退出房间：

```
let onKickedout = function(event){
  console.log('被服务端踢出或房间被解散')
}
this.TRTC.on(EVENT.KICKED_OUT, onKickedout)
```

Flutter

最近更新时间：2025-06-12 09:09:12

本文将介绍如何快速完成 Flutter RTC Engine 的接入，实现一个基本的音视频通话。

环境准备

- Flutter 2.0 及以上版本。
- **Android 端开发：**
  - Android Studio 3.5及以上版本。
  - App 要求 Android 4.1及以上版本设备。
  - 请确保您的项目支持 CMake 3.13 及以上版本
- **iOS 端开发：**
  - Xcode 11.0及以上版本。
  - osx 系统版本要求 10.11 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1. 导入SDK

通过以下命令安装组件 tencent_rtc_sdk：

```
flutter pub add tencent_rtc_sdk
```

步骤2. 配置项目

iOS

1. 需要在 `Info.plist` 的第一级<dict>目录下加入对相机和麦克风的权限申请：

```
<key>NSCameraUsageDescription</key>
<string>授权摄像头权限才能正常视频通话</string>
<key>NSMicrophoneUsageDescription</key>
<string>授权麦克风权限才能正常语音通话</string>
```

2. 添加字段 `io.flutter.embedded_views_preview`，并设定值为 YES。

Android

1. 打开 `/android/app/src/main/AndroidManifest.xml` 文件。
2. 在其中添加如下权限：

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
<uses-permission android:name="android.permission.BLUETOOTH" />
```

```
<uses-permission android:name="android.permission.CAMERA" />
<uses-feature android:name="android.hardware.camera.autofocus" />
```

3. 如果您需要编译运行在 Android 平台，您还需要进行如下配置：

   首先，需要在工程的 `android/app/build.gradle` 文件中对应位置添加：

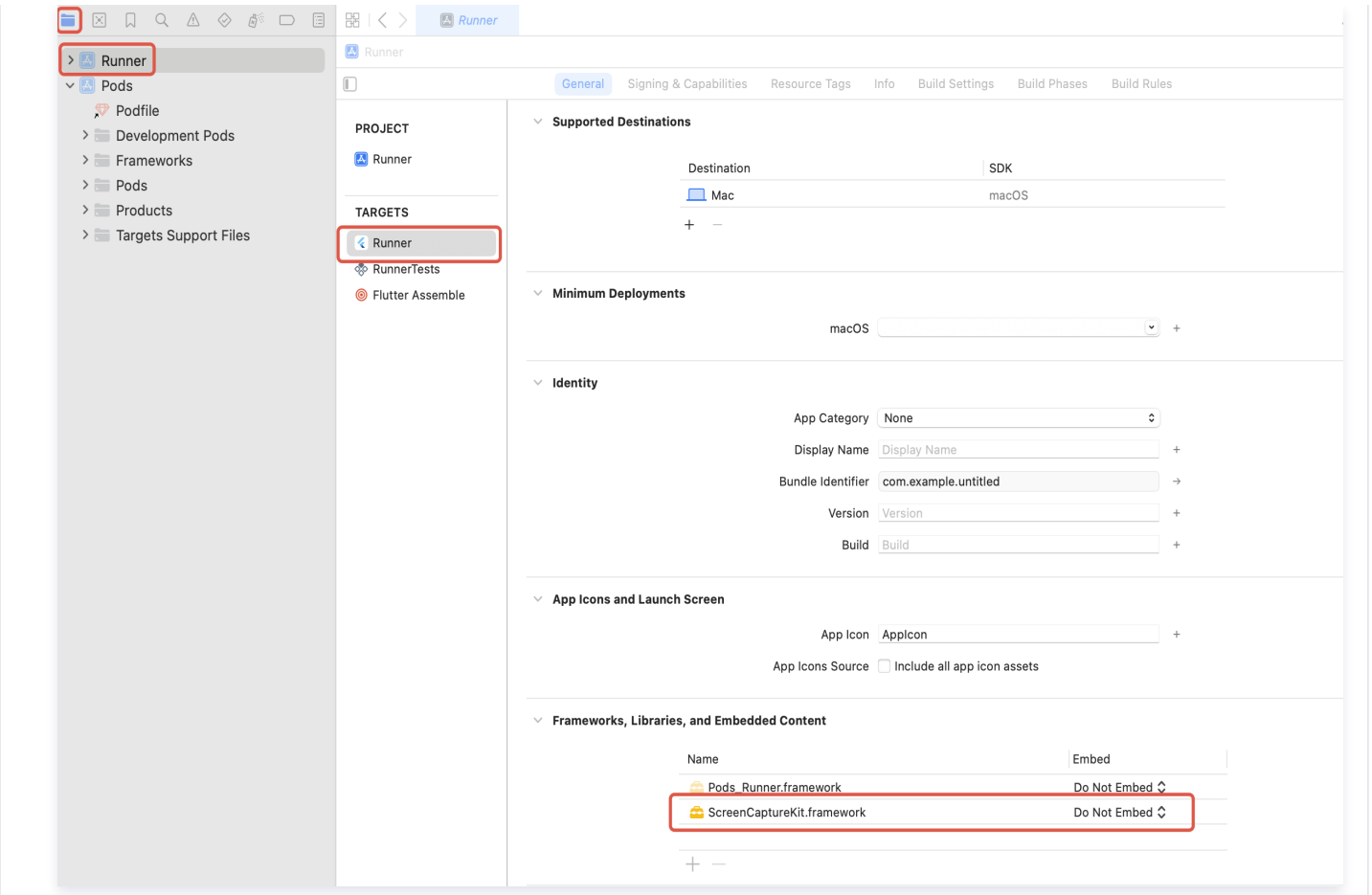
```
android {
  .....
  packagingOptions {
       pickFirst 'lib/**/libliteavsdk.so'
  }
  buildTypes {
       release {
          ......
          minifyEnabled true
          proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-
rules.pro'
       }
  }
}
```

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

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

macOS

1. 打开 `.xcworkspace` 工程文件后，在 Xcode 的导航栏中点击左侧的 Project Navigator，点击 Runner 并确保在编辑区域选择正确的 TARGETS。
2. 在 `General` 选项卡的 Frameworks, Libraries, and Embedded Content 部分添加 ScreenCaptureKit.framework。



说明：
如果您在接入过程中遇到问题，请参见 常见问题 。

步骤3. 创建 TRTC 实例

1. 声明成员变量

```
import 'package:tencent_rtc_sdk/trtc_cloud.dart';
import 'package:tencent_rtc_sdk/trtc_cloud_def.dart';
import 'package:tencent_rtc_sdk/trtc_cloud_listener.dart';
```

```
late TRTCCloud trtcCloud;
```

2. 调用初始化接口创建 TRTC 实例并设置事件回调。

```
// 创建 TRTC 实例
trtcCloud = (await TRTCCloud.sharedInstance())!;

// 创建 TRTCCloudListener 实例
TRTCCloudListener listener = TRTCCloudListener(
  // 根据需要实现对应的回调
  onError: (errCode, errMsg) {
    // TODO
```

```
  }
);

// 注册监听器，将 listener 绑定到 trtcCloud 实例
trtcCloud.registerListener(listener);
```

步骤4. 进入房间

1. 如果您在 Android 设备上运行程序，需要提前请求 **CAMERA** 和 **MICROPHONE** 权限。

```
if (!(await Permission.camera.request().isGranted) ||
    !(await Permission.microphone.request().isGranted)) {
  print('You need to obtain audio and video permission to enter');
  return;
}
```

> **说明：**
> 此处权限申请使用了第三方仓库 permission_handler 。

2. 在 TRTC 控制台 中获取到您应用的 sdkAppId，并且参见 用户鉴权 获取 userSig。
3. 在设置进房参数 **TRTCParams** 后，调用 `enterRoom` 接口函数即可进入房间。

- **主播角色**

```
trtcCloud.enterRoom(
    TRTCParams(
      sdkAppId: sdkAppId, // 请替换为您的 SDKAppID
      userId: "userId",                       // 请替换为您的 userid
      userSig: '',                            // 请替换为您的 userSig
      role: TRTCRoleType.anchor,
      roomId: 123123,                         // 请替换为您的 roomId
    ),
    TRTCAppScene.live
);
```

- **观众角色**

```
trtcCloud.enterRoom(
    TRTCParams(
      sdkAppId: sdkAppId, // 请替换为您的 SDKAppID
      userId: "userId",                       // 请替换为您的 userid
      userSig: '',                            // 请替换为您的 userSig
      role: TRTCRoleType.audience,
      roomId: 123123,                         // 请替换为您的 roomId
    ),
    TRTCAppScene.live
);
```

> **注意：**
> 如果以**观众角色**进入房间，**sdkAppId** 和 **roomId** 需要与主播端相同，而 **userId** 和 **userSig** 需要替换为自己的值。

步骤5. 打开摄像头

1. 在页面的 build 方法中对应位置添加 TRTCCloudVideoView：

```
import 'package:tencent_rtc_sdk/trtc_cloud_video_view.dart';
```

```
TRTCCloudVideoView(
  key: valueKey,
  onViewCreated: (viewId) {
    localViewId = viewId;
    // TODO
  },
),
```

> **说明：**
> `viewId` 为视频渲染控件 `TRTCCloudVideoView` 的唯一标识符。您可以以您想要的方式存储该标识符。此处使用 `localViewId` 来存储，用以后续渲染本地采集的视频流。

2. 在调用接口 `startLocalPreview` 打开摄像头预览之前，您可以通过通过调用接口 `setLocalRenderParams` 来设置本地预览的渲染参数。

```
// 设置本地预览渲染参数
trtcCloud.setLocalRenderParams(
  TRTCRenderParams(
    fillMode: TRTCVideoFillMode.fill,
    mirrorType: TRTCVideoMirrorType.auto,
    rotation: TRTCVideoRotation.rotation0,
  ),
);

// 对前置摄像头采集内容进行本地预览
trtcCloud.startLocalPreview(true, localViewId);

// 对后置摄像头采集内容进行本地预览
trtcCloud.startLocalPreview(false, localViewId);
```

调用 `stopLocalPreview` 关闭摄像头预览并停止推送本地视频信息。

```
trtcCloud.stopLocalPreview();
```

3. 您可以通过调用接口 `TXDeviceManager` 来完成“切换前后摄像头”、“设置对焦模式”、“闪光灯”等设备扩展功能的使用。

```
import 'package:tencent_rtc_sdk/tx_device_manager.dart';
```

```
// 获取设备管理器实例
TXDeviceManager manager = trtcCloud.getDeviceManager();

// 判断是否为前置摄像头
```

```
if (manager.isFrontCamera()) {
  // 切换为后置摄像头
  manager.switchCamera(false);
} else {
  // 切换为前置摄像头
  manager.switchCamera(true);
}
```

```
// 获取设备管理器实例
TXDeviceManager manager = trtcCloud.getDeviceManager();

// 设备是否支持自动识别人脸位置
if (manager.isAutoFocusEnabled()) {
  // 开启自动对焦功能
  manager.enableCameraAutoFocus(true);
} else {
  // 关闭自动对焦功能
  manager.enableCameraAutoFocus(false);
}
```

```
// 获取设备管理器实例
TXDeviceManager manager = trtcCloud.getDeviceManager();

// 切换后置摄像头时可开启闪光灯
manager.enableCameraTorch(true);

// 关闭闪光灯
manager.enableCameraTorch(false);
```

步骤6. 打开麦克风

您可以调用 `startLocalAudio` 来开启麦克风采集，该接口需要您通过 `quality` 参数确定采集模式，建议根据您的需求选择以下其中一个适合您项目的模式。

```
// 开启麦克风采集，设置当前场景为：语音模式
// 具有高的噪声抑制能力，有强有弱的网络阻力
trtcCloud.startLocalAudio(TRTCAudioQuality.speech);
```

```
// 开启麦克风采集，设置当前场景为：音乐模式
// 为获得高保真度，低音质损失，建议配合专业声卡使用
trtcCloud.startLocalAudio(TRTCAudioQuality.music);
```

调用 `stopLocalAudio` 关闭麦克风采集并停止推送本地音频信息。

```
trtcCloud.stopLocalAudio();
```

步骤7. 播放/停止视频流

1. 在进入房间之前对 onUserVideoAvailable 进行监听，当您收到 `onUserVideoAvailable(userId, true)` 通知时，表示该路画面已经有可播放的视频帧到达。

> **说明：**
> 此处假设可播放视频的用户为 `denny`，预计要将用户 `denny` 的视频流渲染到唯一标识符为 `remoteViewId` 的 `TRTCCloudVideoView` 控件中。

2. 您可以通过调用接口 `startRemoteView` 来播放远端用户的视频画面。

```
// 播放远端用户 denny 的主流视频画面
trtcCloud.startRemoteView("denny", TRTCVideoStreamType.big, remoteViewId);
```

之后，您通过调用接口 `stopRemoteView` 停止某个远端用户的视频，也可以通过接口 `stopAllRemoteView` 停止播放所有远端用户的视频。

```
// 停止播放远端用户 denny 的主流视频画面
trtcCloud.stopRemoteView("denny", TRTCVideoStreamType.big);
// 停止播放所有远端用户的视频画面
trtcCloud.stopAllRemoteView();
```

步骤8. 播放/停止音频流

默认情况下，SDK 将自动播放远程音频，因此您不需要调用任何 API 来手动播放它。
但当您不喜欢自动播放音频时，可以调用 `muteRemoteAudio/muteAllRemoteAudio` 选择播放或停止远端声音。

```
// 仅静音远端用户 denny
trtcCloud.muteRemoteAudio("denny", true);

// 静音所有远端用户
trtcCloud.muteAllRemoteAudio(true);
```

```
// 取消静音远端用户 denny
trtcCloud.muteRemoteAudio("denny", false);

// 取消静音所有远端用户
trtcCloud.muteAllRemoteAudio(false);
```

步骤9. 退出房间

调用 `exitRoom` 退出当前的房间：

```
trtcCloud.exitRoom();
```

TRTC SDK 会在退房结束后通过 `onExitRoom` 回调事件通知您。

```
TRTCCloudListener listener = TRTCCloudListener(
  onExitRoom: (reason) {
    // TODO
  }
```

```
);

trtcCloud.registerListener(listener);
```

常见问题

您可以在 API 概览 查看所有函数列表及其描述。
如果您的接入和使用中遇到问题，请参见 常见问题 。

> **说明：**
> 如果您在 iOS 运行 release 版本时遇到找不到符号的问题，请参见 iOS release 包运行时 [symbol not found] 。

联系我们

如果您是开发者，欢迎您加入我们的 RTC Engine 技术交流平台 zhiliao ，进行技术交流和产品沟通。

Electron

最近更新时间：2025-03-04 11:42:03

本文将介绍如何在 Electron 环境下，快速接入实时音视频 SDK，实现一个基本的音视频通话。

环境准备

- Windows 7+，推荐 Windows 10/11。
- MacOS 10.13+。
- Node.js 16.20.2 或更高稳定版。

接入指引

步骤1. 导入 SDK

1. 在命令行窗口中执行如下命令，安装 Electron，建议版本号 >= 4.0.0。

```
npm install electron@24.8.8 --save-dev
```

> **注意：**
> Mac 平台下，Electron 从 25.0.0 版本开始，ElectronSDK 的事件通知会失效，请使用 Electron 25.0.0 以下的版本。

2. 在您的 Electron 项目中使用 npm 命令安装 SDK 包：

```
npm install trtc-electron-sdk@latest --save
```

> **说明：**
> TRTC Electron SDK 最新版可在 trtc-electron-sdk 中查看。

步骤2. 配置工程

1. 为了正确打包运行 Electron 版本的 TRTC SDK（也就是 `trtc_electron_sdk.node` 文件），您还需要执行如下命令以安装 `native-ext-loader` 工具。

```
npm install native-ext-loader@latest --save-dev
```

2. 为了解决 webpack 无法处理 Node.js 的内置 'fs' 模块，我们需要执行以下命令安装 `path-browserify` 工具。

```
npm install path-browserify --save-dev
```

3. 以 webpack项目为例，在 `webpack.config.js` 包含了项目构建的配置信息，`webpack.config.js` 文件的位置如下：
   - 通常情况下，`webpack.config.js` 位于项目的根目录。
   - 使用 `create-react-app` 创建项目的情况下，此配置文件为 `node_modules/react-scripts/config/webpack.config.js`。
   - 使用 `vue-cli` 创建项目的情况下，webpack 的配置存放在 `vue.config.js` 配置中的 `configureWebpack` 属性中。
   - 如您的工程文件经过了定制化，还请自行查找 webpack 配置。

首先使 `webpack.config.js` 在构建时可以接收名为 `--target_platform` 的命令行参数，以使代码构建过程按不同的目标平台特点正确打包，在 `module.exports` 之前添加以下代码：

```
const os = require('os');
const targetPlatform = (function(){
    let target = os.platform();
    for (let i=0; i<process.argv.length; i++) {
        if (process.argv[i].includes('--target_platform=')) {
            target = process.argv[i].replace('--target_platform=', '');
            break;
        }
    }
    if (!['win32', 'darwin'].includes) target = os.platform();
    return target;
})();
```

> **注意：**
> `os.platform()` 返回的结果中，"darwin" 表示 Mac 平台。"win32" 表示 Windows 平台，不论 64 位还是 32 位。

然后在 `module` 选项中添加以下配置，`targetPlatform` 变量可以使 `rewritePath` 可以根据不同的目标平台切换不同的配置：

```
module: {
  rules: [
    {
        test: /\.node$/,
        loader: 'native-ext-loader',
        options: {
            rewritePath: targetPlatform === 'win32' ? './resources' : '../Resources'
            // 针对开发环境
            // rewritePath: './node_modules/trtc-electron-sdk/build/Release'
        }
    },
  ]
}
```

该配置的含义是：

- 打包 Windows 下的 `.exe` 文件时，让 `native-ext-loader` 到 `[应用程序根目录]/resources` 目录下加载 TRTC SDK。
- 打包 Mac 下的 `.dmg` 时，让 `native-ext-loader` 到 `[应用程序目录]/Contents/Frameworsk/../Resources` 目录下加载 TRTC SDK。
- 本地开发运行时，让 `native-ext-loader` 到 `./node_modules/trtc-electron-sdk/build/Release` 目录下加载 TRTC SDK，请参见 simple demo 配置 。

接着在 `resolve` 选项中添加以下配置，该配置用于处理 Node.js 核心模块在浏览器环境中的兼容性问题：

```
resolve: {
  fallback: {
    "fs": false,
    "path": require.resolve("path-browserify"),
  }
}
```

最后在 `externals` 中配置下列代码使得 webpack 允许加载本地模块：

```
externals: {
  "trtc-electron-sdk": "commonjs trtc-electron-sdk"
}
```

4. 设备权限申请示例代码：

```
const { systemPreferences } = require('electron');

async function checkAndApplyDeviceAccessPrivilege() {
  if (process.platform === "darwin" || process.platform === 'win32') {
      // 检查并申请摄像头权限
      const cameraPrivilege = systemPreferences.getMediaAccessStatus("camera");
      if (cameraPrivilege !== "granted") {
        await systemPreferences.askForMediaAccess("camera");
      }

      // 检查并申请麦克风权限
      const micPrivilege = systemPreferences.getMediaAccessStatus("microphone");
      if (micPrivilege !== "granted") {
        await systemPreferences.askForMediaAccess("microphone");
      }

      // 检查访问屏幕权限
      const screenPrivilege = systemPreferences.getMediaAccessStatus("screen");
      if (screenPrivilege !== 'granted') {
        // 没有屏幕访问权限，做后续处理...
      }
  }
}

async function createWindow(width = 1366, height = 668) {
  await checkAndApplyDeviceAccessPrivilege();
  // ... 其他窗口创建逻辑
}
```

步骤3. 创建 TRTC 实例

创建 TRTC 实例并设置监听，通过设置事件回调接口，您可以监听 SDK 在运行期间所发生的错误信息、警告信息、流量统计信息、网络质量信息以及各种音视频事件。

```
import TRTCCloud from 'trtc-electron-sdk';
const rtcCloud = TRTCCloud.getTRTCShareInstance();

function onError(errCode, errMsg) {
  // errorCode 可参考
https://cloud.tencent.com/document/product/647/32257#.E9.94.99.E8.AF.AF.E7.A0.81.E8.A1.A8
  console.log(errCode, errMsg);
}

function onWarning(warningCode, warningMsg) {
```

```
  // warningCode 可参考
https://cloud.tencent.com/document/product/647/32257#.E8.AD.A6.E5.91.8A.E7.A0.81.E8.A1.A8
  console.log(warningCode, warningMsg);
}

rtcCloud.on('onError', onError);
rtcCloud.on('onWarning', onWarning);
```

步骤4. 进入房间

在调用 `enterRoom` 接口进房时需要填写两个关键参数，即 `TRTCParams` 和 `TRTCAppScene` ，接下来进行详细介绍：

参数一：TRTCAppScene

该参数用于指定您的应用场景，即**在线直播**还是**实时通话**：

- 实时通话：包含 `TRTCAppSceneVideoCall` 和 `TRTCAppSceneAudioCal` 两个可选项，分别是视频通话和语音通话，该模式适合 1对1 的音视频通话，或者参会人数在 300 人以内的在线会议。
- 在线直播：包含 `TRTCAppSceneLIVE` 和 `TRTCAppSceneVoiceChatRoom` 两个可选项，分别是视频直播和语音直播，该模式适合十万人以内的在线直播场景，但需要您在接下来介绍的 `TRTCParams` 参数中指定 **角色(role)** 这个字段，也就是将房间中的用户区分为 **主播(anchor)** 和 **观众(audience)** 两种不同的角色。

参数二：TRTCParams

TRTCParams 由很多的字段构成，但通常您只需要关心如下几个字段的填写：

参数名称	字段含义	补充说明	数据类型	填写示例
SDKAppID	应用 ID	您可以在 实时音视频控制台 中找到这个 SDKAppID，如果没有就单击“创建应用”按钮创建一个新的应用。	数字	1400000123
userId	用户 ID	即用户名，只允许包含大小写英文字母（a-z、A-Z）、数字（0-9）及下划线和连词符。注意 TRTC 不支持同一个 userId 在两台不同的设备上同时进入房间，否则会相互干扰。	字符串	“denny” 或者 “123321”
userSig	进房鉴权票据	您可以使用 SDKAppID 和 userId 计算出 userSig，计算方法请参见 如何计算及使用 UserSig 。	字符串	eJyrVareCeYrSy1Ssll...
roomId	房间号	数字类型的房间号。注意如果您想使用字符串类型的房间号，请使用 strRoomId 字段，而不要使用 roomId 字段，因为 strRoomId 和 roomId 不可以混用。	数字	29834
strRoomId	房间号	字符串类型的房间号。注意 strRoomId 和 roomId 不可以混用，“123” 和 123 在 TRTC 后台服务看来并不是同一个房间。	字符串	“29834”
role	角色	分为“主播”和“观众”两种角色，只有当 TRTCAppScene 被指定为 TRTCAppSceneLIVE 或 TRTCAppSceneVoiceChatRoom 这两种直播场景时才需要指定该字段。	枚举值	TRTCRoleAnchor 或 TRTCRoleAudience

⚠ 注意：

- TRTC 不支持同一个 userId 在两台不同的设备上同时进入房间，否则会相互干扰。
- 每个端在应用场景 appScene 上必须要进行统一，否则会出现一些不可预料的问题。

在准备好 `TRTCAppScene` 和 `TRTCParams` 这两个参数后，就可以调用 `enterRoom` 接口函数进入房间了。

```
import { TRTCParams, TRTCRoleType, TRTCAppScene } from 'trtc-electron-sdk';

const param = new TRTCParams();
param.sdkAppId = 1400000123;
param.userId = "denny";
param.roomId = 123321;
param.userSig = "xxx";
param.role = TRTCRoleType.TRTCRoleAnchor;

// 如果您的场景是“在线直播”，请将应用场景设置为 TRTC_APP_SCENE_LIVE
rtcCloud.enterRoom(param, TRTCAppScene.TRTCAppSceneLIVE);
```

调用接口后，您会收到来自 TRTCCallback 中的 onEnterRoom(result) 回调:

- 如果进房成功，result 会是一个正数（result > 0），表示加入房间的时间消耗，单位是毫秒（ms）。
- 如果进房失败，result 会是一个负数（result < 0），表示进房失败的错误码。

```
function onEnterRoom(result) {
  // onEnterRoom 参见 https://web.sdk.qcloud.com/trtc/electron/doc/zh-
cn/trtc_electron_sdk/TRTCCallback.html#event:onEnterRoom
  if (result > 0) {
    console.log('Enter room succeed');
  } else {
    // 参见进房错误码
https://cloud.tencent.com/document/product/647/32257#.E8.BF.9B.E6.88.BF.E7.9B.B8.E5.85.B3.E9
.94.99.E8.AF.AF.E7.A0.81
    console.log('Enter room failed');
  }

}

rtcCloud.on('onEnterRoom', onEnterRoom);
```

步骤5. 打开/关闭摄像头

`startLocalPreview(views)` 接口启动本地摄像头采集和预览，这个接口会启动默认的摄像头，可以通过 setCurrentCameraDevice() 接口选用其它摄像头 当开始渲染首帧摄像头画面时，您会收到 TRTCCallback 中的 onFirstVideoFrame('') 回调。

Name	Type	Description
`views`	Array.<HTMLElement> | HTMLElement | null	`required` 承载预览画面单个的 HTML 块节点或者 HTML 块节点数组 • 如果传入 null 则不预览摄像头画面，但不影响摄像头采集

```
// 要预览摄像头图像，您需要在 DOM 中放置一个 HTML 元素，
// 这可以是一个 div 标签，假设其 ID 为 local-video。
const view = document.getElementById('local-video');
rtcCloud.startLocalPreview(view);
```

`stopLocalPreview()` 接口停止本地摄像头采集和预览。

```
rtcCloud.stopLocalPreview();
```

步骤6. 打开/关闭麦克风

`startLocalAudio(quality)` 开启麦克风采集，请根据您的需求选择以下其中一个声音质量参数 `quality` 。

```
const { TRTCAudioQuality } = require('trtc-electron-sdk');
// 开启麦克风采集，设置当前场景为：标准模式（默认模式）
rtcCloud.startLocalAudio(TRTCAudioQuality.TRTCAudioQualityDefault);
```

```
// 开启麦克风采集，设置当前场景为：语音模式
// 具有高的噪声抑制能力，有强有弱的网络阻力
rtcCloud.startLocalAudio(TRTCAudioQuality.TRTCAudioQualitySpeech);
```

```
// 开启麦克风采集，设置当前场景为：音乐模式
// 为获得高保真度，低音质损失，建议配合专业声卡使用
rtcCloud.startLocalAudio(TRTCAudioQuality.TRTCAudioQualityMusic);
```

调用 `stopLocalAudio()` 关闭麦克风采集并停止推送本地音频信息。

```
rtcCloud.stopLocalAudio()
```

步骤7. 播放/停止视频流

1. 当您收到 onUserVideoAvailable(userId, 1) 通知时，代表该路画面已经有可用的视频数据帧到达。
2. 您需要调用 startRemoteView(userId, views, streamType) 接口加载该用户的远程画面。之后您会收到名为 onFirstVideoFrame(userId) 的首帧画面渲染回调。

Parameters:

Name	Type	Description
`userId`	String	`required` 对方的用户标识
`views`	Array.<HTMLElement> | HTMLElement | null	`required` 承载预览画面单个的 HTML 块节点或者 HTML 块节点数组 • 如果传入 null 则不预览远端用户画面，但不影响拉取远端视频，适合需要自定义渲染的场景
`streamType`	TRTCVideoStreamType	`required` 视频流类型： • TRTCVideoStreamTypeBig 大画面视频流 • TRTCVideoStreamTypeSmall 小画面视频流 • TRTCVideoStreamTypeSub 辅流（屏幕分享）

```
// 要预览摄像头图像，您需要在 DOM 中放置一个 HTML 元素，
// 这可以是一个 div 标签，假设其 ID 为 local-video。
```

```
const view = document.getElementById('local-video');
rtcCloud.startRemoteView("denny", view, TRTCVideoStreamType.TRTCVideoStreamTypeBig);
```

> ⚠ **注意：**
> - 如果主播的上行网络和性能比较好，则可以同时送出大小两路画面。
> - SDK 不支持单独开启小画面，小画面必须依附于主画面而存在。

调用 `stopRemoteView(userId, streamType)` 停止播放远端画面。

```
// 停止播放
rtcCloud.stopRemoteView("denny", TRTCVideoStreamType.TRTCVideoStreamTypeBig); // 只停止播放
denny 画面
rtcCloud.stopAllRemoteView(); // 停止播放所有远端画面
```

> ⚠ **注意：**
> 如果有屏幕分享的画面在显示，则屏幕分享的画面也会一并被关闭。

步骤8. 播放/停止音频流

默认情况下，SDK 将自动播放远程音频，因此您不需要调用任何 API 来手动播放它。
但当您不喜欢自动播放音频时，可以调用 `muteRemoteAudio` 选择播放或停止远端声音。

```
// 静音
rtcCloud.muteRemoteAudio("denny", true); // 仅静音 denny
rtcCloud.muteAllRemoteAudio(true); // 静音所有远端用户
```

```
// 取消静音
rtcCloud.muteRemoteAudio("denny", false); // 仅取消静音 denny
rtcCloud.muteAllRemoteAudio(false); // 取消静音所有远端用户
```

步骤9. 退出房间

调用 `exitRoom` 退出当前的房间，SDK 会通过 `TRTCCallback` 中的 `onExitRoom()` 回调通知您。如果您要再次调用 `enterRoom()` 或者切换到其它的音视频 SDK，请等待 `onExitRoom()` 回调到来后再执行相关操作，否则可能会遇到如摄像头、麦克风设备被强占等各种异常问题。

```
const onExitRoom = (reason) => {
  if (reason == 0) {
      console.log( "Exit current room by calling the 'exitRoom' api of sdk ...");
  } else if (reason == 1) {
      console.log("Kicked out of the current room by server through the restful api...");
  } else if (reason == 2) {
      console.log("Current room is dissolved by server through the restful api...");
  }
}

// 监听 `onExitRoom` 回调
rtcCloud.on('onExitRoom', onExitRoom);
```

步骤10. 应用打包

1. 打包可执行程序

安装打包工具：推荐使用打包工具 `electron-builder` 进行打包，您可以执行如下命令安装 `electron-builder` 。

```
npm install electron-builder@latest --save-dev
```

2. 修改 package.json 配置

`package.json` 位于项目的根目录，其中包含了项目打包所必须的信息。但默认情况下， `package.json` 中的路径是需要修改才能顺利实现打包的，我们可以按如下步骤修改此文件：

1. 修改 `main` 配置。

```
// 多数情况下，main 文件名称可以任意配置，例如 TRTCSimpleDemo 中的可以配置为：
"main": "main.electron.js",

// 但是，使用 create-react-app 脚手架创建的项目，main 文件必须配置为：
"main": "public/electron.js",
```

2. 复制以下 `build` 配置，添加到您的 `package.json` 文件中，这是 `electron-builder` 需要读取到的配置信息。Mac 下因合规和设备原因，不同 SDK 版本需要使用不同的配置。

10.3 以下版本

```
"build": {
  "appId": "[appId 请自行定义]",
  "directories": {
    "output": "./bin"
  },
  "asar": true,
  "win": {
    "extraFiles": [
      {
          "from": "node_modules/trtc-electron-sdk/build/Release/",
          "to": "./resources",
          "filter": ["**/*"]
        }
      ]
  },
  "mac": {
     "extraFiles": [
        {
          "from": "node_modules/trtc-electron-sdk/build/Release/trtc_electron_sdk.node",
          "to": "./Resources"
        }
      ]
   }
```

```
}
```

10.3 版本

从 10.3 版本开始，Mac 因合规要求，拆解出了运行时的两个动态库，安装 SDK 后位于 node_modules/trtc-electron-sdk/build/mac-framework/ 目录下，构建应用安装包时，需要在 build.mac.extraFiles 下配置这两个动态库，否则会导致应用包运行时崩溃。

```
"build": {
  "appId": "[appId 请自行定义]",
  "directories": {
    "output": "./bin"
  },
  "asar": true,
  "win": {
    "extraFiles": [
      {
          "from": "node_modules/trtc-electron-sdk/build/Release/",
          "to": "./resources",
          "filter": ["**/*"]
        }
      ]
  },
  "mac": {
     "extraFiles": [
        {
          "from": "node_modules/trtc-electron-sdk/build/Release/trtc_electron_sdk.node",
          "to": "./Resources"
        },
        {
          "from": "node_modules/trtc-electron-sdk/build/mac-framework/",
          "to": "./Frameworks"
        }
      ]
   }
}
```

10.3 以上版本

从 10.6 版本开始，Mac 下支持原生 ARM64 指令架构的包，同时提供 X64 和 ARM64 指令架构的动态库文件，支持使用 electron-builder 的 “--universal” 参数，构建双指令架构的应用包，优点是可同时在 X64 和 ARM64 芯片的设备上提供优良的性能体验，缺点是应用的包体积大约会增加 70%。不使用 “--universal” 参数时，构建的是单指令架构的应用包，默认与构建时使用的计算机 CPU 芯片类型一致，此时应用安装包在另一种 CPU 芯片类型的机器上将无法安装。
注意：electron-builder 需要升级到 23.0.2 或更高版本。

```
"build": {
  "appId": "[appId 请自行定义]",
```

```
  "directories": {
    "output": "./bin"
  },
  "asar": true,
  "asarUnpack": "**\\*.{node,dll}",
  "win": {
    "extraFiles": [
      {
          "from": "node_modules/trtc-electron-sdk/build/Release/",
          "to": "./resources",
          "filter": ["**/*"]
        }
      ]
  },
  "mac": {
     "extraFiles": [
        {
          "from": "node_modules/trtc-electron-
sdk/build/Release/${arch}/trtc_electron_sdk.node",
          "to": "./Resources"
        },
        {
          "from": "node_modules/trtc-electron-sdk/build/mac-framework/${arch}/",
          "to": "./Frameworks"
        }
      ]
   }
}
```

3. 在 `scripts` 节点下添加以下构建和打包的命令脚本：

4. 本文以 `create-react-app` 和 `vue-cli` 项目为例，其它工具创建的项目也可以参考此配置：

```
// create-react-app 项目请使用此配置
"scripts": {
  "build:mac": "react-scripts build --target_platform=darwin",
  "build:win": "react-scripts build --target_platform=win32",
  "compile:mac": "electron-builder --mac",
  "compile:mac-universal": "electron-builder --mac --universal",                // 10.3 及
以下版本不支持
  "compile:win64": "electron-builder --win --x64",
  "pack:mac": "npm run build:mac && npm run compile:mac",
  "pack:mac-universal": "npm run build:mac && npm run compile:mac-universal",    // 10.3 及
以下版本不支持
  "pack:win64": "npm run build:win && npm run compile:win64"
}

// vue-cli 项目请使用此配置
"scripts": {
  "build:mac": "vue-cli-service build --target_platform=darwin",
  "build:win": "vue-cli-service build --target_platform=win32",
  "compile:mac": "electron-builder --mac",
```

```
  "compile:mac-universal": "electron-builder --mac --universal",                // 10.3 及
以下版本不支持
  "compile:win64": "electron-builder --win --x64",
  "pack:mac": "npm run build:mac && npm run compile:mac",
  "pack:mac-universal": "npm run build:mac && npm run compile:mac-universal",    // 10.3 及
以下版本不支持
  "pack:win64": "npm run build:win && npm run compile:win64"
}
```

参数	说明
main	Electron 的入口文件，一般情况下可以自由配置。但如果项目使用 create-react-app 脚手架创建，则入口文件必须配置为 public/electron.js
build.asar	是否开启 Electron 应用安装包的压缩功能，默认开启。
build.asarUnpack	添加 build.asar 配置的例外文件。10.3 以后的版本，使用 `electron-builder --mac --universal` 命令构建双指令架构的应用包时，需要此配置，否则可能出现文件合并的失败或报错。
build.win.extraFiles	打包 Windows 程序时，electron-builder 会把 from 所指目录下的所有文件复制到 bin/win-unpacked/resources（全小写）
build.mac.extraFiles	打包 Mac 程序时，electron-builder 会把 from 指向的 trtc_electron_sdk.node 文件复制到 bin/mac/your-app-name.app/Contents/Resources（首字母大写）
build.directories.output	打包文件的输出路径。例如这个配置会输出到 bin 目录下，可根据实际需要修改
build.scripts.build:mac	以 Mac 平台为目标构建脚本
build.scripts.build:win	以 Windows 平台为目标构建脚本
build.scripts.compile:mac	编译为 Mac 下的 .dmg 安装文件
build.scripts.compile:win64	编译为 Windows 下的 .exe 安装文件
build.scripts.pack:mac	先调用 build:mac 构建代码，再调用 compile:mac 打包成 .dmg 安装文件
build.scripts.pack:win64	先调用 build:win 构建代码，再调用 compile:win64 打包成 .exe 安装文件

3. 执行打包命令

- **打包 Mac.dmg 安装文件:**

```
cd [项目目录]
npm run pack:mac
// or
npm run pack:mac-universal
```

成功执行后，打包工具会生成 `bin/your-app-name-0.1.0.dmg` 安装文件，请选择此文件发布。

- **打包 Windows.exe 安装文件：**

```
cd [项目目录]
npm run pack:win64
```

成功执行后，打包工具会生成 `bin/your-app-name Setup 0.1.0.exe` 安装文件，请选择此文件发布。

> **注意**
> TRTC Electron SDK 暂不支持跨平台打包（例如在 Mac 下打包 Windows 的 .exe 文件，或在 Windows 平台下打包 Mac 的 .dmg 文件）。目前我们正在研究跨平台打包方案，敬请期待。

常见问题

- 您可以在 API 参考 查看所有函数列表及其描述。
- 如果您的接入和使用中遇到问题，请参见 常见问题 。

联系我们

如果您是开发者，欢迎您加入我们的 RTC Engine 技术交流平台 zhiliao，进行技术交流和产品沟通。

uni-app

最近更新时间：2025-03-04 11:42:03

本文将介绍如何快速完成 TRTC SDK 的接入，实现一个基本的音视频通话。

环境准备

- 建议使用最新的 HBuilderX 编辑器 。
- iOS 9.0 或以上版本且支持音视频的 iOS 设备，暂不支持模拟器。
- Android 版本不低于 4.1 且支持音视频的 Android 设备，暂不支持模拟器；如果为真机，请开启“允许调试”选项。
- iOS/Android 设备已经连接到 Internet。

步骤一：导入实时音视频 SDK 插件

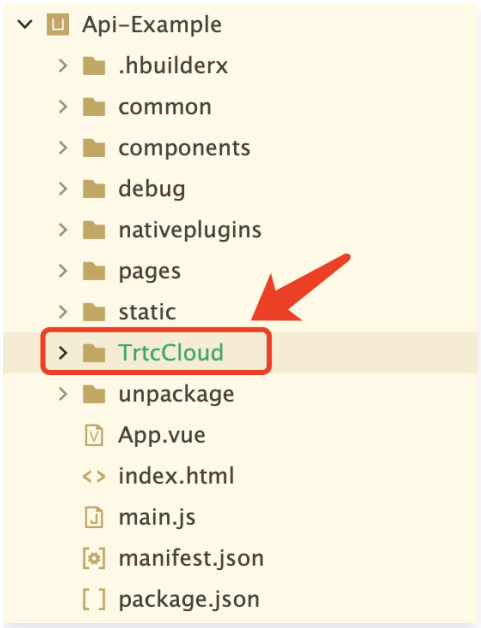
1. 下载并导入【官方】腾讯云实时音视频 SDK 原生插件到本地项目。

2. 制作自定义调试基座，请选择**传统打包**方式进行打包。

3. 自定义调试基座成功后，**使用自定义基座运行**项目。

步骤二：导入 TrtcCloud

下载 TrtcCloud Zip ， 解压后放到本地 uni-app 的项目中，如下图所示：



步骤三：创建 TRTC 实例

1. 导入 TrtcCloud 到项目中。

```
import TrtcCloud from '@/TrtcCloud/lib/index';
```

2. 用 TrtcCloud.createInstance API 方法来创建实例。

```
this.trtcCloud = TrtcCloud.createInstance();
```

步骤四：进入通话房间

调用 trtcCloud.enterRoom API 进入房间。 此方法通常在“开始通话”按钮的点击回调函数中调用。

参数	类型	描述
sdkAppId	number	您在 TRTC 控制台 中创建的音视频应用程序的 sdkAppId。
userId	string	您指定的用户 ID。
userSig	string	用户签名，请参考 UserSig 。
roomId	number	您指定的房间 ID，通常是一个唯一的房间 ID。

```
import { TRTCAppScene } from '@/TrtcCloud/lib/TrtcDefines';

const params = {
  sdkAppId: 0,
  userId: 'xxx',
  roomId: 12345,
  userSig: 'xxx'
};
```

```
this.trtcCloud.enterRoom(params, TRTCAppScene.TRTCAppSceneVideoCall);
```

步骤五：打开/关闭摄像头

调用 trtcCloud.startLocalPreview API 打开摄像头并将视频发布到房间。

```
// 打开摄像头并将视频发布到房间
const viewId = 'local-video'; // 预览摄像头画面需要一个 HTML 元素作为容器；如一个 div 标签，id 为
local-video。
this.trtcCloud.startLocalPreview(true, viewId);

// 关闭摄像头并取消发布
this.trtcCloud.stopLocalPreview();
```

步骤六：打开/关闭麦克风

调用 trtcCloud.startLocalAudio API 打开麦克风并将音频发布到房间。

```
// 打开麦克风并将音频发布到房间
this.trtcCloud.startLocalAudio();

// 关闭麦克风并取消发布
this.trtcCloud.stopLocalAudio();
```

步骤七：播放/停止远端视频流

1. 在进入房间之前，监听 onUserVideoAvailable 事件，以接收所有远端用户视频发布事件。
2. 当您收到该事件时，调用 trtcCloud.startRemoteView API 来播放远端视频流。

```
// 播放远端画面需要一个 HTML 元素作为容器；如一个 div 标签，id 为 remoteUserId。
this.trtcCloud.on('onUserVideoAvailable', (res) => {
  const { userId, available } = res;
  if (userId && available) {
    this.trtcCloud.startRemoteView(this.remoteUserId, this.streamType, this.remoteUserId);
  }
});
```

步骤八：播放/停止远端音频流

- 默认情况下，SDK 会自动播放远端音频，因此您无需手动调用任何 API 来播放远端音频。
- 以下代码片段介绍如何关闭自动播放音频，并手动控制远端音频播放。

```
this.trtcCloud.on('onUserAudioAvailable', (res) => {
  const { userId, available } = res;
  // 停止远端音频
  this.trtcCloud.muteRemoteAudio(userId, true);
  // 播放远端音频
  this.trtcCloud.muteRemoteAudio(userId, false);
});

// 设置 autoReceiveAudio = false 以关闭自动音频播放。
```

```
await trtcCloud.enterRoom({ ..., autoReceiveAudio: false });
```

步骤九：退出房间

调用 trtcCloud.exitRoom API 来退出房间。

```
this.trtcCloud.exitRoom();
```

技术咨询

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

React Native

最近更新时间：2025-04-10 14:55:12

环境准备

1. React Native 0.60+
2. Android 5.0+（真机调试）
3. iOS 12.0+（真机调试）
4. Node.js 14+

关于搭建开发环境的更多注意事项，详见 搭建开发环境 。

步骤1. 导入SDK

通过 npm 或 yarn 安装 SDK：

```
npm install trtc-react-native --save
# 或
yarn add trtc-react-native
```

步骤2. 配置项目

Android

1. 打开 `/android/app/src/main/AndroidManifest.xml` 文件，在其中添加如下权限：

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-feature android:name="android.hardware.camera.autofocus" />
```

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

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

iOS

1. iOS需要在 `Info.plist` 的第一级<dict>目录下加入对相机和麦克风的权限申请：

```
<key>NSCameraUsageDescription</key>
<string>授权摄像头权限才能正常视频通话</string>
<key>NSMicrophoneUsageDescription</key>
<string>授权麦克风权限才能正常语音通话</string>
```

2. 执行 `pod install` 安装原生依赖（需在 `ios` 目录下执行）。

> **说明：**
> 如果您在接入过程中遇到问题，请参见 示例项目 。

步骤3. 创建 TRTC 实例

```
import TRTCCloud from 'trtc-react-native';

// 创建 TRTC 单例实例
const trtcCloud = TRTCCloud.sharedInstance();
```

步骤4. 进入房间

```
import { TRTCParams, TRTCCloudDef } from 'trtc-react-native';

const params = new TRTCParams({
  sdkAppId: SDKAPPID, // 腾讯云控制台申请的 SDKAppID
  userId: 'user123',    // 用户唯一标识
  userSig: 'xxxxxx',    // 用户签名（需通过服务端生成）
  roomId: 2366,         // 房间号
});

// 进入音视频通话场景的房间
trtcCloud.enterRoom(params, TRTCCloudDef.TRTC_APP_SCENE_VIDEOCALL);
```

> **说明：**
> 您可以在 TRTC 控制台 中获取到您应用的 sdkAppId，并且参见 用户鉴权 获取 userSig。

步骤5. 打开摄像头

```
// 启动本地视频预览
trtcCloud.startLocalPreview(true); // `true` 表示开启前置摄像头

// 设置视频编码参数（可选）
const config = {
  videoResolution: TRTCCloudDef.TRTC_VIDEO_RESOLUTION_640_360,
  videoFps: 15,
  videoBitrate: 800,
};
trtcCloud.setVideoEncoderParam(config);
```

步骤6. 打开麦克风

```
trtcCloud.startLocalAudio(); // 开启本地音频采集和上行
```

步骤7. 播放/停止视频流

播放本地视频：

```
import { TXVideoView } from 'trtc-react-native';

// 在组件中渲染本地视频
<TXVideoView.LocalView style={{ width: 1080, height: 1080 }} />
```

播放远程视频：

```
<TXVideoView.RemoteView
  userId={remoteUserId}
  streamType={TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG}
  style={{ width: 1080, height: 1080 }}
/>
```

停止播放：

```
trtcCloud.stopRemoteView(userId, TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG);
```

步骤8. 播放/停止音频流

播放远程音频：

```
trtcCloud.muteRemoteAudio(userId, false); // `false` 表示取消静音
```

停止播放：

```
trtcCloud.muteRemoteAudio(userId, true);  // `true` 表示静音
```

步骤9. 退出房间

```
trtcCloud.exitRoom(); // 退出当前房间
```

常见问题

出现黑屏问题，如何处理？

- 检查摄像头权限是否已授权。
- 确保 `TXVideoView` 组件已正确渲染且尺寸不为零。

出现无法进入房间，如何处理？

- 检查 `sdkAppId` 、`userSig` 和 `roomId` 是否有效。
- 确保网络连接正常。

出现音频异常，如何处理？

- 检查麦克风权限是否已授权。
- 调用 `startLocalAudio()` 后确认麦克风状态。

联系我们

如果您是开发者，欢迎您加入我们的 RTC Engine 技术交流平台 zhiliao，进行技术交流和产品沟通。

版权所有：腾讯云计算（北京）有限责任公司

Windows C++

最近更新时间：2025-03-04 11:42:03

本文将介绍如何快速完成 Windows RTC Engine 的接入，实现一个基本的音视频通话。

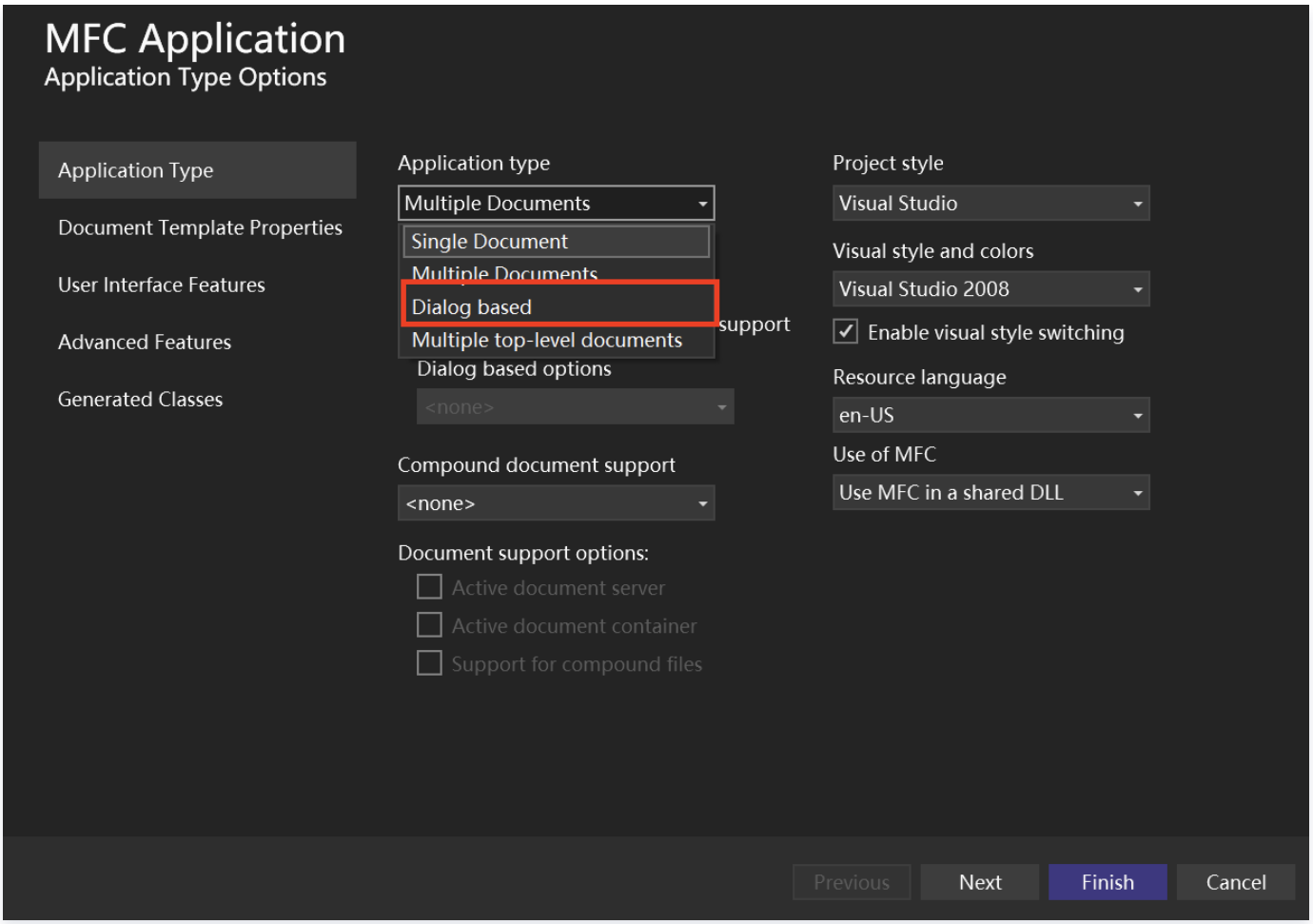
环境准备

- 操作系统：Windows 7及以上版本。
- 开发环境：Visual Studio 2019及以上版本，推荐使用 Visual Studio 2022。

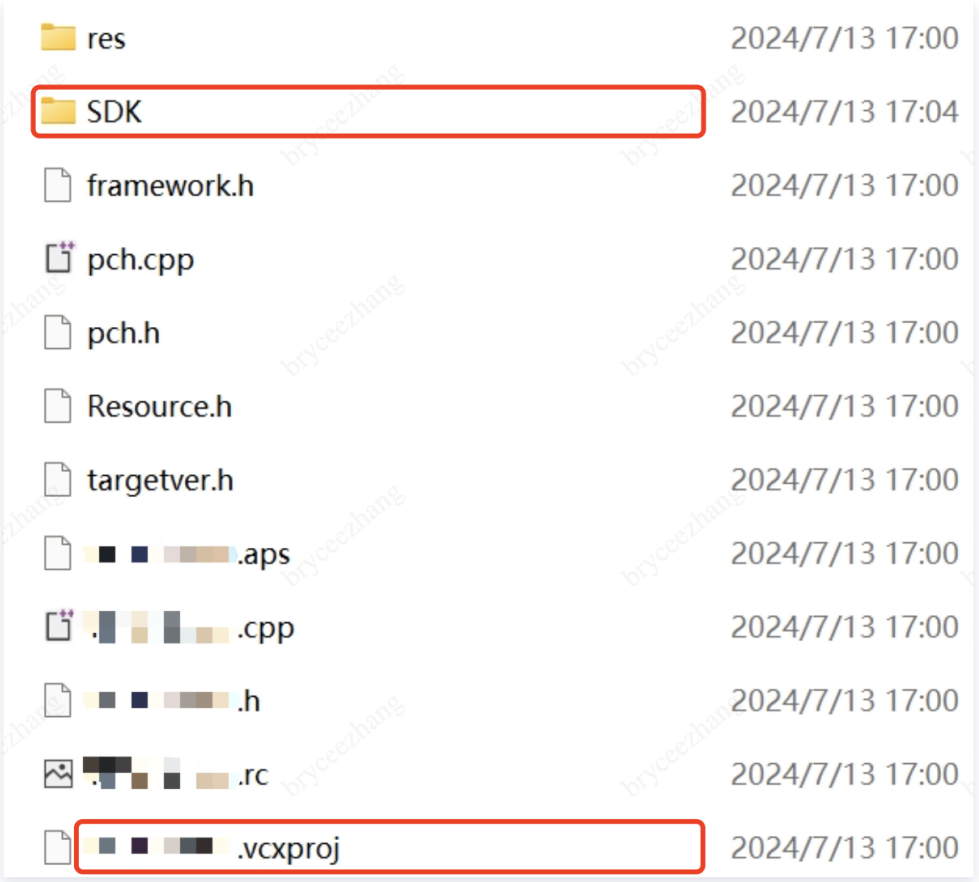
接入指引

步骤1. 导入 SDK

1. 打开 Visual Studio，新建一个您自己的 MFC 应用程序。在向导的 MFC Application 页面，Application type 选择 Dialog based，其他向导配置选择默认即可，单击 Finish。



2. 下载 Windows SDK，将解压后的 SDK 文件拷贝到 `.vcxproj` 文件所在目录下。

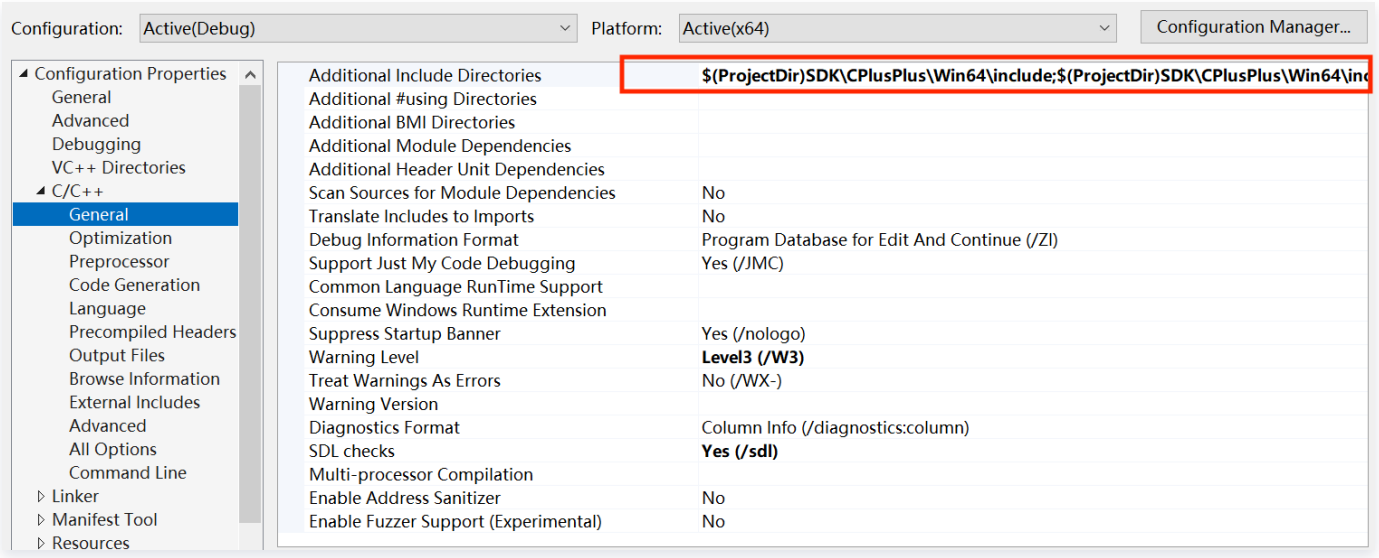


步骤2. 配置项目

双击打开 .sln 工程文件，在右侧工具栏 Solution Explorer > 右键打开项目的选择菜单 > Properties，进行以下配置：

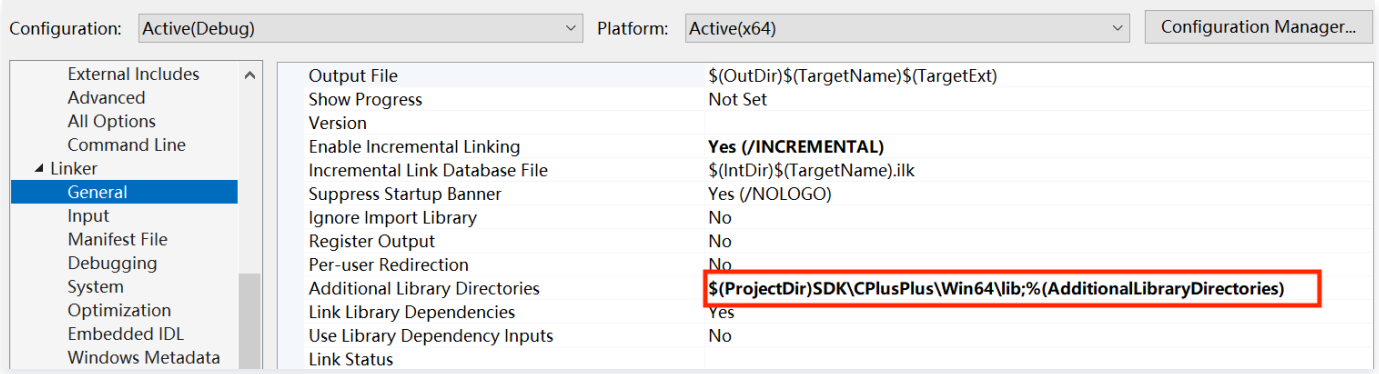
1. 添加 Additional Include Directories：

   在 C/C++ > General > Additional Include Directories，添加 SDK 头文件目录：
   `$(ProjectDir)SDK\CPlusPlus\Win64\include` 和 `$(ProjectDir)SDK\CPlusPlus\Win64\include\TRTC` 。



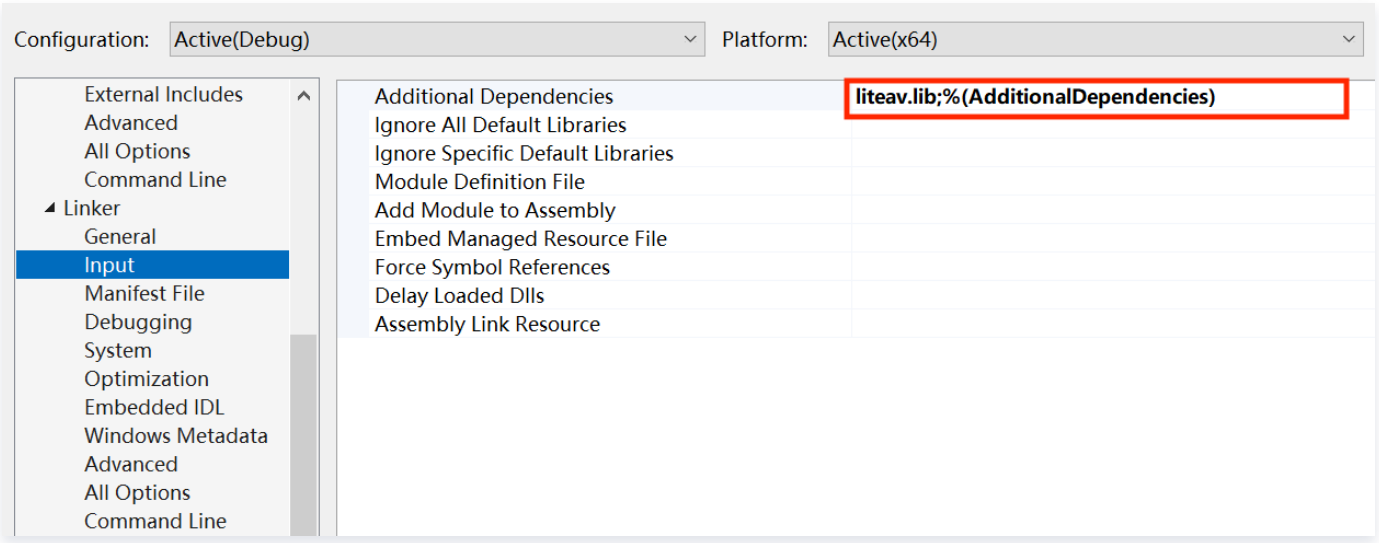
2. 添加 Additional Library Directories：

   在 Linker > General > Additional Library Directories，添加 SDK 库目录 `$(ProjectDir)SDK\CPlusPlus\Win64\lib` 。



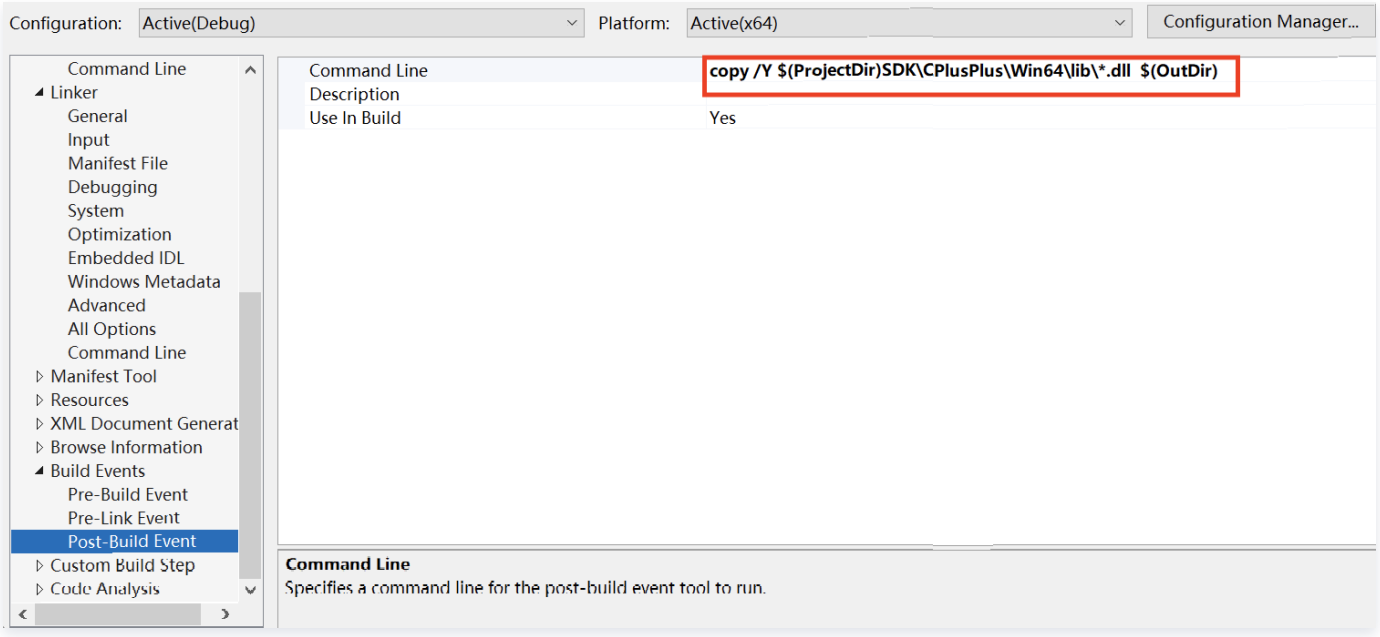
3. **添加 Additional Dependencies：**

   在 Linker > Input > Additional Dependencies，添加 SDK 库文件 `liteav.lib` **。**



4. **添加 Command line：**

   在 Build Events > Post-Build Event > Command line，添加
   `copy /Y $(ProjectDir)SDK\CPlusPlus\Win64\lib\*.dll  $(OutDir)` **。**



5. **打印 SDK 版本号：**

   5.1 双击打开 Solution Explorer 中的 .cpp 文件并引入 TRTC 头文件：

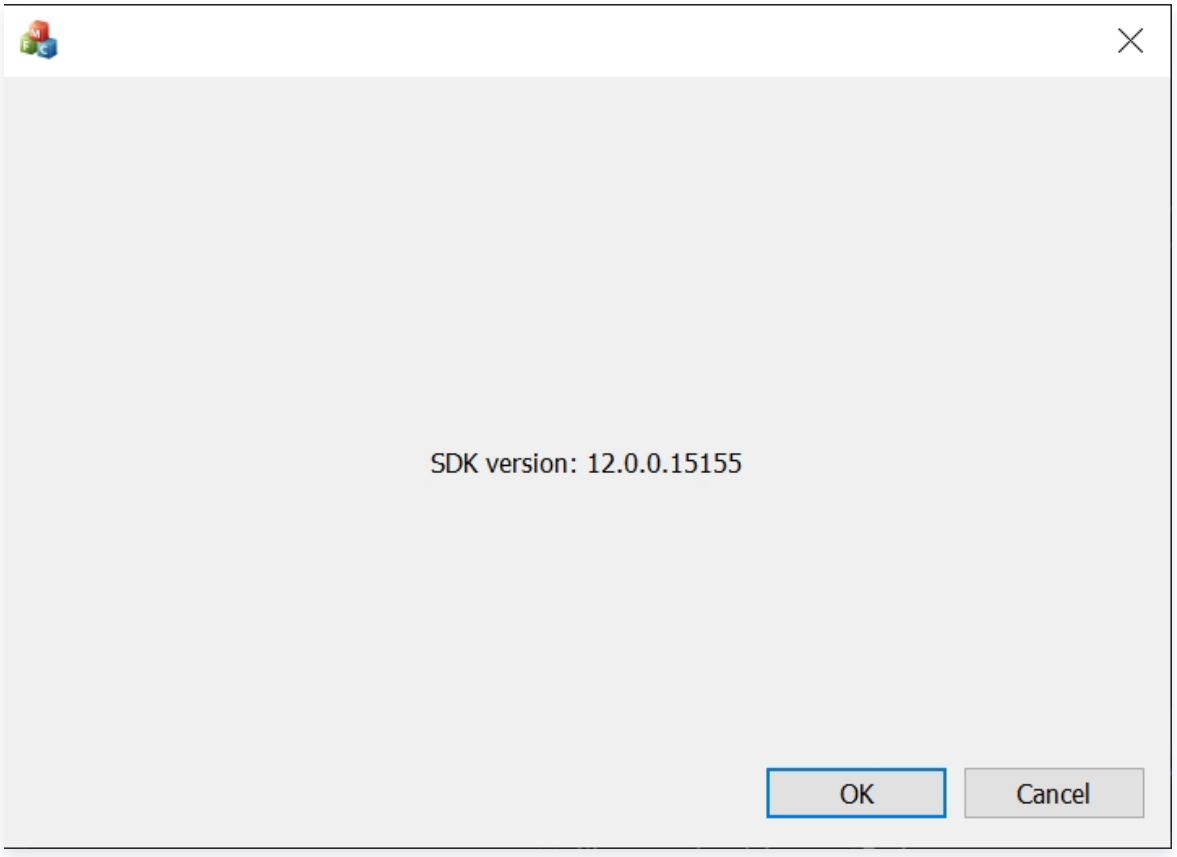
```
#include "ITRTCCloud.h"
```

> **注意：**
> 在已有头文件的后面引用 `"ITRTCCloud.h"`，否则将会出现报错： `ITRTCCloud：未定义标识符` 。

5.2 在 `OnInitDialog` 函数中，添加以下代码：

```
ITRTCCloud * pTRTCCloud = getTRTCShareInstance();
CString szText;
szText.Format(L"SDK version: %hs", pTRTCCloud->getSDKVersion());

CWnd *pStatic = GetDlgItem(IDC_STATIC);
pStatic->SetWindowTextW(szText);
```

5.3 完成以上步骤，单击 OK 成功打印 SDK 版本号即可。



> **说明：**
> 如果出现报错信息：**module machine type 'x86' conflicts with target machine type 'x64'**，在解决方案平台选择 **'x64'**。

步骤3. 创建 TRTC 实例

1. 在 **项目名.h** 文件中引用头文件 `"ITRTCCloud.h"`，将该文件中的类公有继承于 **CWinApp** 和 **ITRTCCloudCallback**，并声明监听事件和 **TRTCCloud** 成员变量。

```
#include "ITRTCCloud.h" // 引入 TRTC 头文件
```

```
class CRTCWindowsApp: public CWinApp, public ITRTCCloudCallback
{
public:
    CRTCWindowsApp();

// Overrides
public:
    virtual BOOL InitInstance();
    virtual void onError(TXLiteAVError errCode, const char* errMsg, void* extraInfo)
override; // 监听 'onError' 事件
    virtual void onWarning(TXLiteAVWarning warningCode, const char* warningMsg, void*
extraInfo) override; // 监听 'onWarning' 事件
    virtual void onEnterRoom(int result) override; // 监听 'onEnterRoom' 事件
    virtual void onExitRoom(int reason) override; // 监听 'onExitRoom' 事件

    // 获取 TRTCCloud 实例
    static CRTCWindowsApp* GetInstance()
    {
        return static_cast<CRTCWindowsApp*>(AfxGetApp());
    }

// Implementation
    DECLARE_MESSAGE_MAP()

public:
    ITRTCCloud* trtc_cloud_; // 声明 TRTCCloud 成员变量
};
```

注意：
本文所创建的工程项目名为 **RTCWindows**，在整个接入的过程中，请替换为您自己项目的名称或类名。

2. 在 **项目名.cpp** 文件的 `InitInstance` 方法中调用 `getTRTCShareInstance` 初始化 TRTC 对象实例。

```
// 初始化 TRTCCloud 对象并添加事件监听
trtc_cloud_ = getTRTCShareInstance();
trtc_cloud_->addCallback(this);
```

说明：
建议将初始化 TRTC 对象实例的代码添加在 `SetRegistryKey(_T("Local AppWizard-Generated Applications"))` 方法后。

3. 实现声明的监听事件。

```
// 对监听事件进行处理
void CRTCWindowsApp::onError(TXLiteAVError errCode, const char* errMsg, void* extraInfo) {
    // 监听 "onError" 事件并对相应信息进行打印
}
```

```
void CRTCWindowsApp::onWarning(TXLiteAVWarning warningCode, const char* warningMsg, void*
extraInfo) {
    // 监听 "onWarning" 事件并对相应信息进行打印
}

void CRTCWindowsApp::onEnterRoom(int result) { // 监听 "onEnterRoom" 事件并对相应信息进行打印
    if (result > 0) {
        MessageBox(NULL, TEXT("Enter room successfully"), TEXT("Notice"), MB_OK |
MB_ICONINFORMATION);
    }
    else {
        MessageBox(NULL, TEXT("Enter room failed"), TEXT("Notice"), MB_OK |
MB_ICONINFORMATION);
    }
}

void CRTCWindowsApp::onExitRoom(int reason) { // 监听 "onExitRoom" 事件并对相应信息进行打印
    if (reason == 0) {
        MessageBox(NULL, TEXT("Exit current room by calling the 'exitRoom' api of sdk"),
TEXT("Notice"), MB_OK | MB_ICONINFORMATION);
    } else if (reason == 1) {
        MessageBox(NULL, TEXT("Kicked out of the current room by server through the
restful api"), TEXT("Notice"), MB_OK | MB_ICONINFORMATION);
    } else if (reason == 2) {
        MessageBox(NULL, TEXT("The current room is dissolved by server through the restful
api"), TEXT("Notice"), MB_OK | MB_ICONINFORMATION);
    }
}
```

步骤4. 进入房间

设置进房参数 `TRTCParams` 并调用 `enterRoom` 即可成功进入房间，该方法通常在点击**开始通话按钮**后调用。

参数	类型	描述
sdkAppId	number	您在 TRTC 控制台 中创建的音频和视频应用程序的sdkAppId。
userId	string	您指定的用户ID。
userSig	string	用户签名，请参见 UserSig 。
roomId	number	您指定的房间ID，通常是唯一的房间ID。

有关更详细的参数描述，请参考接口文档 enterRoom 。

```
void CRTCWindowsApp::OnBnClickedButton() {
    // 将以下 trtcParams 参数修改为自己的参数
    liteav::TRTCParams trtcParams;
    trtcParams.sdkAppId = 1400000123;
    trtcParams.userId = "denny";
    trtcParams.roomId = 123321;
    trtcParams.userSig = "xxx";

    // 对于多人视频通话场景，推荐使用 TRTC_APP_SCENE_LIVE
```

```
    ITRTCCloud* trtcCloud = CRTCWindowsApp::GetInstance()->trtc_cloud_;
    trtcCloud->enterRoom(trtcParams, liteav::TRTCAppSceneLIVE);
}
```

步骤5. 打开/关闭摄像头

1. 在资源文件 IDD_TRTCDEMO_DIALOG 中，单击左侧边框的 Toolbox，拖动 Picture Control 到对话框中。
2. 在 Picture Control 控件的右键菜单中选择 Properties，ID 选择 AFX_IDC_PICTURE。
3. 在资源文件 IDD_TRTCDEMO_DIALOG 中添加一个 Button 组件，双击新建的 Button 后对 `pLocalVideoView` 进行初始化，设置本地预览的渲染参数 `setLocalRenderParams` ，并调用 `startLocalPreview` 进行本地预览，成功调用 `enterRoom` 后开始推流。

```
void CRTCWindowsApp::OnBnClickedButton()
{
    // 设置本地预览渲染参数
    liteav::TRTCRenderParams render_params;
    render_params.mirrorType = liteav::TRTCVideoMirrorType_Enable;
    render_params.fillMode = TRTCVideoFillMode_Fill;
    ITRTCCloud* trtcCloud = CRTCWindowsApp::GetInstance()->trtc_cloud_;
    trtcCloud->setLocalRenderParams(render_params);

    // 初始化 pLocalVideoView
    CWnd* pLocalVideoView = GetDlgItem(AFX_IDC_PICTURE);
    if (pLocalVideoView != nullptr) {
        auto local_view = (liteav::TXView)(pLocalVideoView->GetSafeHwnd());
        // 对采集内容进行本地预览
        trtcCloud->startLocalPreview(local_view);
    }
}
```

调用 `stopLocalPreview` 关闭摄像头预览并停止推送本地视频信息。

```
void CRTCWindowsApp::OnBnClickedButton()
{
    TRTCCloud* trtcCloud = CRTCWindowsApp::GetInstance()->trtc_cloud_;
    trtcCloud->stopLocalPreview();
}
```

步骤6. 打开/关闭麦克风

调用 `startLocalAudio` 开启麦克风采集，请根据您的需求选择以下其中一个声音质量参数 `Quality` 。

```
// 开启麦克风采集，设置当前场景为：语音模式
// 具有高的噪声抑制能力，有强有弱的网络阻力
ITRTCCloud* trtcCloud = CRTCWindowsApp::GetInstance()->trtc_cloud_;
trtcCloud->startLocalAudio(TRTCAudioQualitySpeech);
```

```
// 开启麦克风采集，设置当前场景为：音乐模式
// 为获得高保真度，低音质损失，建议配合专业声卡使用
ITRTCCloud* trtcCloud = CRTCWindowsApp::GetInstance()->trtc_cloud_;
trtcCloud->startLocalAudio(TRTCAudioQualityMusic);
```

版权所有：腾讯云计算（北京）有限责任公司

调用 `stopLocalAudio` 关闭麦克风采集并停止推送本地音频信息。

```
ITRTCCloud* trtcCloud = CRTCWindowsApp::GetInstance()->trtc_cloud_;
trtcCloud->stopLocalAudio();
```

步骤7. 播放/停止视频流

1. 在进入房间之前对 onUserVideoAvailable 进行监听，当您收到 `onUserVideoAvailable(userId, YES)` 通知时，表示该路画面已经有可播放的视频帧到达。
2. 调用 `startRemoteView/stopRemoteView` 选择播放或停止远端视频画面。

```
// 播放视频
CWnd* pLocalVideoView = GetDlgItem(AFX_IDC_PICTURE);
if (pLocalVideoView != nullptr) {
    auto local_view = (liteav::TXView)(pLocalVideoView->GetSafeHwnd());
    ITRTCCloud* trtcCloud = CRTCWindowsApp::GetInstance()->trtc_cloud_;
    trtcCloud->startRemoteView("denny", liteav::TRTCVideoStreamTypeBig, video_view);
}
```

调用 `stopRemoteView` 停止播放远端画面。

```
// 停止播放
ITRTCCloud* trtcCloud = CRTCWindowsApp::GetInstance()->trtc_cloud_;
trtcCloud->stopRemoteView("denny", liteav::TRTCVideoStreamTypeBig); // 仅停止播放 denny 的视频
trtcCloud->stopAllRemoteView(); // 停止所有视频播放
```

步骤8. 播放/停止音频流

调用 `muteRemoteAudio` 选择播放或停止远端声音。

```
// 静音
ITRTCCloud* trtcCloud = CRTCWindowsApp::GetInstance()->trtc_cloud_;
trtcCloud->muteRemoteAudio("denny", true); // 仅静音 denny

trtcCloud->muteAllRemoteAudio(true); // 静音所有远端用户
```

```
// 取消静音
ITRTCCloud* trtcCloud = CRTCWindowsApp::GetInstance()->trtc_cloud_;
trtcCloud->muteRemoteAudio("denny", false); // 仅取消静音 denny

trtcCloud->muteAllRemoteAudio(false); // 取消静音所有远端用户
```

步骤9. 退出房间

在资源文件 IDD_TRTCDEMO_DIALOG 中添加一个 Button 组件，双击新建的 Button 后调用 `exitRoom` 退出当前的房间，TRTC SDK 会在退房结束后通过 `onExitRoom` 回调事件通知您。

```
void CRTCWindowsApp::OnBnClickedButton()
{
    ITRTCCloud* trtcCloud = CRTCWindowsApp::GetInstance()->trtc_cloud_;
```

```
    trtcCloud->exitRoom();
}
```

常见问题

- 您可以在 API 概览 查看所有函数列表及其描述。
- 如果您的接入和使用中遇到问题，请参见 常见问题 。

联系我们

如果您是开发者，欢迎您加入我们的 RTC Engine 技术交流平台 zhiliao，进行技术交流和产品沟通。

Windows C#

最近更新时间：2025-03-25 14:53:02

如何快速在 C# 中接入 Windows RTC Engine，以实现基本音视频通话功能。

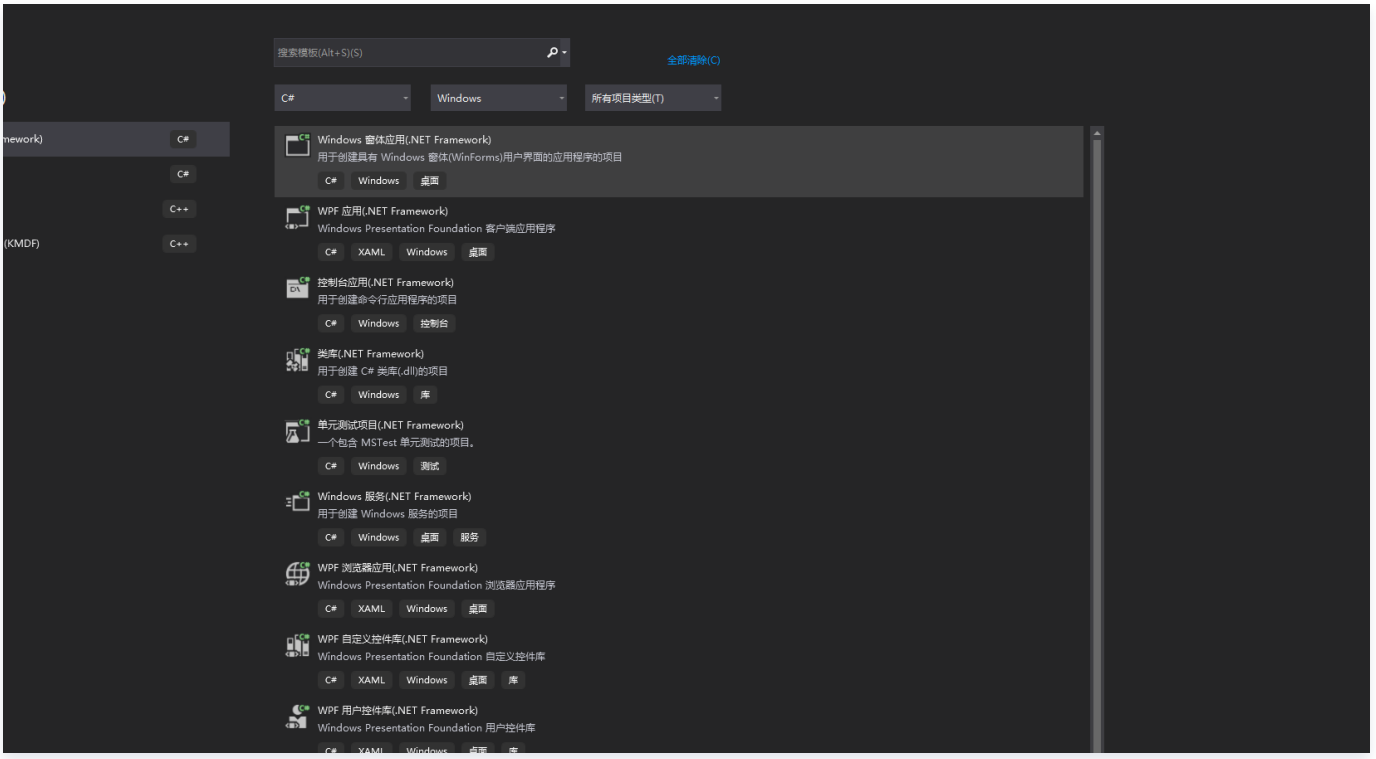
环境准备

- **操作系统**：Windows 7 及以上版本。
- **开发环境**：Visual Studio 2019 及以上版本，推荐使用 Visual Studio 2022。
- **开发工具包**：Visual Studio 选中 .NET 桌面开发工具包。

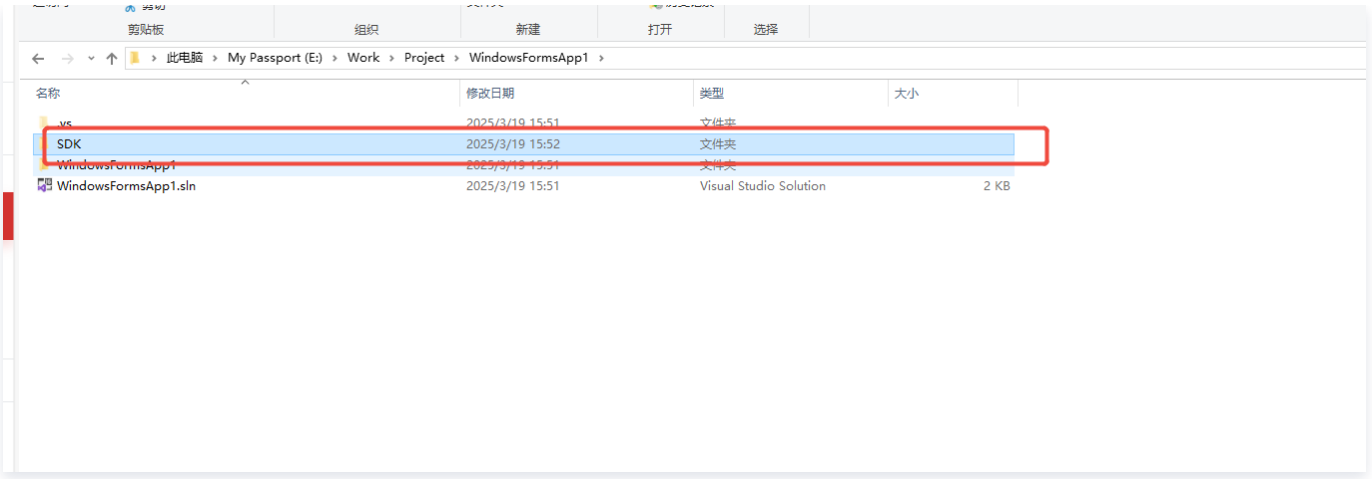
接入指引

步骤 1. 导入 SDK

1. 打开 **Visual Studio**，新建一个 **Windows Forms App (.NET Framework)** 项目。在创建项目的向导中，按照提示完成项目创建，单击**创建**。



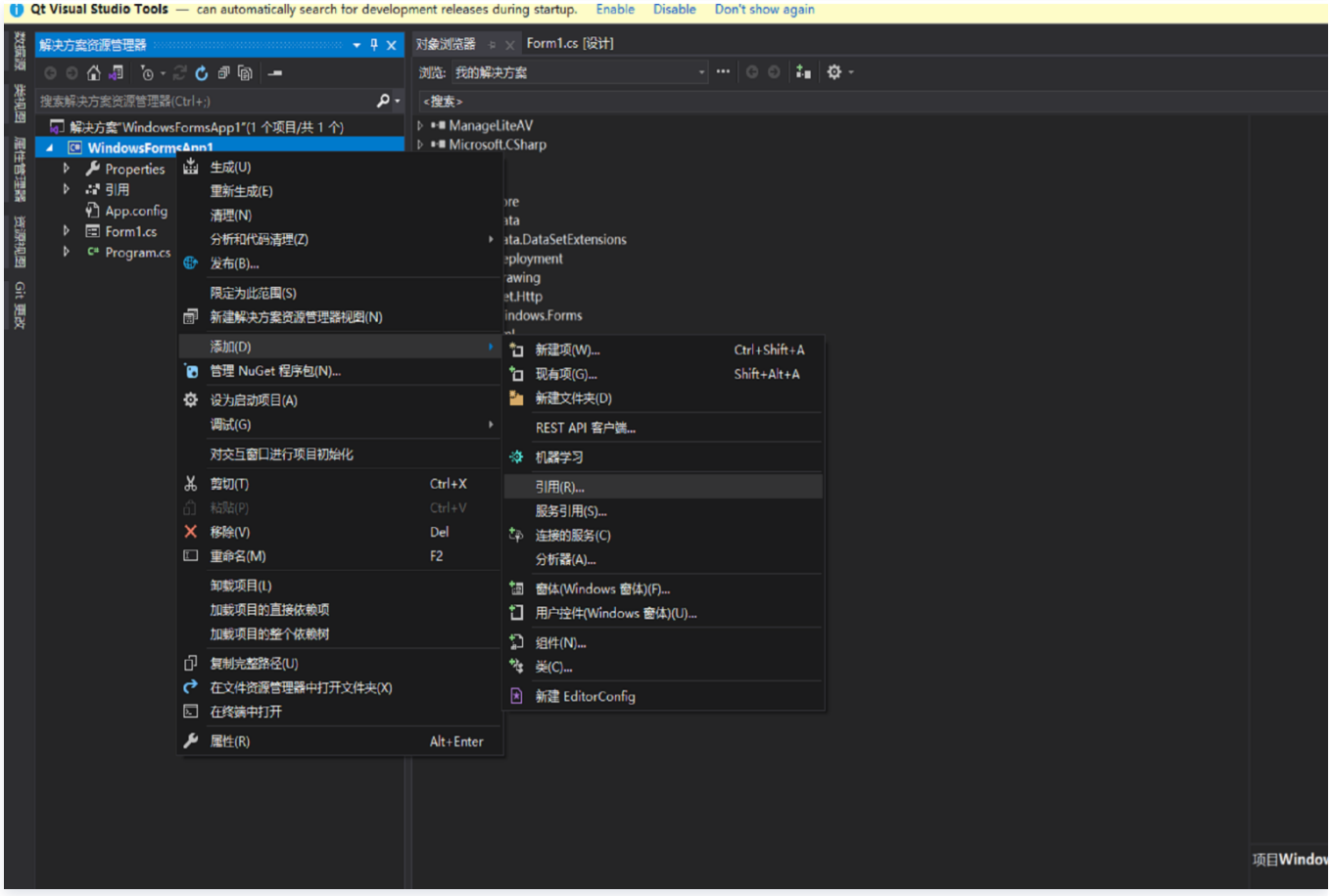
2. 下载 Windows C# SDK，将解压后的 **SDK** 文件拷贝到项目 .sln 文件所在目录下。



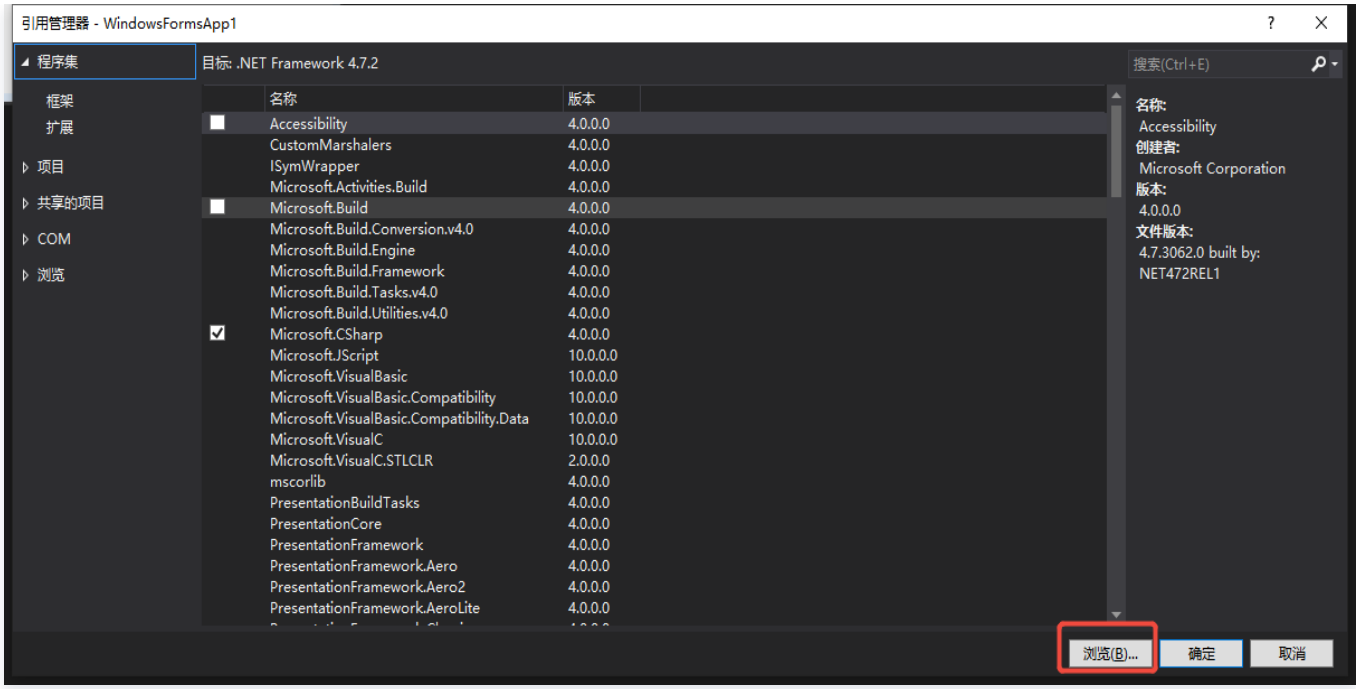
步骤 2. 配置项目

1. **添加引用：**

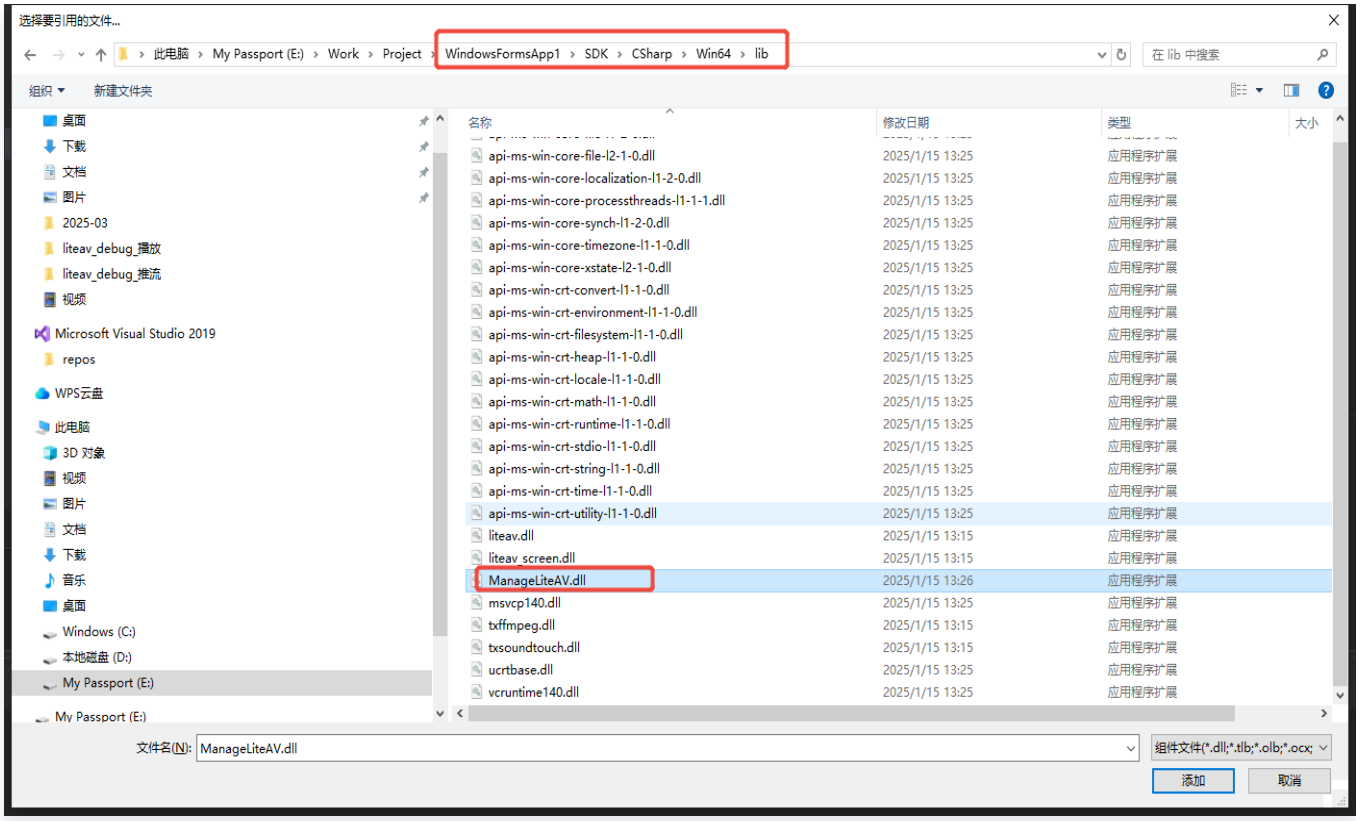
1.1 在**解决方案资源管理器**中，右键点击项目名称，选择**添加 > 引用**。

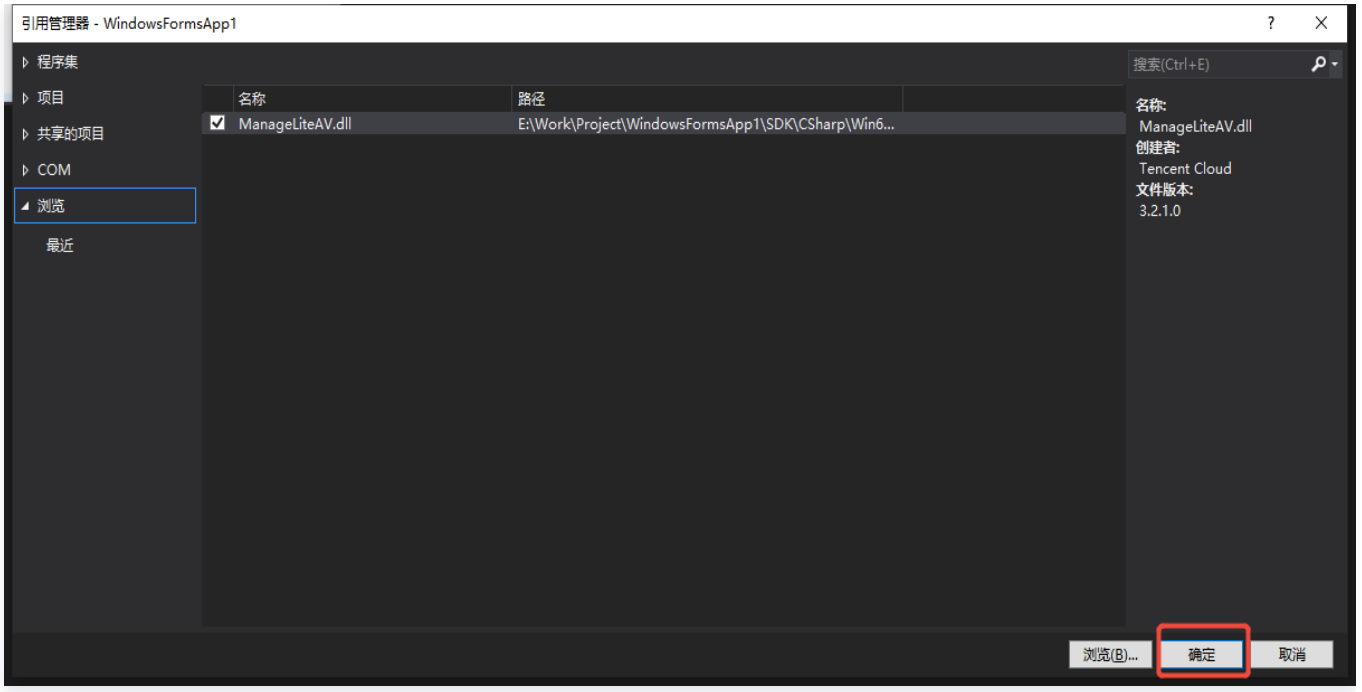


1.2 在弹出的**引用管理器**窗口中，单击**浏览**。



1.3 找到解压后 SDK 文件夹中的 `.dll` 文件，选择需要的程序集进行引用。





2. **复制 DLL 文件：**

   为了确保程序运行时能够找到所需的 DLL 文件，在项目属性的**生成事件 > 后期生成事件命令行**中添加以下命令：

```
copy /Y "$(ProjectDir)..\SDK\CSharp\!Platform!\lib\*.dll" "$(ProjectDir)$(OutDir)"
```

3. **打印 SDK 版本号：**

- 在 `Form1.cs` 文件中引入命名空间：

```
using ManageLiteAV;
```

- 在 `Form1` 类的构造函数或者 `Load` 事件中添加以下代码：

```
private void Form1_Load(object sender, EventArgs e)
{
    ITRTCCloud trtcCloud = ITRTCCloud.getTRTCShareInstance();
    string version = trtcCloud.getSDKVersion();
    MessageBox.Show($"SDK version: {version}");
}
```

步骤 3. 创建 TRTC 实例

在 `Form1.cs` 文件中实现 `ITRTCCloudCallback` 接口，并声明 `TRTCCloud` 成员变量：

> ⚠ **注意：**
>
> 目前 `ITRTCCloudCallback` 的所有回调都得实现不然会出现编译报错

```
using System.Windows.Forms;
using ManageLiteAV;

public partial class Form1 : Form, ITRTCCloudCallback
```

```
    {
        private ITRTCCloud trtcCloud;

        public Form1()
        {
            InitializeComponent();
            trtcCloud = ITRTCCloud.getTRTCShareInstance();
            trtcCloud.addCallback(this);
        }


        // 实现接口方法
        public void onError(TXLiteAVError errCode, string errMsg, object extraInfo)
        {
            // 监听 "onError" 事件并对相应信息进行打印
            MessageBox.Show($"Error: {errCode}, {errMsg}");
        }


        public void onWarning(TXLiteAVWarning warningCode, string warningMsg, object
extraInfo)
        {
            // 监听 "onWarning" 事件并对相应信息进行打印
            MessageBox.Show($"Warning: {warningCode}, {warningMsg}");
        }


        public void onEnterRoom(int result)
        {
            // 监听 "onEnterRoom" 事件并对相应信息进行打印
            if (result > 0)
            {
                MessageBox.Show("Enter room successfully");
            }
            else
            {
                MessageBox.Show("Enter room failed");
            }
        }

        public void onExitRoom(int reason)
        {
            // 监听 "onExitRoom" 事件并对相应信息进行打印
            if (reason == 0)
            {
                MessageBox.Show("Exit current room by calling the 'exitRoom' api of sdk");
            }
            else if (reason == 1)
            {
                MessageBox.Show("Kicked out of the current room by server through the
restful api");
            }
            else if (reason == 2)
            {
```

```
                MessageBox.Show("The current room is dissolved by server through the restful
api");
            }
        }
    }
```

步骤 4. 进入房间

```
//在 `Form1.cs` 文件中添加一个按钮，并为按钮的 `Click` 事件添加以下代码：
private void button1_Click(object sender, EventArgs e)
{
    // 将以下 trtcParams 参数修改为自己的参数
    TRTCParams trtcParams = new TRTCParams();
    trtcParams.sdkAppId = 1400000123;
    trtcParams.userId = "denny";
    trtcParams.roomId = 123321;
    trtcParams.userSig = "xxx";

    // 对于多人视频通话场景，推荐使用 TRTC_APP_SCENE_LIVE
    trtcCloud.enterRoom(trtcParams, TRTCAppScene.TRTC_APP_SCENE_LIVE);
}
```

步骤 5. 打开/关闭摄像头

1. 在工具箱中，拖动一个 `PictureBox` 控件到窗体上。
2. 为窗体添加一个 `Button` 控件，双击该按钮，在生成的 `Click` 事件处理方法中添加以下代码：

```
private void button2_Click(object sender, EventArgs e)
{
    // 设置本地预览渲染参数
    TRTCRenderParams renderParams = new TRTCRenderParams();
    renderParams.mirrorType = TRTCVideoMirrorType.TRTC_VIDEO_MIRROR_TYPE_ENABLE;
    renderParams.fillMode = TRTCVideoFillMode.TRTC_VIDEO_FILL_MODE_FILL;

    // 设置本地预览视图
    trtcCloud.setLocalRenderParams(ref renderParams);
    trtcCloud.startLocalPreview(pictureBox1.Handle);
}
```

以上文档介绍了如何在 C# 项目中接入 Windows RTC Engine，实现基本的音视频通话功能。在实际使用时，请根据自己的项目需求进行调整。

Mac

最近更新时间：2025-03-04 11:42:03

本文将介绍如何快速完成 Objective-C 版本的 Mac RTC Engine 的接入，实现一个基本的音视频通话。

环境准备

- Xcode 13.0+。
- OS X10.10+ 的 Mac 真机。
- 项目已配置有效的开发者签名。

接入指引

步骤1. 导入 SDK

1. 在终端窗口执行以下命令安装 **CocoaPods** ，如果您已经完成 **CocoaPods** 的安装，可以跳过该步。

```
sudo gem install cocoapods
```

2. 在终端窗口中进入**项目根目录**，执行以下命令为项目创建 **Podfile** 文件。

```
pod init
```

3. 编辑并保存 **Podfile** 内容如下。

```
platform :osx, '10.10'

# 将 Your Target 修改为您项目的名称
target 'Your Target' do
pod 'TXLiteAVSDK_TRTC_Mac', :podspec =>
'https://liteav.sdk.qcloud.com/pod/liteavsdkspec/TXLiteAVSDK_TRTC_Mac.podspec'
end
```

4. 在终端窗口中执行以下命令更新本地库文件，完成对 **TRTC SDK** 的下载。

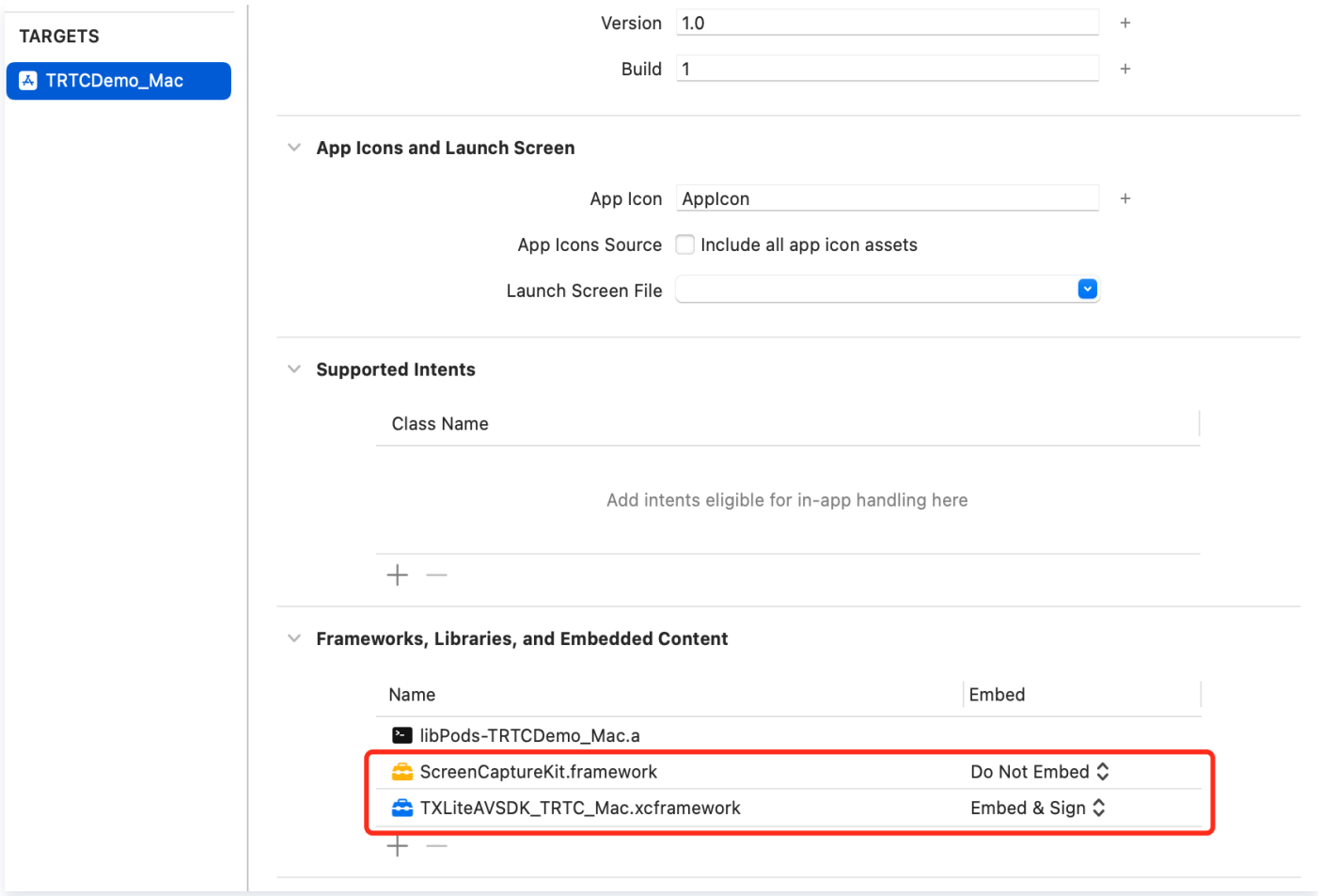
```
pod install
```

> **说明：**
> `pod install` 执行完成，会生成新的的 **.xcworkspace** 工程文件，双击打开**.xcworkspace** 文件进行后续工作。

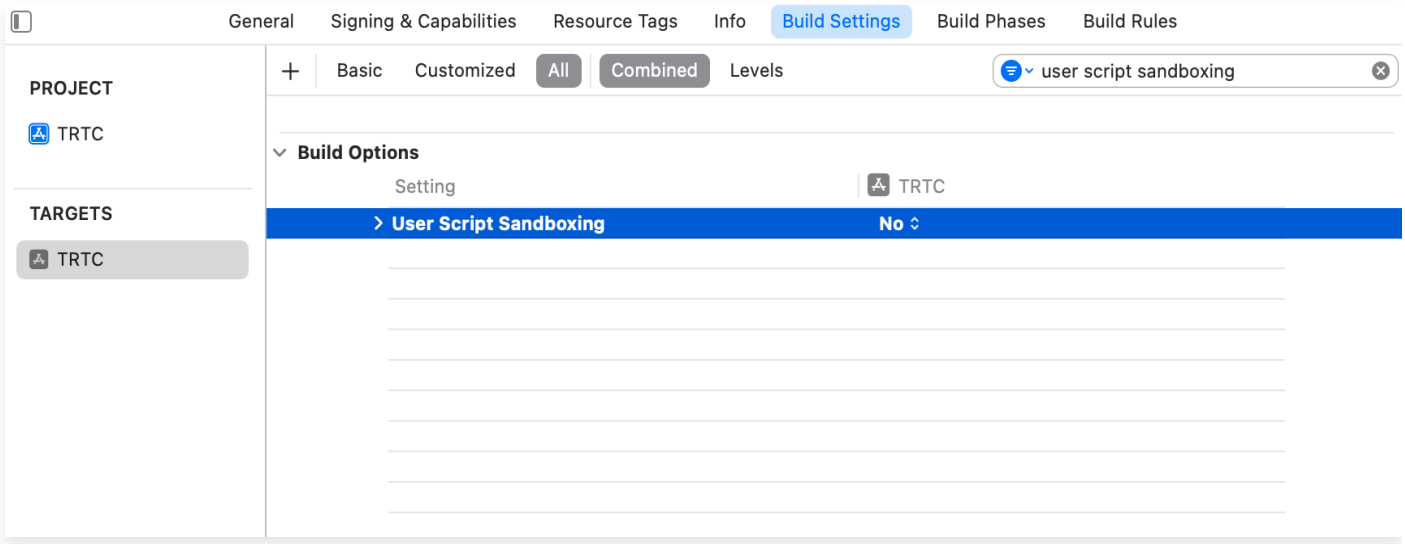
步骤2. 配置项目

1. 打开 `.xcworkspace` 工程文件后，在 **Xcode** 的导航栏中点击左侧的 **Project Navigator**，点击您的项目名称并确保在编辑区域选择正确的 **TARGETS**。
2. 在 `General` 选项卡的 **Frameworks, Libraries, and Embedded Content** 部分添加 **TXLiteAVSDK_TRTC_Mac.xcframework** 和 **ScreenCaptureKit.framework**。

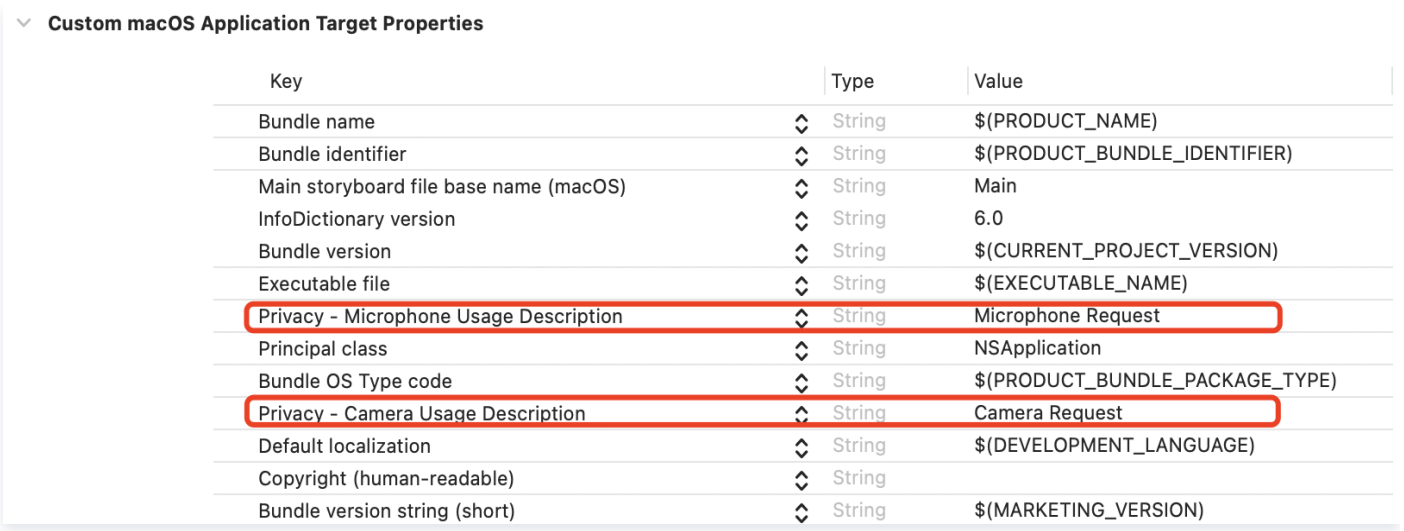
版权所有：腾讯云计算（北京）有限责任公司



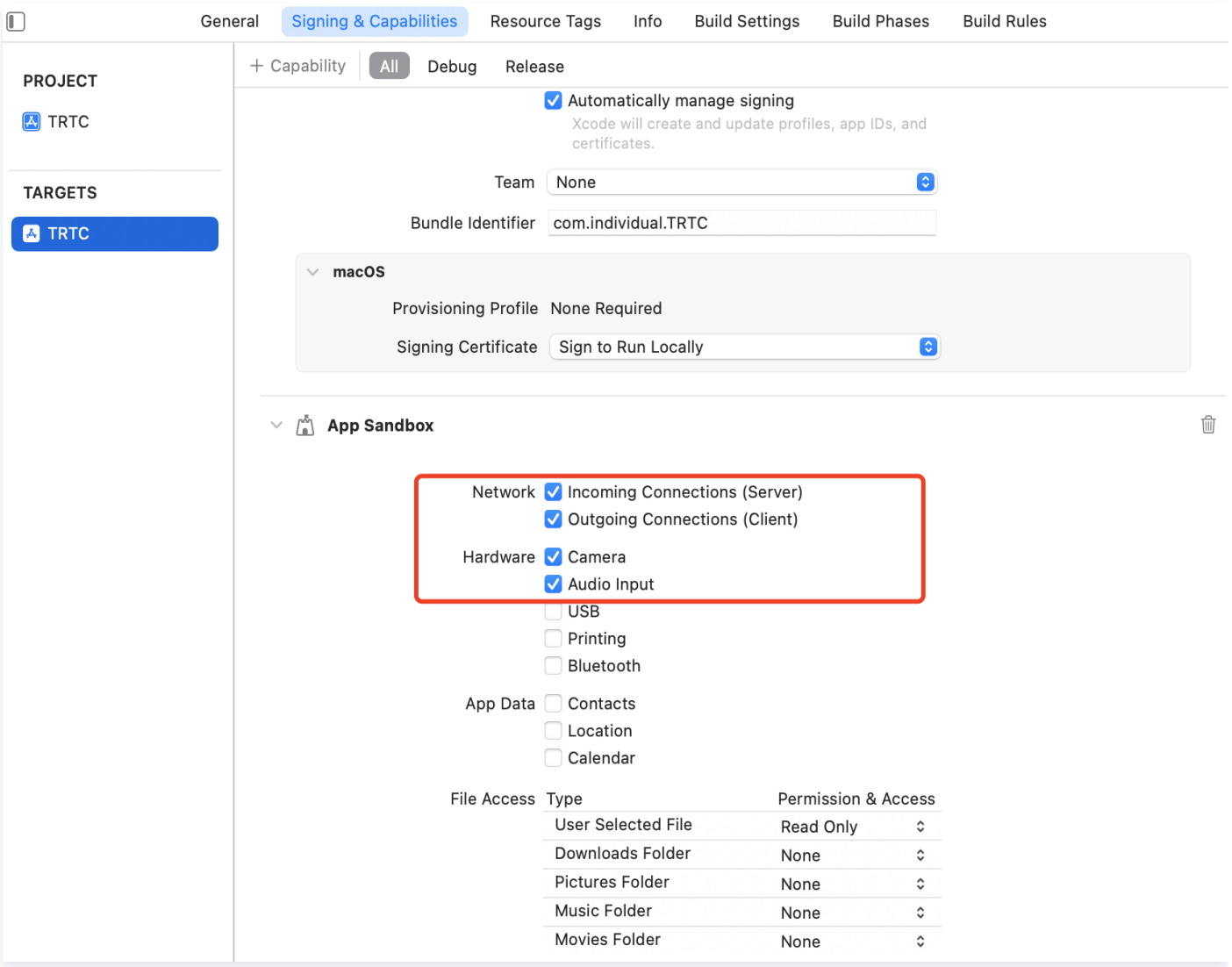
3. 在 `Build Settings` 选项卡中搜索 **User Script Sandboxing**，将其值置为 `No`，以允许用户脚本访问更广泛的系统资源和文件。



4. 在 `Info.plist` 选项卡中添加 **Privacy-Microphone Usage Description** 和 **Privacy-Microphone Usage Description**，并填入 **Microphone/Camera** 使用的目标提示语，获取麦克风和摄像头的使用权限。



5. 在 `Signing & Capabilities` 选项卡中 **App Sandbox** 部分勾选以下内容。



步骤3. 创建 TRTC 实例

1. 在 `AppDelegate.h` 文件中引入 TRTC SDK。

```
@import TXLiteAVSDK_TRTC_Mac; // 引入 TRTC SDK 模块
```

2. 在 `AppDelegate.h` 文件中声明 TRTCCloud 属性。

```
@property (nonatomic, strong) TRTCCloud *trtcCloud; // 添加 TRTCCloud 属性
```

3. 进入 `AppDelegate.m` 文件后，在 `applicationDidFinishLaunching` 方法中调用 `sharedInstance` 创建 TRTC 实例，并设置事件监听。

```
#import <UserNotifications/UserNotifications.h> // 导入 UserNotifications 框架

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification {
    // 创建 TRTC 实例并设置监听
    _trtcCloud = [TRTCCloud sharedInstance];
    _trtcCloud.delegate = self;
}

// 监听 `onError` 事件
- (void)onError:(TXLiteAVError)errCode
         errMsg:(nullable NSString *)errMsg
        extInfo:(nullable NSDictionary *)extInfo{
    // 处理 `Error` 事件，建议对 OnError 的信息通过请求通知权限进行提示

    // 创建一个用户通知中心实例
    NSString *tip = [NSString stringWithFormat:@"Error Code: %ld, Message: %@",
(long)errCode, errMsg];
    // 请求通知权限
    UNUserNotificationCenter *center = [UNUserNotificationCenter
currentNotificationCenter];
    [center requestAuthorizationWithOptions:(UNAuthorizationOptionAlert |
UNAuthorizationOptionSound)
                          completionHandler:^(BOOL granted, NSError * _Nullable error)
{
                              if (granted) {
                                  // 创建通知内容
                                  UNMutableNotificationContent *content =
[[UNMutableNotificationContent alloc] init];
                                  content.title = @"RTC Error";
                                  content.body = tip;

                                  // 创建通知触发器
                                  UNTimeIntervalNotificationTrigger *trigger =
[UNTimeIntervalNotificationTrigger triggerWithTimeInterval:5 repeats:NO];

                                  // 创建通知请求
                                  UNNotificationRequest *request =
[UNNotificationRequest requestWithIdentifier:@"RTCErrorNotification" content:content
trigger:trigger];

                                  // 添加通知到通知中心
                                  [center addNotificationRequest:request
withCompletionHandler:^(NSError * _Nullable error) {
                                      if (error != nil) {
                                          NSLog(@"Error adding notification: %@",
error);
```

```
                                        }
                                    }];
                                }
                            }];
}
```

步骤4. 进入房间

设置进房参数 `TRTCParams` 并调用 `enterRoom` 即可成功进入房间，该方法通常在点击**开始通话按钮**后调用。

参数	类型	描述
sdkAppId	number	您在 TRTC 控制台 中创建的音频和视频应用程序的sdkAppId。
userId	string	您指定的用户 ID。
userSig	string	用户签名，请参见 UserSig 。
roomId	number	您指定的房间 ID，通常是唯一的房间 ID。

有关更详细的参数描述，请参考接口文档 enterRoom 。

```
#import "AppDelegate.h" // 导入 "AppDelegate.h" 文件

- (void)enterRoom {
    // 将以下 trtcParams 参数修改为自己的参数
    TRTCParams *trtcParams = [[TRTCParams alloc] init];
    trtcParams.sdkAppId = 1400000123;
    trtcParams.userId = @"denny";
    trtcParams.userSig = @"";
    trtcParams.roomId = 123321;

    // 对于多人视频通话场景，推荐使用 TRTC_APP_SCENE_LIVE
    AppDelegate *appDelegate = (AppDelegate *)[[UIApplication sharedApplication] delegate];
    [appDelegate.trtcCloud enterRoom:trtcParams appScene:TRTCAppSceneLIVE];
}

// 监听 `onEnterRoom` 事件
- (void)onEnterRoom:(NSInteger)result {
    // 处理 onEnterRoom 事件
}
```

步骤5. 打开/关闭摄像头

1. 在 `ViewController.h` 文件中声明 **NSWindow** 和 **NSView** 属性。

```
@property (nonatomic, strong) NSWindow *window; // 添加 NSWindow 属性
@property (nonatomic, strong) NSView *localCameraVideoView; // 添加 NSView 属性
```

2. 对 **localCameraVideoView** 进行初始化，设置本地预览的渲染参数 `setLocalRenderParams` ，并调用 `startLocalPreview` 进行本地预览，成功调用 `enterRoom` 后开始推流。

```
-(void) startLocalPreview {
```

```
    // 创建 window
    self.window = [[NSWindow alloc] initWithContentRect:NSMakeRect(0, 0, 800, 600)
styleMask:NSWindowStyleMaskTitled | NSWindowStyleMaskClosable |
NSWindowStyleMaskMiniaturizable | NSWindowStyleMaskResizable
backing:NSBackingStoreBuffered defer:NO];
    [self.window center];
    [self.window setTitle:@"TRTCDemo_Mac"];
    [self.window makeKeyAndOrderFront:nil];
    self.window.releasedWhenClosed = NO;

    // 初始化 localCameraVideoView
    self.localCameraVideoView = [[NSView alloc] initWithFrame:NSMakeRect(0, 0, 300, 300)];
    [self.window.contentView addSubview:self.localCameraVideoView];

    // 调整 localCameraVideoView 大小
    self.localCameraVideoView.frame = self.window.contentView.bounds;

    // 设置本地预览渲染参数
    TRTCRenderParams *trtcRenderParams = [[TRTCRenderParams alloc] init];
    trtcRenderParams.fillMode   = TRTCVideoFillMode_Fill;
    trtcRenderParams.mirrorType = TRTCVideoMirrorTypeAuto;

    AppDelegate *appDelegate = (AppDelegate *)[[NSApplication sharedApplication]
delegate];
    [appDelegate.trtcCloud setLocalRenderParams:trtcRenderParams];

    // 对采集内容进行本地预览
    [appDelegate.trtcCloud startLocalPreview:self.localCameraVideoView];
}
```

调用 `stopLocalPreview` 关闭摄像头预览并停止推送本地视频信息。

```
AppDelegate *appDelegate = (AppDelegate *)[[NSApplication sharedApplication] delegate];
[appDelegate.trtcCloud stopLocalPreview];
```

步骤6. 打开/关闭麦克风

调用 `startLocalAudio` 开启麦克风采集，请根据您的需求选择以下其中一个声音质量参数 `Quality` 。

```
// 开启麦克风采集，设置当前场景为：语音模式
// 具有高的噪声抑制能力，有强有弱的网络阻力
AppDelegate *appDelegate = (AppDelegate *)[[NSApplication sharedApplication] delegate];
[appDelegate.trtcCloud startLocalAudio:TRTCAudioQualitySpeech];
```

```
// 开启麦克风采集，设置当前场景为：音乐模式
// 为获得高保真度，低音质损失，建议配合专业声卡使用
AppDelegate *appDelegate = (AppDelegate *)[[NSApplication sharedApplication] delegate];
[appDelegate.trtcCloud startLocalAudio:TRTCAudioQualityMusic];
```

调用 `stopLocalAudio` 关闭麦克风采集并停止推送本地音频信息。

```
AppDelegate *appDelegate = (AppDelegate *)[[NSApplication sharedApplication] delegate];
[appDelegate.trtcCloud stopLocalAudio];
```

步骤7. 播放/停止视频流

1. 在进入房间之前对 onUserVideoAvailable 进行监听，当您收到 `onUserVideoAvailable(userId, YES)` 通知时，表示该路画面已经有可播放的视频帧到达。
2. 调用 `startRemoteView` 进行本地预览，成功调用 `enterRoom` 后开始推流。

```
- (void)startRemoteView {
    // 播放远端画面
    AppDelegate *appDelegate = (AppDelegate *)[[NSApplication sharedApplication]
delegate];
    [appDelegate.trtcCloud startRemoteView:@"denny" streamType:TRTCVideoStreamTypeBig
view:self.localCameraVideoView];
}
```

调用 `stopRemoteView` 停止播放远端画面。

```
// 停止播放
AppDelegate *appDelegate = (AppDelegate *)[[NSApplication sharedApplication] delegate];
[appDelegate.trtcCloud stopRemoteView:@"denny"];  // 仅停止播放 denny 画面
[appDelegate.trtcCloud stopAllRemoteView]; // 停止所有视频播放
```

步骤8. 播放/停止音频流

观众端可以调用 `muteRemoteAudio` 选择播放或停止远端声音。

```
// 静音
AppDelegate *appDelegate = (AppDelegate *)[[NSApplication sharedApplication] delegate];
[appDelegate.trtcCloud muteRemoteAudio:@"denny" mute:YES];// 仅静音 denny

[appDelegate.trtcCloud muteAllRemoteAudio:YES]; // 静音所有远端用户
```

```
// 取消静音
AppDelegate *appDelegate = (AppDelegate *)[[NSApplication sharedApplication] delegate];
[appDelegate.trtcCloud muteRemoteAudio:@"denny" mute:NO]; // 仅取消静音 denny

[appDelegate.trtcCloud muteAllRemoteAudio:NO]; // 取消静音所有远端用户
```

步骤9. 退出房间

调用 `exitRoom` 退出当前的房间，TRTC SDK 会在退房结束后通过 `onExitRoom` 回调事件通知您。

```
// 退出当前房间
AppDelegate *appDelegate = (AppDelegate *)[[NSApplication sharedApplication] delegate];
[appDelegate.trtcCloud exitRoom];

// 监听 `onExitRoom` 回调
- (void)onExitRoom:(NSInteger)reason {
```

```
    if (reason == 0) {
        NSLog(@"Exit current room by calling the 'exitRoom' api of sdk ...");
    } else if (reason == 1) {
        NSLog(@"Kicked out of the current room by server through the restful api...");
    } else if (reason == 2) {
        NSLog(@"Current room is dissolved by server through the restful api...");
    }
}
```

常见问题

- 您可以在 API 概览 查看所有函数列表及其描述。
- 如果您的接入和使用中遇到问题，请参见 常见问题 。

联系我们

如果您是开发者，欢迎您加入我们的 RTC Engine 技术交流平台 zhiliao ，进行技术交流和产品沟通。

基础功能
调整视频画质
Android&iOS&Windows&Mac

最近更新时间：2024-12-10 11:16:52

内容介绍

在 TRTCCloud 中，您可以通过以下方式调整画质：

- TRTCCloud.enterRoom 中的 TRTCAppScene 参数：用于选择您的应用场景。
- TRTCCloud.setVideoEncoderParam：用于设置编码参数。
- TRTCCloud.setNetworkQosParam：用于设置网络调控策略。

本文主要介绍如何配置上述参数，使 TRTC SDK 的画质效果符合您的项目需要。
您也可以参考以下 Demo：

- iOS：SetVideoQualityViewController.m
- Android：SetVideoQualityActivity.java
- Windows：TRTCMainViewController.cpp

支持的平台

iOS	Android	Mac OS	Windows	Web	Electron	微信小程序	Flutter
✓	✓	✓	✓	✓	✓	✓	✓

说明：

- Web 端设定画面质量的详细操作，请参见 设定画面质量 。
- 微信小程序端画面质量，可通过 pusherAttributes 进行设置，maxBitrate（最大码率）/minBitrate（最小码率）/videoWidth（上推的视频流的分辨率宽度）/videoHeight（上推的视频流的分辨率高度），请参见 trtx-wx接口文档 。

TRTCAppScene

- **VideoCall**
  对应视频通话场景，即绝大多数时间都是两人或两人以上视频通话的场景，内部编码器和网络协议优化侧重流畅性，降低通话延迟和卡顿率。
- **LIVE**
  对应直播场景，即绝大多数时间都是一人直播，偶尔有多人视频互动的场景，内部编码器和网络协议优化侧重性能和兼容性，性能和清晰度表现更佳。

TRTCVideoEncParam

推荐的配置

应用场景	videoResolution	videoFps	videoBitrate
视频通话（手机）	640x360	15	550kbps
视频会议（主画面 @ Mac Win）	1280x720	15	1200kbps
视频会议（主画面 @ 手机）	640x360	15	900kbps

版权所有：腾讯云计算（北京）有限责任公司

视频会议（小画面）	320x180	15	250kbps
在线教育（老师 @ Mac Win）	960x540	15	850kbps
在线教育（老师 @ iPad）	640x360	15	550kbps
在线教育（学生）	320x180	15	250kbps

各字段详解

- **(TRTCVideoResolution) videoResolution**
  编码分辨率，例如 640 x 360 是指编码出的画面的宽（像素） x 高（像素），我们在 TRTCVideoResolution 枚举定义里只定义了宽 >= 高的横屏（Landscape）分辨率，如果想要使用竖屏分辨率，需要将 resMode 设置为 Portrait。

> **注意：**
> 由于很多硬件编解码器只支持能被 16 整除的像素宽度，所以 SDK 实际编码出的分辨率并不一定完全按照参数自定，而是会自动进行 16 整除修正。例如 640 x 360 的分辨率，在 SDK 内部有可能会适配为 640 x 368。

- **(TRTCVideoResolutionMode) resMode**
  指横屏或竖屏分辨率，由于 TRTCVideoResolution 中只定义了横屏分辨率，如果您希望使用 360 x 640 这样的竖屏分辨率，就需要指定 resMode 为 TRTCVideoResolutionModePortrait。一般 PC 和 Mac 采用横屏（Landscape）分辨率，手机采用竖屏（Portrait）分辨率。
- **(int) videoFps**
  帧率（FPS），也就是每秒钟要编码多少帧画面。推荐设置为 15 FPS，这样既能保证画面足够流畅，又不会因为每秒帧数太多而拉低单幅画面的清晰度。

如果您对流畅度要求比较高，可以设置为 20 FPS 或 25 FPS。但请不要设置 25 FPS 以上的数值，因为电影的常规帧率也只有 24 FPS。

- **(int) videoBitrate**
  视频码率（Bitrate），即每秒钟编码器输出多少 Kbit 的编码后的二进制数据。如果您将 videoBitrate 设置为 800kbps，那么每秒钟编码器会产生 800kbit 的视频数据，这些数据如果存储成一个文件，那么文件大小就是 800kbit，也就是100KB，也就是 0.1M。

视频码率并不是越高越好，它跟分辨率之间要有比较恰当的映射关系，如下表所示。

分辨率码率参照表

分辨率定义	宽高比	建议码率（VideoCall）	建议码率（LIVE）
TRTCVideoResolution_120_120	1:1	80kbps	120kbps
TRTCVideoResolution_160_160	1:1	100kbps	150kbps
TRTCVideoResolution_270_270	1:1	200kbps	300kbps
TRTCVideoResolution_480_480	1:1	350kbps	525kbps
TRTCVideoResolution_160_120	4:3	100kbps	150kbps
TRTCVideoResolution_240_180	4:3	150kbps	225kbps
TRTCVideoResolution_280_210	4:3	200kbps	300kbps
TRTCVideoResolution_320_240	4:3	250kbps	375kbps
TRTCVideoResolution_400_300	4:3	300kbps	450kbps
TRTCVideoResolution_480_360	4:3	400kbps	600kbps

版权所有：腾讯云计算（北京）有限责任公司

TRTCVideoResolution_640_480	4:3	600kbps	900kbps
TRTCVideoResolution_960_720	4:3	1000kbps	1500kbps
TRTCVideoResolution_160_90	16:9	150kbps	250kbps
TRTCVideoResolution_256_144	16:9	200kbps	300kbps
TRTCVideoResolution_320_180	16:9	250kbps	400kbps
TRTCVideoResolution_480_270	16:9	350kbps	550kbps
TRTCVideoResolution_640_360	16:9	550kbps	900kbps
TRTCVideoResolution_960_540	16:9	850kbps	1300kbps
TRTCVideoResolution_1280_720	16:9	1200kbps	1800kbps
TRTCVideoResolution_1920_1080	16:9	2000kbps	3000kbps
TRTCVideoResolution_2560×1440（目前仅Window端支持）	16:9	4000kbps	6000kps
TRTCVideoResolution_4096×2160（目前仅Window端支持）	16:9	6000kbps	9000kps

TRTCNetworkQosParam

QosPreference

在网络带宽比较充裕的情况下，清晰和流畅是可以兼顾的，但当用户的网络并不理想时，究竟是优先保证清晰还是优先保证流畅？您可以通过指定 TRTCNetworkQosParam 中的 preference 参数来做出选择。

- **流畅优先（TRTCVideoQosPreferenceSmooth）**
  在用户遭遇弱网环境时，画面会变得模糊，且会有较多马赛克，但可以保持流畅或轻微卡顿。
- **清晰优先（TRTCVideoQosPreferenceClear）**
  在用户遭遇弱网环境时，画面会尽可能保持清晰，但可能会更容易出现卡顿。

ControlMode

controlMode 参数选择 **TRTCQosControlModeServer** 即可，TRTCQosControlModeClient 是腾讯云研发团队做内部调试用的，请勿关注。

常见的误区

1. 分辨率越高越好?
较高的分辨率也需要较高的码率来支撑，如果分辨率选择 1280 x 720，但码率却指定为 200kbps，画面就会有大量的马赛克。推荐参考 分辨率码率参照表 进行设置。
2. 帧率越高越好?
由于摄像头采集的画面是曝光阶段中所有现实物体的完整映射，所以并不是帧率越高，感官就越流畅，这一点跟游戏里的FPS是不一样的。恰恰相反，帧率过高，会拉低每帧画面的画质，也会减少摄像机的曝光时间，效果可能会更差。
3. 码率越高越好?
较高的码率也需要较高的分辨率来匹配，对于 320 x 240 这样分辨率，1000kbps 的码率就很浪费了，推荐参考 分辨率码率参照表 进行设置。
4. 用 Wi-Fi 的时候就可以设置很高的分辨率和码率
并不是说 Wi-Fi 的网速是恒定不变的，如果离无线路由器较远， 或者路由器信道被占用，可能网速还不如 4G。
针对这种情况， TRTC SDK 提供了测速功能，可以在视频通话前先进行测速，根据打分值来确定网络好坏。

Web

最近更新时间：2023-08-16 16:41:02

本文主要介绍如何在视频通话或互动直播中设置视频属性（包括分辨率、帧率和码率），开发者可以根据具体业务需求调整视频画面的清晰度和流畅度，获得更好的用户体验。

> **说明：**
> - 本教程基于 5.x TRTC Web SDK 实现，若您使用 4.x 版本 SDK，可参考 此教程 。
> - 5.x 与 4.x 版本的区别 。

设置分辨率、帧率、码率

通过 trtc 对象的 `TRTC.startLocalVideo()` 或 `TRTC.updateLocalVideo()` 方法设置视频属性：

- 指定一个预定义的 Profile，每个 Profile 对应着一套推荐的分辨率、帧率和码率。

```
// 启动时指定视频属性
await trtc.startLocalVideo({
    option: { profile: '480p' }
});
// 通话过程中动态调整视频属性
await trtc.updateLocalVideo({
    option: { profile: '360p' }
});
```

- 指定自定义分辨率、帧率和码率。

```
// 启动时指定视频属性
await trtc.startLocalVideo({
    option: { profile: { width: 640, height: 480, frameRate: 15, bitrate: 500 /* kpbs */}
}
});
// 通话过程中动态调整视频属性
await trtc.updateLocalVideo({
    option: { profile: { width: 640, height: 360, frameRate: 15, bitrate: 400 /* kpbs */}
}
});
```

设置清晰度优先、流畅度优先

在弱网下，TRTC 对于摄像头及屏幕分享有不同的编码策略。

- 对于摄像头，默认是流畅度优先。即在弱网时，会通过降低编码分辨率的方式，来优先保障编码帧率。
- 对于屏幕分享，默认是清晰度优先。即在弱网时，会通过降低编码帧率的方式，来优先保障编码分辨率。

若默认的编码策略不符合您的需求，您可以通过如下方式调整编码策略。

```
// 将摄像头改为清晰度优先
await trtc.startLocalVideo({
    option: {
      qosPreference: TRTC.TYPE.QOS_PREFERENCE_CLEAR
    }
```

```
});

// 可以动态调整
await trtc.updateLocalVideo({
    option: {
      qosPreference: TRTC.TYPE.QOS_PREFERENCE_SMOOTH
    }
});

// 将屏幕分享改为流畅度优先
await trtc.startScreenShare({
    option: {
      qosPreference: TRTC.TYPE.QOS_PREFERENCE_SMOOTH
    }
})

// 可以动态调整
await trtc.updateScreenShare({
    option: {
      qosPreference: TRTC.TYPE.QOS_PREFERENCE_CLEAR
    }
})
```

视频属性 Profile 列表

视频 Profile	分辨率（宽 x 高）	帧率（fps）	码率（kbps）
120p	160 x 120	15	200
180p	320 x 180	15	350
240p	320 x 240	15	400
360p	640 x 360	15	800
480p	640 x 480	15	900
720p	1280 x 720	15	1500
1080p	1920 x 1080	15	2000
1440p	2560 x 1440	30	4860
4K	3840 x 2160	30	9000

注意：
由于设备和浏览器的限制，视频分辨率不一定能够完全匹配，在这种情况下，浏览器会自动调整分辨率使其接近 Profile 对应的分辨率。

Electron

最近更新时间：2023-09-18 18:08:02

本文主要介绍如何在视频通话或互动直播中设置画面质量，开发者可以根据具体业务需求调整视频画面的清晰度和流畅度，获得更好的用户体验。
视频属性包括分辨率、帧率和码率。

内容介绍

在 Electron SDK 中，您可以通过以下方式调整画质：

- enterRoom 中的 TRTCAppScene 参数：用于选择您的应用场景。
- setVideoEncoderParam ：用于设置编码参数。
- setNetworkQosParam ：用于设置网络调控策略。

本文主要介绍如何配置上述参数，使 TRTC SDK 的画质效果符合您的项目需要。
您也可以参考 Electron API Example：video-quality Demo。

TRTCAppScene

- **VideoCall：**对应视频通话场景，即绝大多数时间都是两人或两人以上视频通话的场景，内部编码器和网络协议优化侧重流畅性，降低通话延迟和卡顿率。
- **LIVE：**对应直播场景，即绝大多数时间都是一人直播，偶尔有多人视频互动的场景，内部编码器和网络协议优化侧重性能和兼容性，性能和清晰度表现更佳。

TRTCVideoEncParam

推荐的配置

应用场景	videoResolution	videoFps	videoBitrate
视频会议（主画面 @ Mac Win）	1280x720	15	1200kbps
在线教育（老师 @ Mac Win）	960x540	15	850kbps

各字段详解

- **(TRTCVideoResolution) videoResolution**
  编码分辨率，例如 640 x 360 是指编码出的画面的宽（像素）x 高（像素），我们在 TRTCVideoResolution 枚举定义里只定义了宽 >= 高的横屏（Landscape）分辨率，如果想要使用竖屏分辨率，需要将 resMode 设置为 Portrait。

  > ⚠ **注意**
  > 由于很多硬件编解码器只支持能被 16 整除的像素宽度，所以 SDK 实际编码出的分辨率并不一定完全按照参数自定，而是会自动进行 16 整除修正。例如 640 x 360 的分辨率，在 SDK 内部有可能会适配为 640 x 368。

- **(TRTCVideoResolutionMode) resMode**
  指横屏或竖屏分辨率，由于 TRTCVideoResolution 中只定义了横屏分辨率，如果您希望使用 360 x 640 这样的竖屏分辨率，就需要指定 resMode 为 TRTCVideoResolutionModePortrait。一般 PC 和 Mac 采用横屏（Landscape）分辨率，手机采用竖屏（Portrait）分辨率。
- **(int) videoFps**
  帧率（FPS），也就是每秒钟要编码多少帧画面。推荐设置为 15 FPS，这样既能保证画面足够流畅，又不会因为每秒帧数太多而拉低单幅画面的清晰度。

如果您对流畅度要求比较高，可以设置为 20 FPS 或 25 FPS。但请不要设置 25 FPS 以上的数值，因为电影的常规帧率也只有 24 FPS。

版权所有：腾讯云计算（北京）有限责任公司

- **(int) videoBitrate**
  视频码率（Bitrate），即每秒钟编码器输出多少 Kbit 的编码后的二进制数据。如果您将 videoBitrate 设置为 800kbps，那么每秒钟编码器会产生 800kbit 的视频数据，这些数据如果存储成一个文件，那么文件大小就是 800kbit，也就是100KB，也就是 0.1M。

视频码率并不是越高越好，它跟分辨率之间要有比较恰当的映射关系，如下表所示。

分辨率码率参照表

分辨率定义	宽高比	建议码率（VideoCall）	建议码率（LIVE）
TRTCVideoResolution_120_120	1:1	80kbps	120kbps
TRTCVideoResolution_160_160	1:1	100kbps	150kbps
TRTCVideoResolution_270_270	1:1	200kbps	300kbps
TRTCVideoResolution_480_480	1:1	350kbps	525kbps
TRTCVideoResolution_160_120	4:3	100kbps	150kbps
TRTCVideoResolution_240_180	4:3	150kbps	225kbps
TRTCVideoResolution_280_210	4:3	200kbps	300kbps
TRTCVideoResolution_320_240	4:3	250kbps	375kbps
TRTCVideoResolution_400_300	4:3	300kbps	450kbps
TRTCVideoResolution_480_360	4:3	400kbps	600kbps
TRTCVideoResolution_640_480	4:3	600kbps	900kbps
TRTCVideoResolution_960_720	4:3	1000kbps	1500kbps
TRTCVideoResolution_160_90	16:9	150kbps	250kbps
TRTCVideoResolution_256_144	16:9	200kbps	300kbps
TRTCVideoResolution_320_180	16:9	250kbps	400kbps
TRTCVideoResolution_480_270	16:9	350kbps	550kbps

TRTCVideoResolution_640_360	16:9	550kbps	900kbps
TRTCVideoResolution_960_540	16:9	850kbps	1300kbps
TRTCVideoResolution_1280_720	16:9	1200kbps	1800kbps
TRTCVideoResolution_1920_1080	16:9	2000kbps	3000kbps

TRTCNetworkQosParam

QosPreference

在网络带宽比较充裕的情况下，清晰和流畅是可以兼顾的，但当用户的网络并不理想时，究竟是优先保证清晰还是优先保证流畅？您可以通过指定 TRTCNetworkQosParam 中的 preference 参数来做出选择。

- **流畅优先（TRTCVideoQosPreferenceSmooth）**
  在用户遭遇弱网环境时，画面会变得模糊，且会有较多马赛克，但可以保持流畅或轻微卡顿。
- **清晰优先（TRTCVideoQosPreferenceClear）**
  在用户遭遇弱网环境时，画面会尽可能保持清晰，但可能会更容易出现卡顿。

ControlMode

controlMode 参数选择 **TRTCQosControlModeServer** 即可，TRTCQosControlModeClient 是腾讯云研发团队做内部调试用的，请勿关注。

常见的误区

1. 分辨率越高越好?

较高的分辨率也需要较高的码率来支撑，如果分辨率选择1280 x 720，但码率却指定为200kbps，画面就会有大量的马赛克。推荐参考 分辨率码率参照表 进行设置。

2. 帧率越高越好?

由于摄像头采集的画面是曝光阶段中所有现实物体的完整映射，所以并不是帧率越高，感官就越流畅，这一点跟游戏里的 FPS 是不一样的。恰恰相反，帧率过高，会拉低每帧画面的画质，也会减少摄像机的曝光时间，效果可能会更差。

3. 码率越高越好?

较高的码率也需要较高的分辨率来匹配，对于320 x 240这样分辨率，1000kbps的码率就很浪费了，推荐参考 分辨率码率参照表 进行设置。

4. 用 Wi-Fi 的时候就可以设置很高的分辨率和码率

并不是说 Wi-Fi 的网速是恒定不变的，如果离无线路由器较远， 或者路由器信道被占用，可能网速还不如4G。
针对这种情况， TRTC SDK 提供了测速功能，可以在视频通话前先进行测速，根据打分值来确定网络好坏。

Flutter

最近更新时间：2024-02-01 21:16:22

内容介绍

在 TRTCCloud 中，您可以通过以下方式调整画质：

- TRTCCloud.enterRoom 中的 TRTCAppScene 参数：用于选择您的应用场景。
- TRTCCloud.setVideoEncoderParam：用于设置编码参数。
- TRTCCloud.setNetworkQosParam：用于设置网络调控策略。

本文主要介绍如何配置上述参数，使 TRTC SDK 的画质效果符合您的项目需要。
您也可以参考以下 Demo：

- Flutter：SetVideoQualityPage.dart

支持的平台

iOS	Android	Mac OS	Windows	Web	Electron	Flutter
✓	✓	✓	✓	✓	✓	✓

Web端设定画面质量的详细操作，请参见 设定指引 。

房间场景

场景类型	场景介绍
TRTC_APP_SCENE_VIDEOCALL	在视频通话场景中，支持720p和1080p高清图像质量。单个房间可容纳最多300名同时在线用户，最多50人同时发言。
TRTC_APP_SCENE_LIVE	在互动视频直播场景中，麦克风可以顺畅地打开/关闭，无需等待切换，主播延迟低至300毫秒以下。支持数十万并发观众用户的直播，播放延迟降至1000毫秒。 **注意**：在此场景下，您必须使用 TRTCParams 中的 role 字段指定当前用户的角色。
TRTC_APP_SCENE_AUDIOCALL	在音频通话场景中，支持48kHz双声道音频通话。单个房间可容纳最多300名同时在线用户，最多50人同时发言。
TRTC_APP_SCENE_VOICE_CHATROOM	在互动音频直播场景中，麦克风可以顺畅地打开/关闭，无需等待切换，主播延迟低至300毫秒以下。支持数十万并发观众用户的直播，播放延迟降至1000毫秒。 **注意**：在此场景下，您必须使用 TRTCParams 中的 role 字段指定当前用户的角色。

TRTCVideoEncParam

推荐的配置

应用场景	videoResolution	videoFps	videoBitrate
视频通话（手机）	640x360	15	550kbps
视频会议（主画面 @ Mac Win）	1280x720	15	1200kbps
视频会议（主画面 @ 手机）	640x360	15	900kbps
视频会议（小画面）	320x180	15	250kbps
在线教育（老师 @ Mac Win）	960x540	15	850kbps

版权所有：腾讯云计算（北京）有限责任公司

在线教育（老师 @ iPad）	640x360	15	550kbps
在线教育（学生）	320x180	15	250kbps

各字段详解

- **(int) videoResolution**
  编码分辨率（TRTCCloudDef.TRTC_VIDEO_RESOLUTION_）），例如640 x 360是指编码出的画面的宽（像素）x 高（像素），我们在 TRTCVideoResolution 枚举定义里只定义了宽 >= 高的横屏（Landscape）分辨率，如果想要使用竖屏分辨率，需要将 resMode 设置为 Portrait。

  > **注意：**
  > 由于很多硬件编解码器只支持能被 16 整除的像素宽度，所以 SDK 实际编码出的分辨率并不一定完全按照参数自定，而是会自动进行 16 整除修正。例如 640 x 360 的分辨率，在 SDK 内部有可能会适配为 640 x 368。

- **(int) videoResolutionMode**
  指横屏或竖屏分辨率（TRTCCloudDef.TRTC_VIDEO_RESOLUTION_MODE_），由于 TRTCVideoResolution 中只定义了横屏分辨率，如果您希望使用 360 x 640 这样的竖屏分辨率，就需要指定 resMode 为 TRTCVideoResolutionModePortrait。一般 PC 和 Mac 采用横屏（Landscape）分辨率，手机采用竖屏（Portrait）分辨率。
- **(int) videoFps**
  帧率（FPS），也就是每秒钟要编码多少帧画面。推荐设置为 15 FPS，这样既能保证画面足够流畅，又不会因为每秒帧数太多而拉低单幅画面的清晰度。如果您对流畅度要求比较高，可以设置为 20 FPS 或 25 FPS。但请不要设置 25 FPS 以上的数值，因为电影的常规帧率也只有 24 FPS。
- **(int) videoBitrate**
  视频码率（Bitrate），即每秒钟编码器输出多少 Kbit 的编码后的二进制数据。如果您将 videoBitrate 设置为 800kbps，那么每秒钟编码器会产生 800kbit 的视频数据，这些数据如果存储成一个文件，那么文件大小就是 800kbit，也就是100KB，也就是 0.1M。

视频码率并不是越高越好，它跟分辨率之间要有比较恰当的映射关系，如下表所示。

分辨率码率参照表

分辨率定义	宽高比	建议码率（VideoCall）	建议码率（LIVE）
TRTCVideoResolution_120_120	1:1	80kbps	120kbps
TRTCVideoResolution_160_160	1:1	100kbps	150kbps
TRTCVideoResolution_270_270	1:1	200kbps	300kbps
TRTCVideoResolution_480_480	1:1	350kbps	525kbps
TRTCVideoResolution_160_120	4:3	100kbps	150kbps
TRTCVideoResolution_240_180	4:3	150kbps	225kbps
TRTCVideoResolution_280_210	4:3	200kbps	300kbps
TRTCVideoResolution_320_240	4:3	250kbps	375kbps
TRTCVideoResolution_400_300	4:3	300kbps	450kbps
TRTCVideoResolution_480_360	4:3	400kbps	600kbps
TRTCVideoResolution_640_480	4:3	600kbps	900kbps

版权所有：腾讯云计算（北京）有限责任公司

TRTCVideoResolution_960_720	4:3	1000kbps	1500kbps
TRTCVideoResolution_160_90	16:9	150kbps	250kbps
TRTCVideoResolution_256_144	16:9	200kbps	300kbps
TRTCVideoResolution_320_180	16:9	250kbps	400kbps
TRTCVideoResolution_480_270	16:9	350kbps	550kbps
TRTCVideoResolution_640_360	16:9	550kbps	900kbps
TRTCVideoResolution_960_540	16:9	850kbps	1300kbps
TRTCVideoResolution_1280_720	16:9	1200kbps	1800kbps
TRTCVideoResolution_1920_1080	16:9	2000kbps	3000kbps

TRTCNetworkQosParam

QosPreference

在网络带宽比较充裕的情况下，清晰和流畅是可以兼顾的，但当用户的网络并不理想时，究竟是优先保证清晰还是优先保证流畅？您可以通过指定 TRTCNetworkQosParam 中的 preference 参数来做出选择。

- **流畅优先（TRTCVideoQosPreferenceSmooth）**
  在用户遭遇弱网环境时，画面会变得模糊，且会有较多马赛克，但可以保持流畅或轻微卡顿。
- **清晰优先（TRTCVideoQosPreferenceClear）**
  在用户遭遇弱网环境时，画面会尽可能保持清晰，但可能会更容易出现卡顿。

ControlMode

controlMode 参数选择 **TRTCQosControlModeServer** 即可，TRTCQosControlModeClient 是腾讯云研发团队做内部调试用的，请勿关注。

常见的误区

1. **分辨率越高越好?**
   较高的分辨率也需要较高的码率来支撑，如果分辨率选择1280 x 720，但码率却指定为200kbps，画面就会有大量的马赛克。推荐参考 分辨率码率参照表 进行设置。
2. **帧率越高越好?**
   由于摄像头采集的画面是曝光阶段中所有现实物体的完整映射，所以并不是帧率越高，感官就越流畅，这一点跟游戏里的FPS是不一样的。恰恰相反，帧率过高，会拉低每帧画面的画质，也会减少摄像机的曝光时间，效果可能会更差。
3. **码率越高越好?**
   较高的码率也需要较高的分辨率来匹配，对于320 x 240这样分辨率，1000kbps的码率就很浪费了，推荐参考 分辨率码率参照表 进行设置。
4. **用 Wi-Fi 的时候就可以设置很高的分辨率和码率**
   并不是说 Wi-Fi 的网速是恒定不变的，如果离无线路由器较远， 或者路由器信道被占用，可能网速还不如4G。
   针对这种情况， TRTC SDK 提供了测速功能，可以在视频通话前先进行测速，根据打分值来确定网络好坏。

调整画面方向
Android&iOS&Windows&Mac

最近更新时间：2023-09-20 09:53:11

内容介绍

跟手机直播千篇一律的竖屏体验不同，实时音视频（TRTC）需要兼顾横屏和竖屏两种场景，因此就会有很多横竖屏的处理逻辑需要去应对，本文主要介绍：

- 如何实现竖屏模式，例如：微信的视频通话就是一个典型的竖屏体验模式。
- 如何实现横屏模式，例如：多人音视频房间 App（类似小鱼易连）往往都是采用横屏模式。
- 如何自定义控制本地画面和远程画面的旋转方向和填充模式。

平台支持

iOS	Android	Mac OS	Windows	Electron	微信小程序	Web 端
✓	✓	✓	✓	✓	×	×

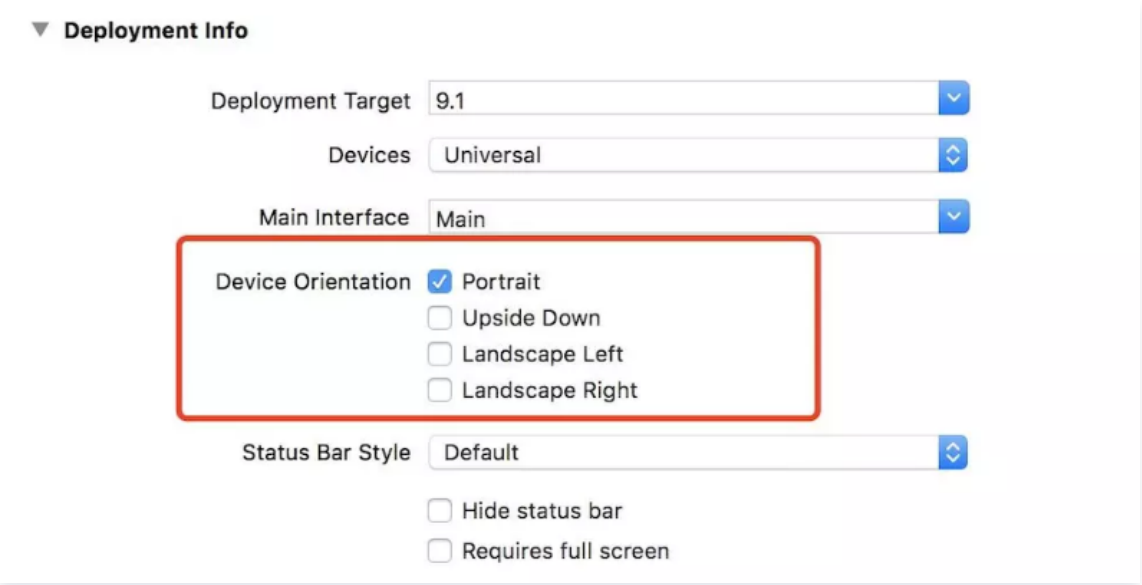
竖屏模式

如果要实现类似微信视频通话的体验模式，需要做两项工作：

1. 配置 App 的 UI 界面为竖屏

iOS 平台

可以直接在 XCode 的 **General** > **Deployment Info** > **Device Orientation** 中进行设置：



您也可以通过实现 Appdelegate 中的 `supportedInterfaceOrientationsForWindow` 方法来达到相同目标：

```
- (UIInterfaceOrientationMask)application:(UIApplication *)application
    supportedInterfaceOrientationsForWindow:(UIWindow *)window
{

    return  UIInterfaceOrientationMaskPortrait ;

}
```

> **说明**
> CSDN 上有一篇文章 iOS横竖屏旋转及其基本适配方法，详细介绍了 iOS 平台中关于屏幕方向的一些开发经验。

Android 平台

通过指定 activity 的 `screenOrientation` 属性为 portrait，即可指定该界面为竖屏模式：

```
<activity android:name=".trtc.TRTCMainActivity"  android:launchMode="singleTask"
android:windowSoftInputMode="adjustPan"
          android:screenOrientation="portrait" />
```

2. 配置 SDK 使用竖屏分辨率

在使用 TRTCCloud 的 setVideoEncoderParam 接口设置视频编码参数时，将 resMode 指定为 `TRTCVideoResolutionModePortrait` 即可。
示例代码如下：

iOS

```
TRTCVideoEncParam* encParam = [TRTCVideoEncParam new];
encParam.videoResolution = TRTCVideoResolution_640_360;
encParam.videoBitrate = 600;
encParam.videoFps = 15;
encParam.resMode = TRTCVideoResolutionModePortrait; //设置分辨率模式为竖屏模式

[trtc setVideoEncoderParam: encParam];
```

Android

```
TRTCCloudDef.TRTCVideoEncParam encParam = new TRTCCloudDef.TRTCVideoEncParam();
encParam.videoResolution = TRTCCloudDef.TRTC_VIDEO_RESOLUTION_640_360;
encParam.videoBitrate = 600;
encParam.videoFps = 15;
encParam.videoResolutionMode = TRTCCloudDef.TRTC_VIDEO_RESOLUTION_MODE_PORTRAIT; //设置分辨率
模式为竖屏模式
trtc.setVideoEncoderParam(encParam);
```

横屏模式

如果希望 App 是横屏体验，那么您需要做的工作跟竖屏模式类似，只是将第一步和第二步中的参数都进行相应的调整即可。
尤其是 第二步 中，TRTCVideoEncParam 中的 resMode 值：

- 在 iOS 平台中应该指定为 `TRTCVideoResolutionModeLandscape` 。
- 在 Android 平台中应该指定为 `TRTC_VIDEO_RESOLUTION_MODE_LANDSCAPE` 。

自定义控制

TRTC SDK 本身提供了大量的接口函数可以操控本地和远程画面的旋转方向和填充模式：

接口函数	功能作用	备注说明
setLocalViewRotation	本地预览画面的顺时针旋转角度	支持顺时针旋转90度、180度和270度三个方向
setLocalViewFillMode	本地预览画面的填充模式	是裁剪还是留黑边
setRemoteViewRotation	远端视频画面的顺时针旋转角度	支持顺时针旋转90度、180度和270度三个方向
setRemoteViewFillMode	远端视频画面的填充模式	是裁剪还是留黑边
setVideoEncoderRotation	设置编码器输出的画面顺时针旋转角度	支持顺时针旋转90度、180度和270度三个方向

GSensorMode

考虑到画面旋转牵扯到录制和 CDN 旁路直播的各种适配问题， TRTC SDK 仅提供了一种简单的重力感应自适应功能，您可以通过 TRTCCloud 的 `setGSensorMode` 接口来开启。
该功能支持90度、180度、270度旋转的自适应，也就是当用户自己的手机旋转时，对方看到的画面朝向还是会保持不变。而且这种自适应是基于对编码器的方向调整而实现的，因此录制出的视频，以及小程序和 H5 端看到的视频画面也能做到保持原方向不变。

> **注意**
> 重力感应自适应的另一种实现方案是在每一帧视频信息里都带上当前视频的重力朝向，然后在远程用户那里自适应的调整渲染方向，但这种方案需要引入额外的转码资源才能解决录制出的视频朝向跟期望的视频朝向保持一致的问题，因此并不推荐。

Electron

最近更新时间：2023-07-12 18:07:42

实时音视频（TRTC）支持自定义控制本地画面和远程画面的旋转方向和填充模式。

自定义控制本地画面

可以通过调用 setLocalRenderParams 设置本地渲染参数。

```
import TRTCCloud, {
  TRTCRenderParams, TRTCVideoRotation, TRTCVideoFillMode,
  TRTCVideoMirrorType
} from 'trtc-electron-sdk';
const trtcCloud = new TRTCCloud();
const param = new TRTCRenderParams(
  TRTCVideoRotation.TRTCVideoRotation90,
  TRTCVideoFillMode.TRTCVideoFillMode_Fill,
  TRTCVideoMirrorType.TRTCVideoMirrorType_Enable
);

trtcCloud.setLocalRenderParams(param);
const localUserDom = document.querySelector('local-user');
trtcCloud.startLocalPreview(localUserDom);
```

自定义控制远程画面

可以通过调用 setRemoteRenderParams 设置远端渲染参数。

```
import TRTCCloud, {
  TRTCRenderParams, TRTCVideoRotation, TRTCVideoFillMode,
  TRTCVideoMirrorType, TRTCVideoStreamType
} from 'trtc-electron-sdk';
const trtcCloud = new TRTCCloud();
const param = new TRTCRenderParams(
  TRTCVideoRotation.TRTCVideoRotation180,
  TRTCVideoFillMode.TRTCVideoFillMode_Fill,
  TRTCVideoMirrorType.TRTCVideoMirrorType_Disable
);

const remoteUserId = 'remoteUser';
trtcCloud.setRemoteRenderParams(remoteUserId, TRTCVideoStreamType.TRTCVideoStreamTypeBig,
param);
const remoteUserDom = document.querySelector('remote-user');
trtcCloud.startRemoteView(remoteUserId, remoteUserDom,
TRTCVideoStreamType.TRTCVideoStreamTypeBig);
```

版权所有：腾讯云计算（北京）有限责任公司

Web

最近更新时间：2023-08-16 16:41:01

本文主要介绍如何使用 TRTC Web SDK 控制视频渲染时的镜像、填充模式。

> **说明：**
> - 本教程基于5.x TRTC Web SDK 实现，若您使用4.x版本 SDK，可参考stream.play 的 objectFit 属性 。
> - 5.x与4.x版本的区别 。

镜像

通过 trtc.startLocalVideo({ option: { mirror: true } }) 和 trtc.startRemoteVideo({ option: { mirror: true }}) 的方式来控制本地和远端视频的渲染镜像效果。

```
// 本地摄像头渲染镜像，默认为 true
await trtc.startLocalVideo({ option: { mirror: true }});

// 动态更新参数
await trtc.updateLocalVideo({ option: { mirror: false }});

trtc.on(TRTC.EVENT.REMOTE_VIDEO_AVAILABLE, async ({ userId, streamType }) => {
  await trtc.startRemoteVideo({
    userId,
    streamType,
    // 您需在 DOM 中提前放置视频容器，建议以 `${userId}_${streamType}` 作为 element id。
    view: `${userId}_${streamType}`,
    // 镜像播放远端视频，默认为 false
    option: { mirror: true }
    });

  // 动态更新参数
  await trtc.updateRemoteVideo({ userId, streamType, option: { mirror: false }})
});
```

> **注意：**
> 该镜像效果只是针对渲染时做处理，实际编码或解码出的画面是没有镜像效果的。您可以通过 canvas 自定义采集 的方式，对 canvas 进行翻转，从而实现编码镜像的效果。

填充模式

通过 trtc.startLocalVideo({ option: { fillMode: 'cover' } }) 和 trtc.startRemoteVideo({ option: { fillMode: 'cover' }}) 的方式来控制本地和远端视频的渲染填充模式。

不同参数的含义：

- `contain` 保留宽高比，在目标容器中完整显示画面，若宽高比与目标容器不匹配，则会以黑边填充。建议播放屏幕分享使用该参数。
- `cover` 默认值，保留宽高比，在目标容器中显示，若宽高比与目标容器不匹配，则画面则会被裁剪，以填满整个目标容器。
- `fill` 不保留宽高比，在目标容器中显示，若宽高比与目标容器不匹配，则画面会被拉伸，以填满整个模板容器。

参考：CSS object-fit 属性 。

版权所有：腾讯云计算（北京）有限责任公司

```
// 本地摄像头填充模式，默认为 cover
await trtc.startLocalVideo({ option: { fillMode: 'cover' }});

// 动态更新参数
await trtc.updateLocalVideo({ option: { fillMode: 'contain' }});

trtc.on(TRTC.EVENT.REMOTE_VIDEO_AVAILABLE, async ({ userId, streamType }) => {
  await trtc.startRemoteVideo({
    userId,
    streamType,
    // 您需在 DOM 中提前放置视频容器，建议以 `${userId}_${streamType}` 作为 element id。
    view: `${userId}_${streamType}`,
    option: { fillMode: 'contain' }
    });

  // 动态更新参数
  await trtc.updateRemoteVideo({ userId, streamType, option: { fillMode: 'cover' }})
});
```

Flutter

最近更新时间：2024-02-01 21:16:22

内容介绍

跟手机直播千篇一律的竖屏体验不同，实时音视频（TRTC）需要兼顾横屏和竖屏两种场景，因此就会有很多横竖屏的处理逻辑需要去应对，本文主要介绍：

- 如何实现竖屏模式，例如：微信的视频通话就是一个典型的竖屏体验模式。
- 如何实现横屏模式，例如：多人音视频房间 App（类似小鱼易连）往往都是采用横屏模式。
- 如何自定义控制本地画面和远程画面的旋转方向和填充模式。

平台支持

iOS	Android	Mac OS	Windows	Electron	Web 端
✓	✓	✓	✓	✓	×

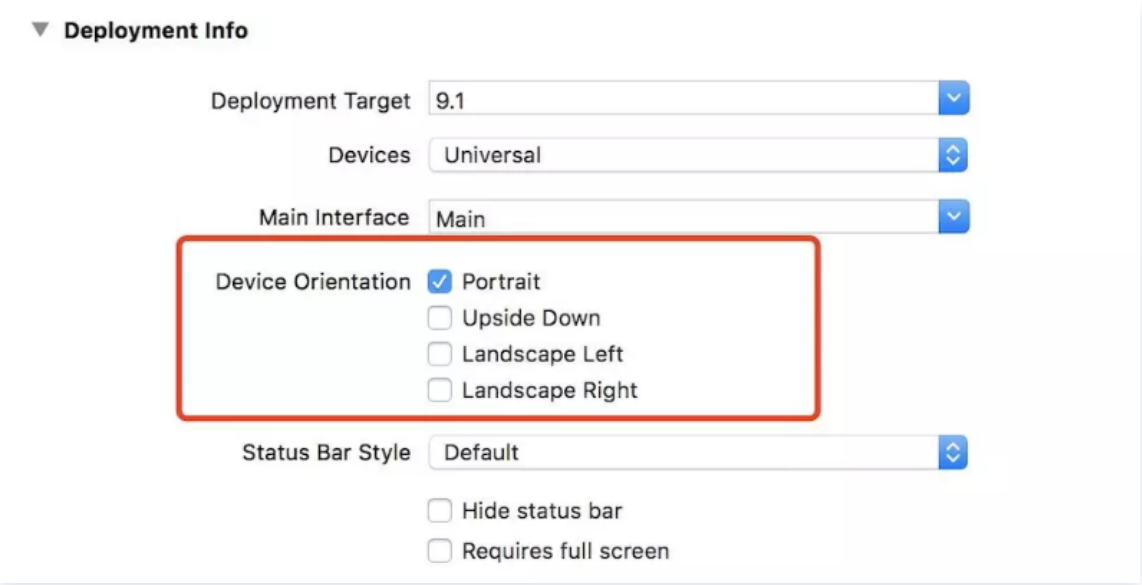
竖屏模式

如果要实现类似微信视频通话的体验模式，需要做两项工作：

1. 配置 App 的 UI 界面为竖屏

iOS 平台

可以直接在 XCode 的 **General** > **Deployment Info** > **Device Orientation** 中进行设置：



Android 平台

通过指定 activity 的 `screenOrientation` 属性为 portrait，即可指定该界面为竖屏模式：

```
<activity android:name=".trtc.TRTCMainActivity"  android:launchMode="singleTask"
android:windowSoftInputMode="adjustPan"
          android:screenOrientation="portrait" />
```

2. 配置 SDK 使用竖屏分辨率

在使用 TRTCCloud 的 setVideoEncoderParam 设置视频编码参数时，将 videoResolutionMode 指定为 `TRTC_VIDEO_RESOLUTION_MODE_PORTRAIT` 即可。
示例代码如下：

```
trtcCloud.setVideoEncoderParam(TRTCVideoEncParam(
    videoFps: 15,
    videoResolution: TRTCCloudDef.TRTC_VIDEO_RESOLUTION_640_360,
    videoBitrate: 600,
    videoResolutionMode:
        TRTCCloudDef.TRTC_VIDEO_RESOLUTION_MODE_PORTRAIT));
```

横屏模式

如果希望 App 是横屏体验，那么您需要做的工作跟竖屏模式类似，只是将第一步和第二步中的参数都进行相应的调整即可。
尤其是 第二步 中，TRTCVideoEncParam 中的 videoResolutionMode 值：`TRTC_VIDEO_RESOLUTION_MODE_LANDSCAPE` 。

自定义控制

TRTC SDK 本身提供了接口函数可以操控本地和远程画面的旋转方向和填充模式：

接口函数	功能作用	备注说明
setVideoEncoderRotation	设置编码器输出的画面顺时针旋转角度	目前仅支持0度和180度

GSensorMode

考虑到画面旋转牵扯到录制和 CDN 旁路直播的各种适配问题， TRTC SDK 仅提供了一种简单的重力感应自适应功能，您可以通过 TRTCCloud 的 `setGSensorMode` 接口来开启。
该功能支持90度、180度、270度旋转的自适应，也就是当用户自己的手机旋转时，对方看到的画面朝向还是会保持不变。而且这种自适应是基于对编码器的方向调整而实现的，因此录制出的视频，以及小程序和 H5 端看到的视频画面也能做到保持原方向不变。

> **注意：**
> 重力感应自适应的另一种实现方案是在每一帧视频信息里都带上当前视频的重力朝向，然后在远程用户那里自适应的调整渲染方向，但这种方案需要引入额外的转码资源才能解决录制出的视频朝向跟期望的视频朝向保持一致的问题，因此并不推荐。

开启屏幕分享
iOS

最近更新时间：2023-11-08 14:37:11

腾讯云 TRTC 在 iOS 平台下支持两种不同的屏幕分享方案：

- **应用内分享**
  即只能分享当前 App 的画面，该特性需要 iOS 13 及以上版本的操作系统才能支持。由于无法分享当前 App 之外的屏幕内容，因此适用于对隐私保护要求高的场景。
- **跨应用分享**
  基于苹果的 Replaykit 方案，能够分享整个系统的屏幕内容，但需要当前 App 额外提供一个 Extension 扩展组件，因此对接步骤也相对应用内分享要多一点。

支持的平台

iOS	Android	Mac OS	Windows	Electron	微信小程序	Chrome 浏览器
✓	✓	✓	✓	✓	×	✓

应用内分享

应用内分享的方案非常简单，只需要调用 TRTC SDK 提供的接口 startScreenCaptureInApp 并传入编码参数 `TRTCVideoEncParam` 即可。该参数可以设置为 nil，此时 SDK 会沿用开始屏幕分享之前的编码参数。
我们推荐的用于 iOS 屏幕分享的编码参数是：

参数项	参数名称	常规推荐值	文字教学场景
分辨率	videoResolution	1280 × 720	1920 × 1080
帧率	videoFps	10 FPS	8 FPS
最高码率	videoBitrate	1600 kbps	2000 kbps
分辨率自适应	enableAdjustRes	NO	NO

- 由于屏幕分享的内容一般不会剧烈变动，所以设置较高的 FPS 并不经济，推荐10 FPS即可。
- 如果您要分享的屏幕内容包含大量文字，可以适当提高分辨率和码率设置。
- 最高码率（videoBitrate）是指画面在剧烈变化时的最高输出码率，如果屏幕内容变化较少，实际编码码率会比较低。

跨应用分享

iOS 系统上的跨应用屏幕分享，需要增加 **Broadcast Upload Extension** 录屏进程以配合主 App 进程进行推流。Extension 录屏进程由系统在需要录屏的时候创建，并负责接收系统采集到屏幕图像。因此需要：

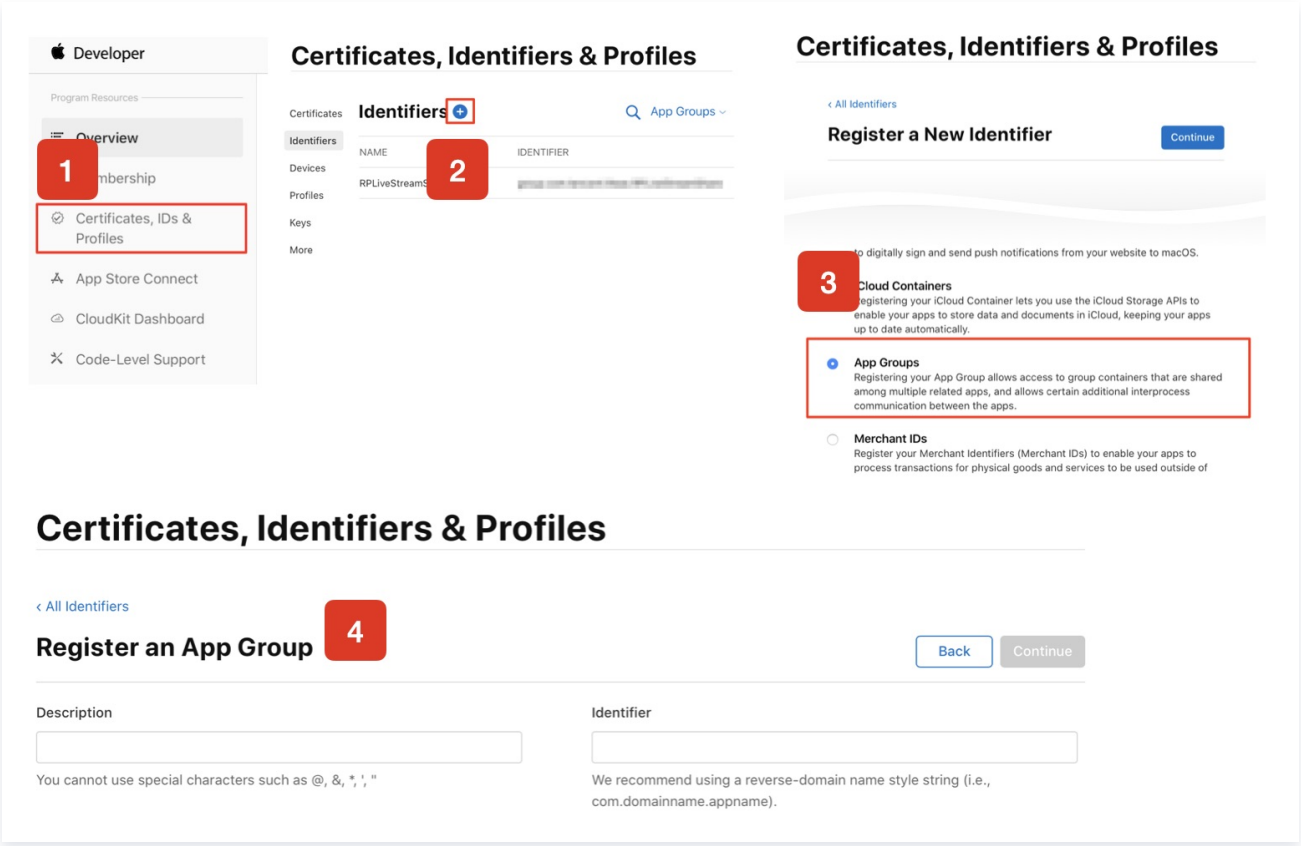
1. 创建 **App Group**，并在 XCode 中进行配置（可选）。这一步的目的是让 Extension 录屏进程可以同主 App 进程进行跨进程通信。
2. 在您的工程中，新建一个 **Broadcast Upload Extension** 的 Target，并在其中集成 SDK 压缩包中专门为扩展模块定制的 `TXLiteAVSDK_ReplayKitExt.framework`。
3. 对接主 App 端的接收逻辑，让主 App 等待来自 **Broadcast Upload Extension** 的录屏数据。

> **注意**
> 如果跳过步骤1，也就是不配置 App Group（接口传 nil），屏幕分享依然可以运行，但稳定性要打折扣，故虽然步骤较多，但请尽量配置正确的 App Group 以保障屏幕分享功能的稳定性。

步骤1：创建 App Group

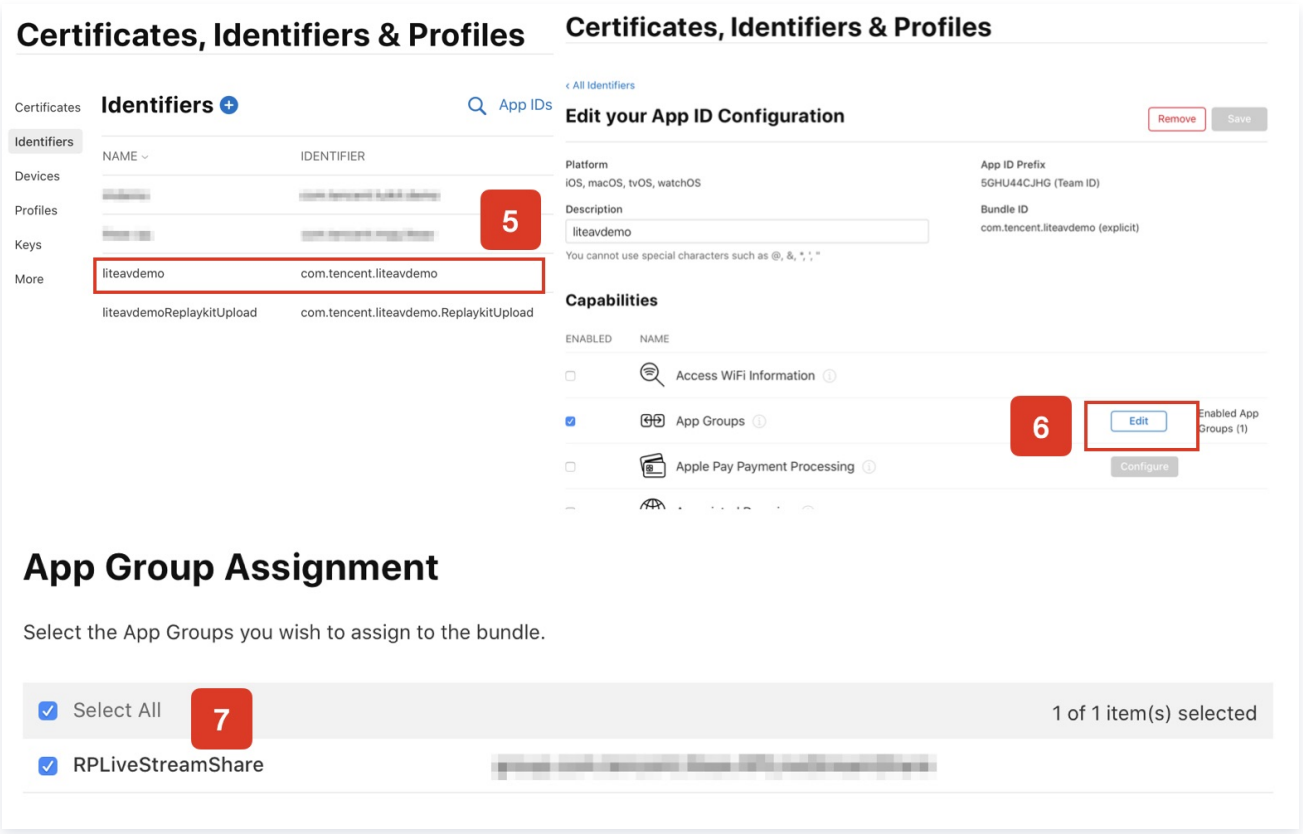
使用您的账号登录 https://developer.apple.com/ ，进行以下操作，**注意完成后需要重新下载对应的 Provisioning Profile**。

1. 单击 **Certificates, IDs & Profiles**。
2. 在右侧的界面中单击加号。
3. 选择 **App Groups**，单击 **Continue**。
4. 在弹出的表单中填写 Description 和 Identifier，其中 Identifier 需要传入接口中的对应的 AppGroup 参数。完成后单击 **Continue**。



5. 回到 Identifier 页面，左上边的菜单中选择 **App IDs**，然后单击您的 App ID（主 App 与 Extension 的 AppID 需要进行同样的配置）。
6. 选中 **App Groups** 并单击 **Edit**。

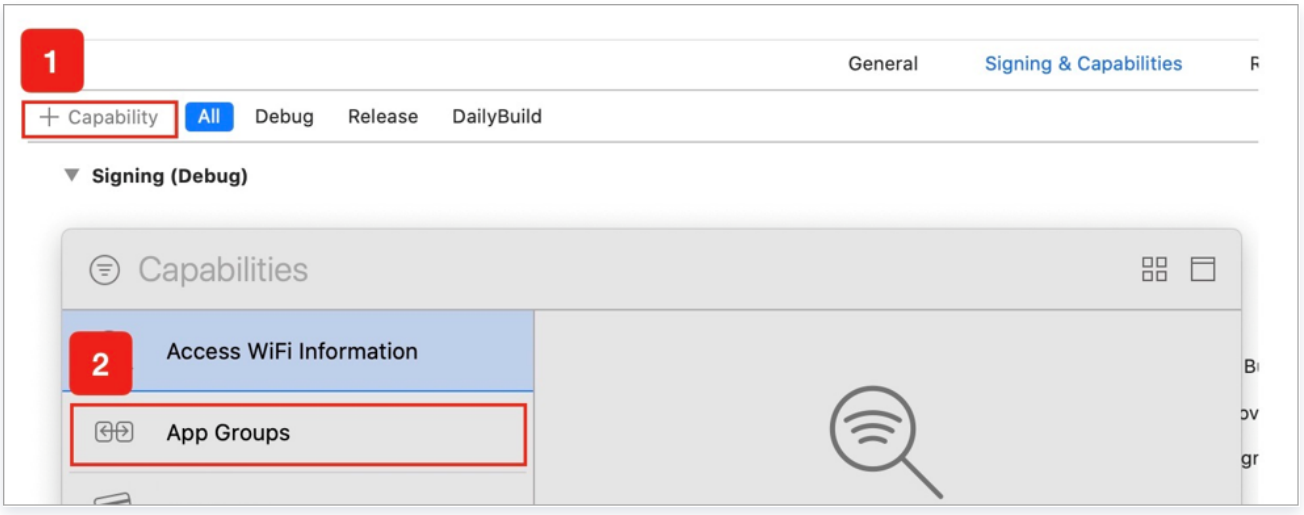
7. 在弹出的表单中选择您之前创建的 App Group，单击 **Continue** 返回编辑页，单击 **Save** 保存。



8. 重新下载 Provisioning Profile 并配置到 XCode 中。

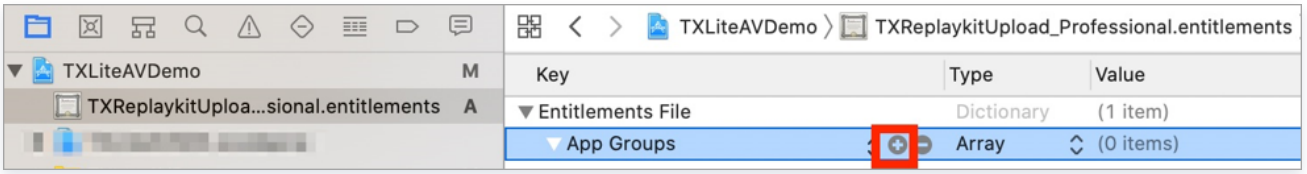
步骤2：创建 Broadcast Upload Extension

1. 在 Xcode 菜单依次单击 **File**、**New** 、**Target...**，选择 **Broadcast Upload Extension**。
2. 在弹出的对话框中填写相关信息，**不用**勾选 **Include UI Extension**，单击 **Finish** 完成创建。
3. 将下载到的 SDK 压缩包中的 TXLiteAVSDK_ReplayKitExt.framework 拖动到工程中，勾选刚创建的 Target。
4. 添加系统库依赖：找到录屏 target -> General -> Frameworks and Libraries，点击添加按钮添加系统库依赖：Accelerate.framework、VideoToolbox.framework、libc++.tbd。
5. 选中新增加的 Target，依次单击 **+ Capability**，双击 **App Groups**，如下图：



操作完成后，会在文件列表中生成一个名为 `Target.entitlements` 的文件，如下图所示，选中该文件并单击 + 号填写上述步骤中的

App Group 即可。



6. 选中主 App 的 Target，**并按照上述步骤对主 App 的 Target 做同样的处理。**
7. 在新创建的 Target 中，Xcode 会自动创建一个名为 "SampleHandler" 的类，用如下代码进行替换其中的.m文件。**需将代码中的 APPGROUP 改为上文中的创建的 App Group Identifier。**

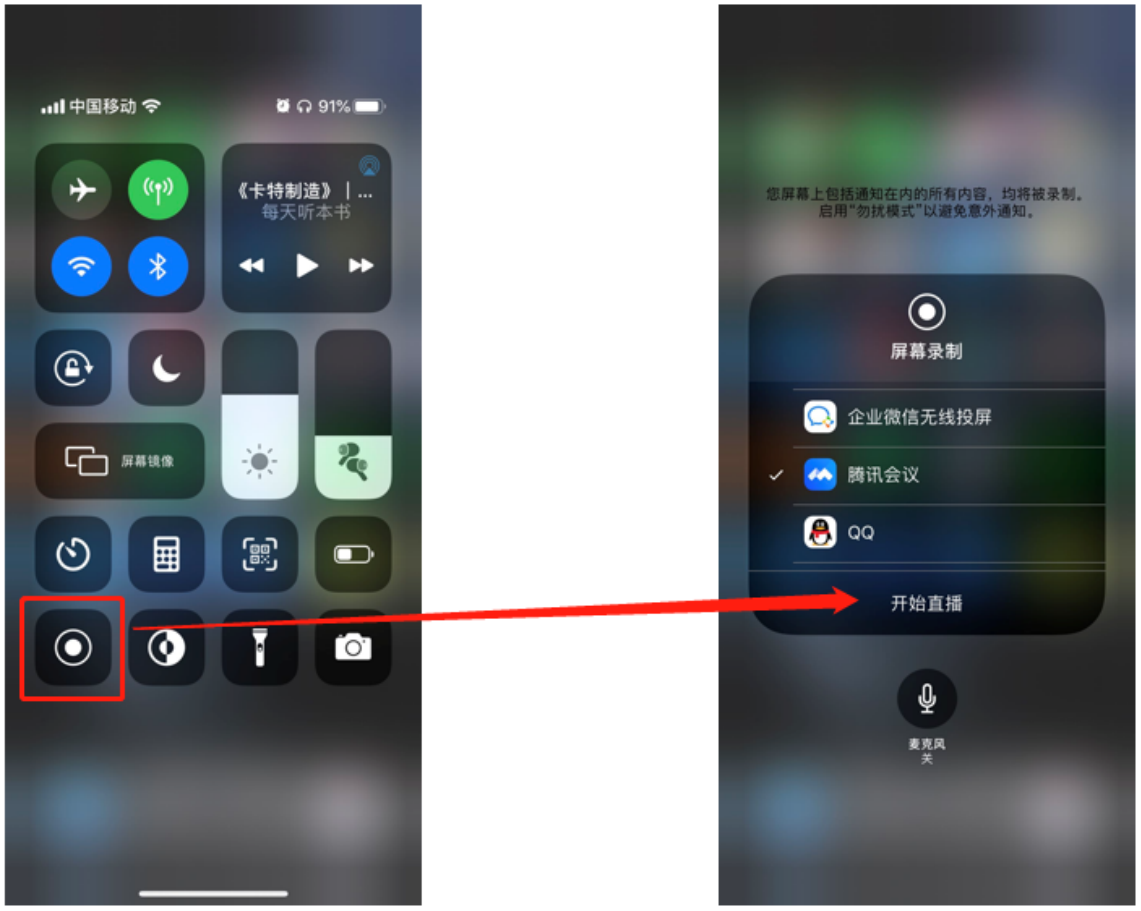
```
#import "SampleHandler.h"
@import TXLiteAVSDK_ReplayKitExt;

#define APPGROUP @"group.com.tencent.liteav.RPLiveStreamShare"

@interface SampleHandler() <TXReplayKitExtDelegate>

@end

@implementation SampleHandler
// 注意：此处的 APPGROUP 需要改成上文中的创建的 App Group Identifier。
- (void)broadcastStartedWithSetupInfo:(NSDictionary<NSString *,NSObject *> *)setupInfo {
    [[TXReplayKitExt sharedInstance] setupWithAppGroup:APPGROUP delegate:self];
}

- (void)broadcastPaused {
    // User has requested to pause the broadcast. Samples will stop being delivered.
}

- (void)broadcastResumed {
    // User has requested to resume the broadcast. Samples delivery will resume.
}

- (void)broadcastFinished {
    [[TXReplayKitExt sharedInstance] finishBroadcast];
    // User has requested to finish the broadcast.
}

#pragma mark - TXReplayKitExtDelegate
- (void)broadcastFinished:(TXReplayKitExt *)broadcast reason:(TXReplayKitExtReason)reason
{
    NSString *tip = @"";
    switch (reason) {
        case TXReplayKitExtReasonRequestedByMain:
            tip = @"屏幕共享已结束";
            break;
        case TXReplayKitExtReasonDisconnected:
            tip = @"应用断开";
            break;
        case TXReplayKitExtReasonVersionMismatch:
            tip = @"集成错误（SDK 版本号不相符合）";
            break;
```

```
    }

    NSError *error = [NSError errorWithDomain:NSStringFromClass(self.class)
                                         code:0
                                     userInfo:@{
                                         NSLocalizedFailureReasonErrorKey:tip
                                     }];
    [self finishBroadcastWithError:error];
}

- (void)processSampleBuffer:(CMSampleBufferRef)sampleBuffer withType:
(RPSampleBufferType)sampleBufferType {
    switch (sampleBufferType) {
        case RPSampleBufferTypeVideo:
            [[TXReplayKitExt sharedInstance] sendVideoSampleBuffer:sampleBuffer];
            break;
        case RPSampleBufferTypeAudioApp:
            // Handle audio sample buffer for app audio
            break;
        case RPSampleBufferTypeAudioMic:
            // Handle audio sample buffer for mic audio
            break;

        default:
            break;
    }
}
@end
```

步骤3：对接主 App 端的接收逻辑

按照如下步骤，对接主 App 端的接收逻辑。也就是在用户触发屏幕分享之前，要让主 App 处于“等待”状态，以便随时接收来自 Broadcast Upload Extension 进程的录屏数据。

1. 确保 TRTCCloud 已经关闭了摄像头采集，如果尚未关闭，请调用 stopLocalPreview 关闭摄像头采集。
2. 调用 startScreenCaptureByReplaykit:appGroup: 方法，并传入 步骤1 中设置的 AppGroup，让 SDK 进入“等待”状态。
3. 等待用户触发屏幕分享。如果不实现 步骤4 中的“触发按钮”，屏幕分享就需要用户在 iOS 系统的控制中心，通过长按录屏按钮来触发，这一操作步骤如下图所示：



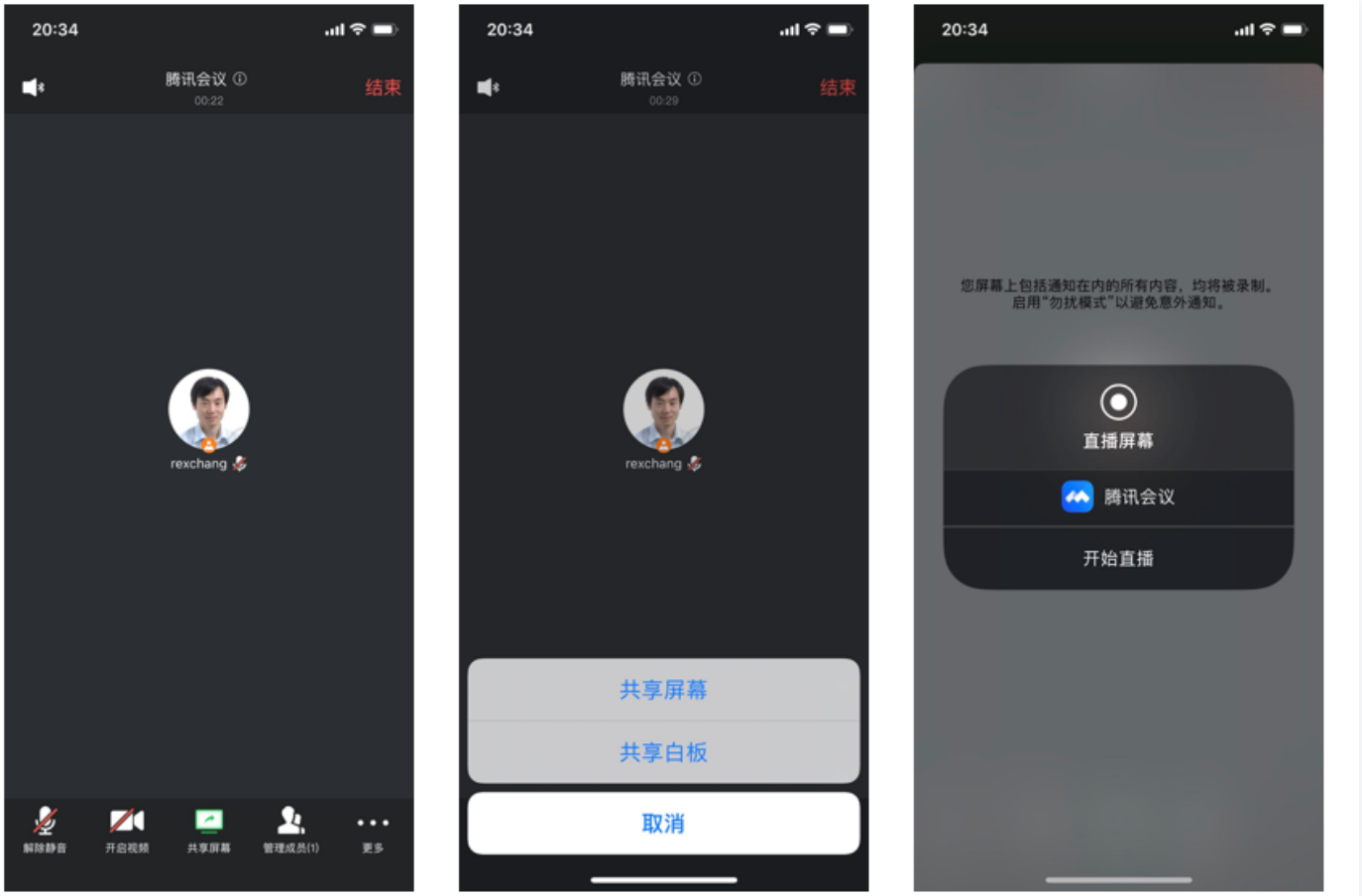
长按“录屏” 按钮　　点击“开始直播”

4. 通过调用 stopScreenCapture 接口可以随时中止屏幕分享。

```
// 开始屏幕分享，需要将 APPGROUP 替换为上述步骤中创建的 App Group Identifier。
- (void)startScreenCapture {
    TRTCVideoEncParam *videoEncConfig = [[TRTCVideoEncParam alloc] init];
    videoEncConfig.videoResolution = TRTCVideoResolution_1280_720;
    videoEncConfig.videoFps = 10;
    videoEncConfig.videoBitrate = 2000;
    //需要将 APPGROUP 替换为上述步骤中创建的 App Group Identifier:
    [[TRTCCloud sharedInstance] startScreenCaptureByReplaykit:videoEncConfig
                                                     appGroup:APPGROUP];
}

// 停止屏幕分享
- (void)stopScreenCapture {
    [[TRTCCloud sharedInstance] stopScreenCapture];
}

// 屏幕分享的启动事件通知，可以通过 TRTCCloudDelegate 进行接收
- (void)onScreenCaptureStarted {
    [self showTip:@"屏幕分享开始"];
}
```

步骤4：增加屏幕分享的触发按钮（可选）

截止到 步骤3 ，我们的屏幕分享还必须要用户从控制中心中长按录屏按钮来手动启动。您可通过下述方法实现类似腾讯会议的单击按钮即可触发的效果：



1. 在 Demo 中的 TRTCBroadcastExtensionLauncher 文件实现了唤起屏幕分享，找到并将其加入到您的工程中。
2. 在您的界面上放置一个按钮，并在按钮的响应函数中调用 `TRTCBroadcastExtensionLauncher` 中的 `launch` 函数，就可以唤起屏幕分享功能了。

```
// 自定义按钮响应方法
- (IBAction)onScreenButtonTapped:(id)sender {
    [TRTCBroadcastExtensionLauncher launch];
}
```

注意

- 苹果在 iOS 12.0 中增加了 `RPSystemBroadcastPickerView` 可以从应用中弹出启动器供用户确认启动屏幕分享，到目前为止，`RPSystemBroadcastPickerView` 尚不支持自定义界面，也没有官方的唤起方法。
- TRTCBroadcastExtensionLauncher 的原理就是遍历 `RPSystemBroadcastPickerView` 的子 View 寻找 UIButton 并触发了其点击事件。
- 但该方案不被苹果官方推荐，并可能在新一轮的系统更新中失效，因此 步骤4 只是一个可选方案，您需要自行承担风险来选用此方案。

观看屏幕分享

- **观看 Mac / Windows 屏幕分享**
  当房间里有一个 Mac / Windows 用户启动了屏幕分享，会通过辅流进行分享。房间里的其他用户会通过 TRTCCloudDelegate 中的 onUserSubStreamAvailable 事件获得这个通知。
  希望观看屏幕分享的用户可以通过 startRemoteSubStreamView 接口来启动渲染远端用户辅流画面。
- **观看 Android / iOS 屏幕分享**
  若用户通过 Android / iOS 进行屏幕分享，会通过主流进行分享。房间里的其他用户会通过 TRTCCloudDelegate 中的 onUserVideoAvailable 事件获得这个通知。
  希望观看屏幕分享的用户可以通过 startRemoteView 接口来启动渲染远端用户主流画面。

版权所有：腾讯云计算（北京）有限责任公司

Android

最近更新时间：2024-01-03 17:30:41

本文档主要介绍如何使用屏幕分享，目前一个 TRTC 音视频房间只能有一路屏幕分享。

调用指引

开启屏幕分享

步骤1：添加 Activity

在 manifest 文件中粘贴如下 activity（若项目代码中存在则不需要添加）。

```
<activity
    android:name="com.tencent.rtmp.video.TXScreenCapture$TXScreenCaptureAssistantActivity"
    android:theme="@android:style/Theme.Translucent"/>
```

步骤2： 启动屏幕分享

要开启 Android 端的屏幕分享，只需调用 `TRTCCloud` 中的 startScreenCapture() 接口即可。
通过设置 startScreenCapture() 中的首个参数 `encParams` ，您可以指定屏幕分享的编码质量。如果您指定 `encParams` 为 null，SDK 会自动使用之前设定的编码参数，我们推荐的参数设定如下：

参数项	参数名称	常规推荐值	文字教学场景
分辨率	videoResolution	1280 × 720	1920 × 1080
帧率	videoFps	10 FPS	8 FPS
最高码率	videoBitrate	1600 kbps	2000 kbps
分辨率自适应	enableAdjustRes	NO	NO

- 由于屏幕分享的内容一般不会剧烈变动，所以设置较高的 FPS 并不经济，推荐10 FPS即可。
- 如果您要分享的屏幕内容包含大量文字，可以适当提高分辨率和码率设置。
- 最高码率（videoBitrate）是指画面在剧烈变化时的最高输出码率，如果屏幕内容变化较少，实际编码码率会比较低。

步骤3：弹出悬浮窗防止应用被强杀（可选）

从 Android 7.0 系统开始，切入到后台运行的普通 App 进程，但凡有 CPU 活动，都很容易会被系统强杀掉。 所以当 App 在切入到后台默默进行屏幕分享时，通过弹出悬浮窗的方案，可以避免被系统强杀掉。 同时，在手机屏幕上显示悬浮窗也有利于告知用户当前正在做屏幕分享，避免用户泄露个人隐私。
要弹出悬浮窗，您只需要参考示例代码 FloatingView.java 中的实现即可：

```
public void showView(View view, int width, int height) {
        mWindowManager = (WindowManager) mContext.getSystemService(Context.WINDOW_SERVICE);
        int type = WindowManager.LayoutParams.TYPE_TOAST;
        //TYPE_TOAST仅适用于4.4+系统，假如要支持更低版本使用TYPE_SYSTEM_ALERT(需要在manifest中声明权
限)
        //7.1（包含）及以上系统对TYPE_TOAST做了限制
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            type = WindowManager.LayoutParams.TYPE_APPLICATION_OVERLAY;
        } else if (Build.VERSION.SDK_INT > Build.VERSION_CODES.N) {
```

```
            type = WindowManager.LayoutParams.TYPE_PHONE;
        }
        mLayoutParams = new WindowManager.LayoutParams(type);
        mLayoutParams.flags = WindowManager.LayoutParams.FLAG_NOT_FOCUSABLE;
        mLayoutParams.flags |= WindowManager.LayoutParams.FLAG_WATCH_OUTSIDE_TOUCH;
        mLayoutParams.width = width;
        mLayoutParams.height = height;
        mLayoutParams.format = PixelFormat.TRANSLUCENT;
        mWindowManager.addView(view, mLayoutParams);
}
```

观看屏幕分享

- **观看 Mac / Windows 屏幕分享**
  当房间里有一个 Mac / Windows 用户启动了屏幕分享，会通过辅流进行分享。房间里的其他用户会通过 TRTCCloudListener 中的 onUserSubStreamAvailable 事件获得这个通知。
- **观看 Android / iOS 屏幕分享**
  若用户通过 Android / iOS 进行屏幕分享，会通过主流进行分享。房间里的其他用户会通过 TRTCCloudListener 中的 onUserVideoAvailable 事件获得这个通知。

观看屏幕分享的用户可以通过 startRemoteView 接口来启动渲染远端用户主流画面。

常见问题

1、提示错误码-3343是什么问题?

如果使用屏幕分享过程中接收到 `-3343` 错误码，是因为网络连接问题导致的报错，需检查用户网络。

2、Android 10以上版本使用 LiteAVSDK 录屏/屏幕共享功能时 crash 是什么问题?

由于 Android 系统隐私策略的更改，在 Android 10 及以上版本 App 若使用录屏等功能需要在前台 Service 中进行，否则录屏/屏幕共享时系统将报错或 App 被系统终止。如果您的项目 targetSdkVersion 设置大于 29，使用录屏/屏幕共享时，您可以参考如下步骤进行处理。
第一步，创建一个 Service ，并绑定一个 Notification 使其作为前台 Service。

```
public class MediaService extends Service {
   private final String NOTIFICATION_CHANNEL_ID="com.tencent.trtc.apiexample.MediaService";
   private final String
NOTIFICATION_CHANNEL_NAME="com.tencent.trtc.apiexample.channel_name";
   private final String
NOTIFICATION_CHANNEL_DESC="com.tencent.trtc.apiexample.channel_desc";
    public MediaService() {
    }

    @Override
    public void onCreate() {
        super.onCreate();
        startNotification();
    }

    public void startNotification() {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            //Call Start foreground with notification
            Intent notificationIntent = new Intent(this, MediaService.class);
            PendingIntent pendingIntent;
```

```
            if (android.os.Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
                pendingIntent = PendingIntent.getActivity(this, 0, notificationIntent,
PendingIntent.FLAG_IMMUTABLE);
            } else {
                pendingIntent = PendingIntent.getActivity(this, 0, notificationIntent, 0);
            }
            NotificationCompat.Builder notificationBuilder = new
NotificationCompat.Builder(this, NOTIFICATION_CHANNEL_ID)
                    .setLargeIcon(BitmapFactory.decodeResource(getResources(),
R.drawable.ic_launcher_foreground))
                    .setSmallIcon(R.drawable.ic_launcher_foreground)
                    .setContentTitle("Starting Service")
                    .setContentText("Starting monitoring service")
                    .setContentIntent(pendingIntent);
            Notification notification = notificationBuilder.build();
            NotificationChannel channel = new NotificationChannel(NOTIFICATION_CHANNEL_ID,
NOTIFICATION_CHANNEL_NAME, NotificationManager.IMPORTANCE_DEFAULT);
            channel.setDescription(NOTIFICATION_CHANNEL_DESC);
            NotificationManager notificationManager = (NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);
            notificationManager.createNotificationChannel(channel);
            startForeground(1, notification); //必须使用此方法显示通知，不能使用
notificationManager.notify，否则还是会报上面的错误
        }
    }

    @Override
    public IBinder onBind(Intent intent) {
        throw new UnsupportedOperationException("Not yet implemented");
    }
}
```

第二步，在 AndroidManifest.xml 中配置加入以下权限：

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

指定 Service 的 android:foregroundServiceType="mediaProjection"。

```
<service
    android:name=".MediaService"
    android:enabled="true"
    android:foregroundServiceType="mediaProjection"
    android:exported="true"/>
```

第三步：在录屏前启动 Service。

```
public class TRTCApplication extends Application {
    @Override
    protected void attachBaseContext(Context base) {
        super.attachBaseContext(base);
        MultiDex.install(this);
        startService(new Intent(this,MediaService.class));
```

```
        }
    }
```

Mac

最近更新时间：2023-09-19 14:58:13

腾讯云 TRTC 支持屏幕分享功能，Mac 平台下的屏幕分享支持主路分享和辅路分享两种方案：

- **辅路分享**
  在 TRTC 中，我们可以单独为屏幕分享开启一路上行的视频流，并称之为“辅路（substream）”。辅路分享即主播同时上行摄像头画面和屏幕画面两路画面。这是腾讯会议的使用方案，您可以在调用 `startScreenCapture` 接口时，通过将 `TRTCVideoStreamType` 参数指定为 `TRTCVideoStreamTypeSub` 来启用该模式。观看该路画面需要使用专门的 `startRemoteSubStreamView` 接口。
- **主路分享**
  在 TRTC 中，我们一般把摄像头走的通道叫做“主路（bigstream）”，主路分享即用摄像头通道分享屏幕。该模式下，主播只有一路上行视频流，要么上行摄像头画面，要么上行屏幕画面，两者是互斥的。您可以在调用 `startScreenCapture` 接口时，通过将 `TRTCVideoStreamType` 参数指定为 `TRTCVideoStreamTypeBig` 来启用该模式。

支持的平台

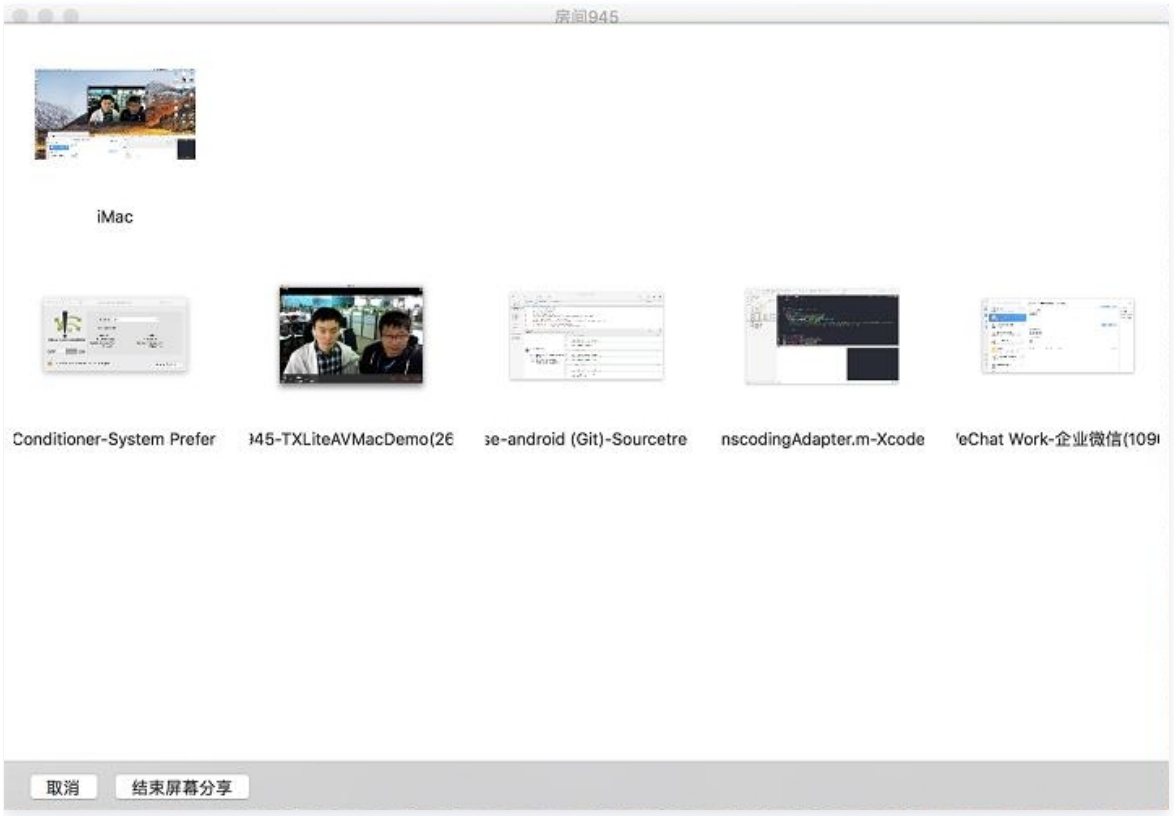
iOS	Android	Mac OS	Windows	Electron	微信小程序	Chrome 浏览器
✓	✓	✓	✓	✓	×	✓

获取分享目标

通过 getScreenCaptureSourcesWithThumbnailSize 可以枚举可共享的窗口列表，每一个可共享的目标都是一个 `TRTCScreenCaptureSourceInfo` 对象。
Mac OS 里的桌面屏幕也是一个可共享目标，普通的 Mac 窗口的 type 为 `TRTCScreenCaptureSourceTypeWindow`，桌面屏幕的 type 为 `TRTCScreenCaptureSourceTypeScreen`。
除了 type，每一个 `TRTCScreenCaptureSourceInfo` 还有如下字段信息：

字段	类型	含义
type	TRTCScreenCaptureSourceType	采集源类型：指定类型为窗口或屏幕
sourceId	NSString	采集源 ID：对于窗口，该字段指示窗口句柄。对于屏幕，该字段指示屏幕 ID
sourceName	NSString	窗口名字，如果是屏幕则返回 Screen0 Screen1...
extInfo	NSDictionary	共享窗口的附加信息
thumbnail	NSImage	窗口缩略图
icon	NSImage	窗口图标

有了上面这些信息，您就可以实现一个简单的列表页面，将可以分享的目标罗列出来供用户选择，如下图：



选择分享目标

TRTC SDK 支持三种分享模式，您可以通过 selectScreenCaptureTarget 来指定：

- **整个屏幕分享：**
  即把整个屏幕窗口分享出去，支持多显示器分屏的情况。需要指定一个 type 为 `TRTCScreenCaptureSourceTypeScreen` 的 screenSource 参数，并将 rect 设为 { 0, 0, 0, 0 }。
- **指定区域分享：**
  即把屏幕的某个区域分享出去，需要用户圈定区域的位置坐标。需要指定一个 type 为 `TRTCScreenCaptureSourceTypeScreen` 的 screenSource 参数，并将 captureRect 设为非 NULL，例如 { 100, 100, 300, 300 }。
- **指定窗口分享：**
  即把目标窗口的内容分享出去，需要用户选择要分享的是哪一个窗口。需要指定一个 type 为 `TRTCScreenCaptureSourceTypeWindow` 的 screenSource 参数，并将 captureRect 设为 { 0, 0, 0, 0 }。

> **说明**
> 两个额外参数：
> - 参数 capturesCursor 用于指定是否捕获鼠标指针。
> - 参数 highlight 用于指定是否高亮正在共享的窗口，以及当捕获图像被遮挡时提示用户移走遮挡。（这一分部的 UI 特效是 SDK 内部实现的）

开始屏幕分享

- 选取分享目标之后，使用 startScreenCapture 接口可以启动屏幕分享。
- 两个函数 pauseScreenCapture 和 stopScreenCapture 的区别在于 pause 会停止屏幕内容的采集，并以暂停那一刻的画面垫片，所以在远端看到一直都是最后一帧画面，直到 resumeScreenCapture 。

```
/**
```

```
 *  7.6 **屏幕共享**启动屏幕分享
 *  @param view 渲染控件所在的父控件
 */
- (void)startScreenCapture:(NSView *)view;

/**
 *  7.7 **屏幕共享**停止屏幕采集
 *  @return 0: 成功 <0:失败
 */
- (int)stopScreenCapture;

/**
 *  7.8 **屏幕共享**暂停屏幕分享
 *  @return 0: 成功 <0:失败
 */
- (int)pauseScreenCapture;

/**
 *  7.9 **屏幕共享**恢复屏幕分享
 *
 *  @return 0: 成功 <0:失败
 */
- (int)resumeScreenCapture;
```

设定画面质量

您可以通过 setSubStreamEncoderParam 接口设定屏幕分享的画面质量，包括分辨率、码率和帧率，我们提供如下建议参考值：

清晰度级别	分辨率	帧率	码率
超高清（HD+）	1920 × 1080	10	800kbps
高清（HD）	1280 × 720	10	600kbps
标清（SD）	960 × 720	10	400kbps

观看屏幕分享

- **观看 Mac / Windows 屏幕分享**
  当房间里有一个 Mac / Windows 用户启动了屏幕分享，会通过辅流进行分享。房间里的其他用户会通过 TRTCCloudDelegate 中的 onUserSubStreamAvailable 事件获得这个通知。
  希望观看屏幕分享的用户可以通过 startRemoteSubStreamView 接口来启动渲染远端用户辅流画面。
- **观看 Android / iOS 屏幕分享**
  若用户通过 Android / iOS 进行屏幕分享，会通过主流进行分享。房间里的其他用户会通过 TRTCCloudDelegate 中的 onUserVideoAvailable 事件获得这个通知。
  希望观看屏幕分享的用户可以通过 startRemoteView 接口来启动渲染远端用户主流画面。

```
//示例代码：观看屏幕分享的画面

- (void)onUserSubStreamAvailable:(NSString *)userId available:(BOOL)available {
    if (available) {
        [self.trtcCloud startRemoteSubStreamView:userId
view:self.capturePreviewWindow.contentView];
```

```
    } else {
        [self.trtcCloud stopRemoteSubStreamView:userId];
    }
}
```

常见问题

一个房间里可以同时有多个人共享屏幕吗?

目前一个 TRTC 音视频房间只能有一路屏幕分享。

指定窗口分享（SourceTypeWindow），当窗口大小变化时，视频流的分辨率会不会也跟着变化?

默认情况下，SDK 内部会自动根据分享的窗口大小进行编码参数的调整。
如需固定分辨率，需调用 setSubStreamEncoderParam 接口设置屏幕分享的编码参数，或在调用 startScreenCapture 时指定对应的编码参数。

Web

最近更新时间：2023-08-16 16:41:02

功能描述

本文主要介绍如何在 TRTC Web SDK 实现屏幕分享功能。

> **说明：**
> - 本教程基于 5.x TRTC Web SDK 实现，若您使用 4.x 版本 SDK，可参考 此教程 。
> - 5.x 与 4.x 版本的区别 。

实现流程

1. “推流端”开启屏幕分享。

```
const trtcA = TRTC.create();
await trtcA.enterRoom({
  scene: 'rtc',
  sdkAppId: 140000000, // 填写您的 sdkAppId
  userId: 'userA', // 填写您的 userId
  userSig: 'userA_sig', // 填写 userId 对应的 userSig
  roomId: 6969
})
await trtcA.startScreenShare();
```

2. “拉流端”播放屏幕分享。

```
const trtcB = TRTC.create();
trtcB.on(TRTC.EVENT.REMOTE_VIDEO_AVAILABLE, ({ userId, streamType }) => {
  // 主路视频流，一般是推摄像头的那路流
  if (streamType === TRTC.TYPE.STREAM_TYPE_MAIN) {
    // 1. 在页面中放置一个 id 为 `${userId}_main` 的 div 标签，用于在 div 标签内播放主路流。业务侧可自定义 div 标签的 id，此处只是举例说明。
    // 2. 播放主路视频流
    trtcB.startRemoteVideo({ userId, streamType,  view: `${userId}_main` });
  } else {
    // 辅路视频流，一般是推屏幕分享的那路流。
    // 1. 在页面中放置一个 id 为 `${userId}_screen` 的 div 标签，用于在 div 标签内播放屏幕分享。业务侧可自定义 div 标签的 id，此处只是举例说明。
    // 2. 播放屏幕分享
    trtcB.startRemoteVideo({ userId, streamType, view: `${userId}_screen` });
  }
});

await trtcB.enterRoom({
  scene: 'rtc',
  sdkAppId: 140000000, // 填写您的 sdkAppId
  userId: 'userB', // 填写您的 userId
  userSig: 'userB_sig', // 填写 userId 对应的 userSig
  roomId: 6969
```

```
})
```

3. 同时推摄像头 + 屏幕分享。

```
await trtcA.startLocalVideo();
await trtcA.startScreenShare();
```

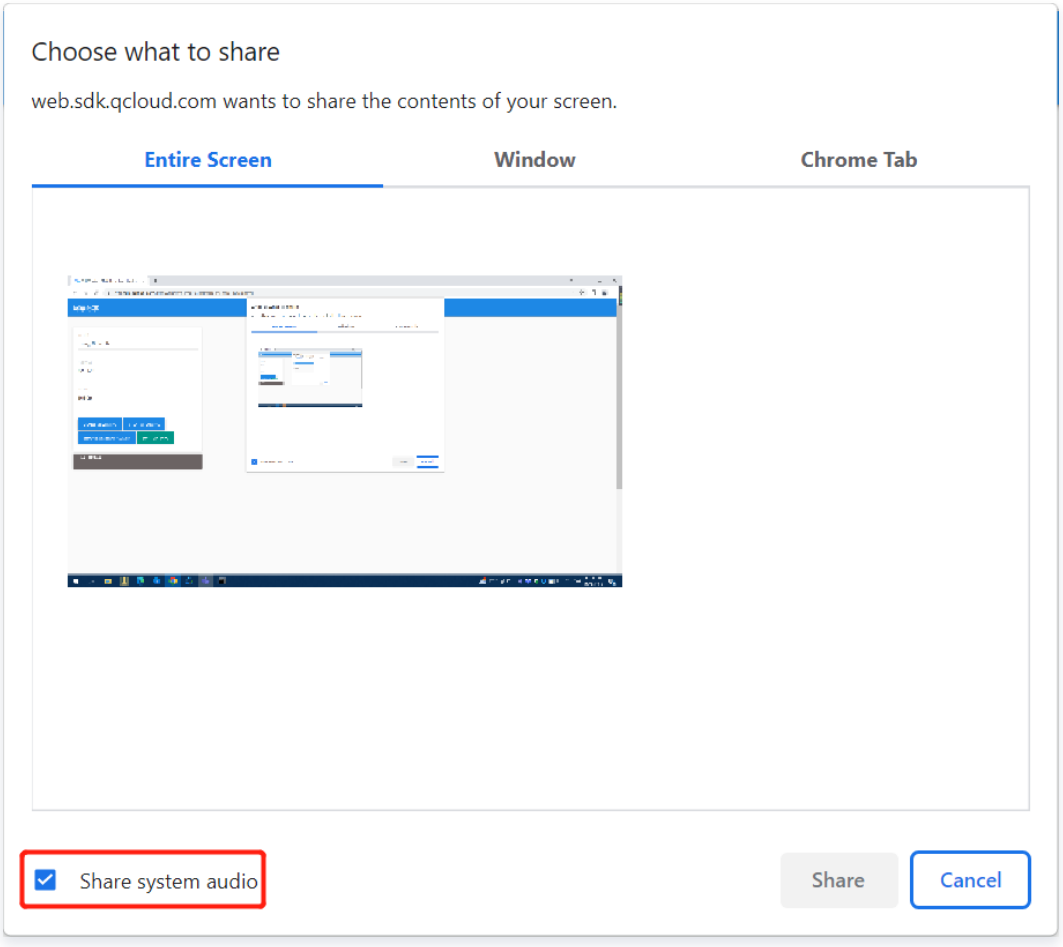
4. 屏幕分享 + 系统音频。

采集系统音频支持 Chrome M74+

- 在 Windows 和 Chrome OS 上，可以采集整个系统的音频。
- 在 Linux 和 Mac 上，只能采集某个页面的音频。
- 其它 Chrome 版本、其它系统、其它浏览器均不支持。

```
await trtcA.startScreenShare({ option: { systemAudio: true }});
```

在弹出的对话框中勾选分享音频，系统音频会与本地麦克风混音后发布，房间内其他用户会收到 TRTC.EVENT.REMOTE_AUDIO_AVAILABLE 事件

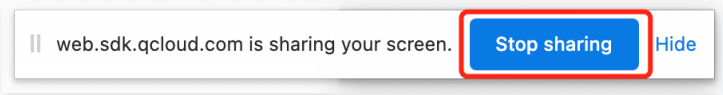


5. 停止屏幕分享

```
// 停止屏幕分享采集及发布
await trtcA.stopScreenShare();
```

```
// 房间内的其他用户会收到 TRTC.EVENT.REMOTE_VIDEO_UNAVAILABLE 事件，streamType 是
TRTC.TYPE.STREAM_TYPE_SUB。
trtcB.on(TRTC.EVENT.REMOTE_VIDEO_UNAVAILABLE, ({ userId, streamType }) => {
    if (streamType === TRTC.TYPE.STREAM_TYPE_SUB) {

    }
})
```

另外用户还可能会通过浏览器自带的按钮停止屏幕分享，因此屏幕分享流需要监听屏幕分享停止事件，并进行相应的处理。




```
// 监听屏幕分享停止事件
trtcA.on(TRTC.EVENT.SCREEN_SHARE_STOPPED, () => {
  console.log('screen sharing was stopped');
});
```

注意事项

1. 什么是主流，辅流?
2. SDK 默认使用 1080p 参数配置来采集屏幕分享，具体参考接口：`TRTC.startScreenShare`。

常见问题

Safari 屏幕分享出现报错 `getDisplayMedia must be called from a user gesture handler`？

这是因为 Safari 限制了 `getDisplayMedia` 屏幕采集的接口，必须在用户点击事件的回调函数执行的 1 秒内才可以调用。
参考：webkit issue。

```
// good
async function onClick() {
  // 建议在 onClick 执行时，先执行采集逻辑
  await trtcA.startScreenShare();
  await trtcA.enterRoom({
    roomId: 123123,
    sdkAppId: 140000000, // 填写您的 sdkAppId
    userId: 'userA', // 填写您的 userId
    userSig: 'userA_sig', // 填写 userId 对应的 userSig });
});

// bad
async function onClick() {
  await trtcA.enterRoom({
    roomId: 123123,
    sdkAppId: 140000000, // 填写您的 sdkAppId
    userId: 'userA', // 填写您的 userId
    userSig: 'userA_sig', // 填写 userId 对应的 userSig });
  })
  // 进房可能耗时超过 1s，可能会采集失败
   await trtcA.startScreenShare();
}
```

Mac Chrome 在已授权屏幕录制的情况下屏幕分享失败，出现 "NotAllowedError: Permission denied by system" 或者 "NotReadableError: Could not start video source" 错误信息，Chrome bug?

解决方案：打开【设置】> 点击【安全性与隐私】> 点击【隐私】> 点击【屏幕录制】> 关闭 Chrome 屏幕录制授权 > 重新打开 Chrome 屏幕录制授权 > 关闭 Chrome 浏览器 > 重新打开 Chrome 浏览器。

WebRTC 屏幕分享已知问题及规避方案?

详细信息介绍请点击 WebRTC 屏幕分享已知问题及规避方案 。

Windows

最近更新时间：2023-10-16 14:48:42

本文档主要介绍如何使用屏幕分享，目前一个 TRTC 音视频房间只能有一路屏幕分享。
Windows 平台下的屏幕分享支持主路分享和辅路分享两种方案：

- **辅路分享**
  在 TRTC 中，我们可以单独为屏幕分享开启一路上行的视频流，并称之为“辅路（substream）”。辅路分享即主播同时上行摄像头画面和屏幕画面两路画面。这是腾讯会议的使用方案，您可以在调用 `startScreenCapture` 接口时，通过将 `TRTCVideoStreamType` 参数指定为 `TRTCVideoStreamTypeSub` 来启用该模式。
- **主路分享**
  在 TRTC 中，我们一般把摄像头走的通道叫做“主路（bigstream）”，主路分享即用摄像头通道分享屏幕。该模式下，主播只有一路上行视频流，要么上行摄像头画面，要么上行屏幕画面，两者是互斥的。您可以在调用 `startScreenCapture` 接口时，通过将 `TRTCVideoStreamType` 参数指定为 `TRTCVideoStreamTypeBig` 来启用该模式。

依赖的 API

API 功能	C++ 版本	C# 版本	Electron 版本
选择分享目标	selectScreenCaptureTarget	selectScreenCaptureTarget	selectScreenCaptureTarget
开始屏幕分享	startScreenCapture	startScreenCapture	startScreenCapture
暂停屏幕分享	pauseScreenCapture	pauseScreenCapture	pauseScreenCapture
恢复屏幕分享	resumeScreenCapture	resumeScreenCapture	resumeScreenCapture
结束屏幕分享	stopScreenCapture	stopScreenCapture	stopScreenCapture

获取分享目标

通过 `getScreenCaptureSources` 可以枚举可共享的窗口列表，列表通过出参 sourceInfoList 返回。

> **说明**
> Windows 里的桌面屏幕也是一个窗口，叫桌面窗口（Desktop），有两台显示器时，每一台显示器都有一个对应的桌面窗口。所以，getScreenCaptureSources 返回的窗口列表里也会有 Desktop 窗口。

根据获取到的窗口信息，您可以实现一个简单的列表页面，将可以分享的目标罗列出来供用户选择，如下图：

版权所有：腾讯云计算（北京）有限责任公司

开始屏幕分享

- 选取分享目标后，使用 `startScreenCapture` 接口可以启动屏幕分享。
- 分享过程中，您依然可以通过调用 `selectScreenCaptureTarget` 更换分享目标。
- `pauseScreenCapture` 和 `stopScreenCapture` 的区别在于 pause 会停止屏幕内容的采集，并以暂停那一刻的画面垫片，所以在远端看到一直都是最后一帧画面，直到 resume。

设定画面质量

您可以通过 `setSubStreamEncoderParam` 接口设定屏幕分享的画面质量，包括分辨率、码率和帧率，我们提供如下建议参考值：

清晰度级别	分辨率	帧率	码率
超高清（HD+）	1920 × 1080	10	800kbps
高清（HD）	1280 × 720	10	600kbps
标清（SD）	960 × 720	10	400kbps

观看屏幕分享

当房间里有一个用户启动了屏幕分享，会通过辅流进行分享。房间里的其他用户会通过 ITRTCCloudCallback 中的 onUserSubStreamAvailable 事件获得这个通知。
希望观看屏幕分享的用户可以通过 startRemoteView 接口来启动渲染远端用户辅流画面。

```
//示例代码：观看屏幕分享的画面
void CTRTCCloudSDK::onUserSubStreamAvailable(const char * userId, bool available) {
    LINFO(L"onUserSubStreamAvailable userId[%s] available[%d]\n", UTF82Wide(userId).c_str(),
available);
    liteav::ITRTCCloud* trtc_cloud_ = getTRTCShareInstance();
    if (available) {
        trtc_cloud_->startRemoteView(userId, liteav::TRTCVideoStreamTypeSub, hWnd);
```

```
    } else {
        trtc_cloud_->stopRemoteView(userId, liteav::TRTCVideoStreamTypeSub);
    }
}
```

常见问题

1. 一个房间里可以同时有多路屏幕分享吗?

目前一个 TRTC 音视频房间只能有一路屏幕分享。

2. 指定窗口分享（SourceTypeWindow），当窗口大小变化时，视频流的分辨率会不会也跟着变化?

默认情况下，SDK 内部会自动根据分享的窗口大小进行编码参数的调整。
如需固定分辨率，需调用 setSubStreamEncoderParam 接口设置屏幕分享的编码参数，或在调用 startScreenCapture 时指定对应的编码参数。

Electron

最近更新时间：2023-10-16 14:48:42

本文档主要介绍如何使用屏幕分享，目前一个 TRTC 音视频房间只能有一路屏幕分享。
Electron 平台下的屏幕分享支持主路分享和辅路分享两种方案：

- **辅路分享**
  在 TRTC 中，我们可以单独为屏幕分享开启一路上行的视频流，并称之为“辅路（substream）”。辅路分享即主播同时上行摄像头画面和屏幕画面两路画面。这是腾讯会议的使用方案，您可以在调用 `startScreenCapture` 接口时，通过将 `TRTCVideoStreamType` 参数指定为 `TRTCVideoStreamTypeSub` 来启用该模式。
- **主路分享**
  在 TRTC 中，我们一般把摄像头走的通道叫做“主路（bigstream）”，主路分享即用摄像头通道分享屏幕。该模式下，主播只有一路上行视频流，要么上行摄像头画面，要么上行屏幕画面，两者是互斥的。您可以在调用 `startScreenCapture` 接口时，通过将 `TRTCVideoStreamType` 参数指定为 `TRTCVideoStreamTypeBig` 来启用该模式。

步骤1：获取分享目标

通过 `getScreenCaptureSources` 可以枚举可共享的窗口列表，列表通过出参 sourceInfoList 返回。

> **说明**
> Electron 里的桌面屏幕也是一个窗口，叫桌面窗口（Desktop），有两台显示器时，每一台显示器都有一个对应的桌面窗口。所以，getScreenCaptureSources 返回的窗口列表里也会有 Desktop 窗口。

根据获取到的窗口信息，您可以实现一个简单的列表页面，将可以分享的目标罗列出来供用户选择，如下图：


```
import TRTCCloud from 'trtc-electron-sdk';
const rtcCloud = new TRTCCloud();
// https://web.sdk.qcloud.com/trtc/electron/doc/zh-
cn/trtc_electron_sdk/TRTCCloud.html#getScreenCaptureSources
const screenList = rtcCloud.getScreenCaptureSources(320, 180, 32, 32);
```

步骤2：开始屏幕分享

- 可以通过调用 selectScreenCaptureTarget 选取分享目标。
- 选取分享目标后，使用 startScreenCapture 接口可以启动屏幕分享。
- 分享过程中，您依然可以通过调用 selectScreenCaptureTarget 更换分享目标。
- pauseScreenCapture 和 stopScreenCapture 的区别在于 pause 会停止屏幕内容的采集，并以暂停那一刻的画面垫片，所以在远端看到一直都是最后一帧画面，直到 resume。

```
import TRTCCloud, {
    Rect, TRTCScreenCaptureProperty, TRTCVideoStreamType, TRTCVideoEncParam,
    TRTCVideoResolution, TRTCVideoResolutionMode
} from 'trtc-electron-sdk';
const rtcCloud = new TRTCCloud();
// https://web.sdk.qcloud.com/trtc/electron/doc/zh-
cn/trtc_electron_sdk/TRTCCloud.html#getScreenCaptureSources
const screenList = rtcCloud.getScreenCaptureSources(320, 180, 32, 32);
// https://web.sdk.qcloud.com/trtc/electron/doc/zh-cn/trtc_electron_sdk/Rect.html
const captureRect = new Rect(0, 0, 0, 0);
// https://web.sdk.qcloud.com/trtc/electron/doc/zh-
cn/trtc_electron_sdk/TRTCScreenCaptureProperty.html
const property = new TRTCScreenCaptureProperty(
    true, true, true, 0, 0, false
);
if (screenList.length > 0) {
    rtcCloud.selectScreenCaptureTarget(screenList[0], captureRect, property)
}

const screenshareDom = document.querySelector('screen-dom');
// https://web.sdk.qcloud.com/trtc/electron/doc/zh-
cn/trtc_electron_sdk/TRTCVideoEncParam.html
const encParam = new TRTCVideoEncParam(
    TRTCVideoResolution.TRTCVideoResolution_1920_1080,
    TRTCVideoResolutionMode.TRTCVideoResolutionModeLandscape,
    15,
    2000,
    0,
    false
);
rtcCloud.startScreenCapture(screenshareDom, TRTCVideoStreamType.TRTCVideoStreamTypeSub,
encParam);
```

步骤3：设定画面质量

您可以通过 `startScreenCapture` 接口的第三个参数 `encParam` 设定屏幕分享的画面质量(参见 步骤2)，包括分辨率、码率和帧率，我们提供如下建议参考值：

清晰度级别	分辨率	帧率	码率
超高清（HD+）	1920 × 1080	10	2000kbps
高清（HD）	1280 × 720	10	600kbps
标清（SD）	960 × 720	10	400kbps

步骤4：观看屏幕分享

当房间里有一个用户启动了屏幕分享，会通过辅流进行分享。房间里的其他用户会通过 onUserSubStreamAvailable 事件获得这个通知。希望观看屏幕分享的用户可以通过 startRemoteView 接口来启动渲染远端用户辅流画面。

```
import TRTCCloud, {
    TRTCVideoStreamType
} from 'trtc-electron-sdk';
const rtcCloud = new TRTCCloud();

const remoteDom = document.querySelector('.remote-user');
function onUserSubStreamAvailable(userId, available) {
    if (available === 1) {
        rtcCloud.startRemoteView(userId, remoteDom,
TRTCVideoStreamType.TRTCVideoStreamTypeSub);
    } else {
        rtcCloud.stopRemoteView(userId, TRTCVideoStreamType.TRTCVideoStreamTypeSub);
    }
}

rtcCloud.on('onUserSubStreamAvailable', onUserSubStreamAvailable);
```

常见问题

1. 一个房间里可以同时有多路屏幕分享吗?

目前一个 TRTC 音视频房间只能有一路屏幕分享。

2. 指定窗口分享（SourceTypeWindow），当窗口大小变化时，视频流的分辨率会不会也跟着变化?

默认情况下，SDK 内部会自动根据分享的窗口大小进行编码参数的调整。
如需固定分辨率，需调用 setSubStreamEncoderParam 接口设置屏幕分享的编码参数，或在调用 startScreenCapture 时指定对应的编码参数。

Flutter

最近更新时间：2024-05-29 10:58:31

基于 Android 平台

腾讯云 TRTC 在 Android 系统上支持屏幕分享，即将当前系统的屏幕内容通过 TRTC SDK 分享给房间里的其他用户。关于此功能，有两点需要注意：

- 移动端 TRTC Android 8.6 之前的版本屏幕分享并不像桌面端版本一样支持“辅路分享”，因此在启动屏幕分享时，摄像头的采集需要先被停止，否则会相互冲突；8.6 及之后的版本支持“辅路分享”，则不需要停止摄像头的采集。
- 当一个 Android 系统上的后台 App 在持续使用 CPU 时，很容易会被系统强行杀掉，而且屏幕分享本身又必然会消耗 CPU。要解决这个看似矛盾的冲突，我们需要在 App 启动屏幕分享的同时，在 Android 系统上弹出悬浮窗。由于 Android 不会强杀包含前台 UI 的 App 进程，因此该种方案可以让您的 App 可以持续进行屏幕分享而不被系统自动回收。如下图所示：

启动屏幕分享

要开启 Android 端的屏幕分享，只需调用 `TRTCCloud` 中的 startScreenCapture() 接口即可。但如果要达到稳定和清晰的分享效果，您需要关注如下三个问题：

添加 Activity

在 manifest 文件中粘贴如下 activity（若项目代码中存在则不需要添加）。

```
<activity
    android:name="com.tencent.rtmp.video.TXScreenCapture$TXScreenCaptureAssistantActivity"
    android:theme="@android:style/Theme.Translucent"/>
```

设定视频编码参数

通过设置 startScreenCapture() 中的首个参数 `encParams` ，您可以指定屏幕分享的编码质量。如果您指定 `encParams` 为 null，SDK 会自动使用之前设定的编码参数，我们推荐的参数设定如下：

参数项	参数名称	常规推荐值	文字教学场景
分辨率	videoResolution	1280 × 720	1920 × 1080
帧率	videoFps	10 FPS	8 FPS
最高码率	videoBitrate	1600 kbps	2000 kbps
分辨率自适应	enableAdjustRes	NO	NO

说明：

- 由于屏幕分享的内容一般不会剧烈变动，所以设置较高的 FPS 并不经济，推荐10 FPS即可。
- 如果您要分享的屏幕内容包含大量文字，可以适当提高分辨率和码率设置。
- 最高码率（videoBitrate）是指画面在剧烈变化时的最高输出码率，如果屏幕内容变化较少，实际编码码率会比较低。

基于 iOS 平台

- 应用内分享
  即只能分享当前 App 的画面，该特性需要 iOS 13 及以上版本的操作系统才能支持。由于无法分享当前 App 之外的屏幕内容，因此适用于对隐私保护要求高的场景。
- 跨应用分享
  基于苹果的 Replaykit 方案，能够分享整个系统的屏幕内容，但需要当前 App 额外提供一个 Extension 扩展组件，因此对接步骤也相对应用内分享要多一点。

方案1：iOS 平台应用内分享

应用内分享的方案非常简单，只需要调用 TRTC SDK 提供的接口 startScreenCapture 并传入编码参数 `TRTCVideoEncParam` 和 参数 `appGroup` 设置为 `''` 。`TRTCVideoEncParam` 参数可以设置为 null，此时 SDK 会沿用开始屏幕分享之前的编码参数。
我们推荐的用于 iOS 屏幕分享的编码参数是：

参数项	参数名称	常规推荐值	文字教学场景
分辨率	videoResolution	1280 × 720	1920 × 1080
帧率	videoFps	10 FPS	8 FPS
最高码率	videoBitrate	1600 kbps	2000 kbps
分辨率自适应	enableAdjustRes	NO	NO

说明：

- 由于屏幕分享的内容一般不会剧烈变动，所以设置较高的 FPS 并不经济，推荐10 FPS 即可。
- 如果您要分享的屏幕内容包含大量文字，可以适当提高分辨率和码率设置。
- 最高码率（videoBitrate）是指画面在剧烈变化时的最高输出码率，如果屏幕内容变化较少，实际编码码率会比较低。

方案2：iOS 平台跨应用分享

示例代码

我们在 Github 中的 trtc_demo/ios 目录下放置了一份跨应用分享的示例代码，其包含如下一些文件：

```
├── Broadcast.Upload          //录屏进程 Broadcast Upload Extension 代码详见步骤2
│    ├── Broadcast.Upload.entitlements        //用于设置进程间通信的 AppGroup 信息
│    ├── Broadcast.UploadDebug.entitlements   //用于设置进程间通信的 AppGroup 信息（debug环境）
│    ├── Info.plist
│    └── SampleHandler.swift      // 用于接收来自系统的录屏数据
├── Resource                     // 资源文件
├── Runner                       // TRTC 精简化 Demo
├── TXLiteAVSDK_ReplayKitExt.framework      //TXLiteAVSDK_ReplayKitExt SDK
```

您可以通过 README 中的指引跑通该示例 Demo。

对接步骤

iOS 系统上的跨应用屏幕分享，需要增加 Extension 录屏进程以配合主 App 进程进行推流。Extension 录屏进程由系统在需要录屏的时候创建，并负责接收系统采集到屏幕图像。因此需要：

1. 创建 App Group，并在 XCode 中进行配置（可选）。这一步的目的是让 Extension 录屏进程可以同主 App 进程进行跨进程通信。
2. 在您的工程中，新建一个 Broadcast Upload Extension 的 Target，并在其中集成 SDK 压缩包中专门为扩展模块定制的 `TXLiteAVSDK_ReplayKitExt.framework`。
3. 对接主 App 端的接收逻辑，让主 App 等待来自 Broadcast Upload Extension 的录屏数据。
4. 编辑 `pubspec.yaml` 文件引入 `replay_kit_launcher` 插件，实现类似TRTC Demo Screen中点击一个按钮即可唤起屏幕分享的效果（可选）。

```
# 引入 trtc sdk和replay_kit_launcher
dependencies:
  tencent_trtc_cloud: ^0.2.1
  replay_kit_launcher: ^0.2.0+1
```

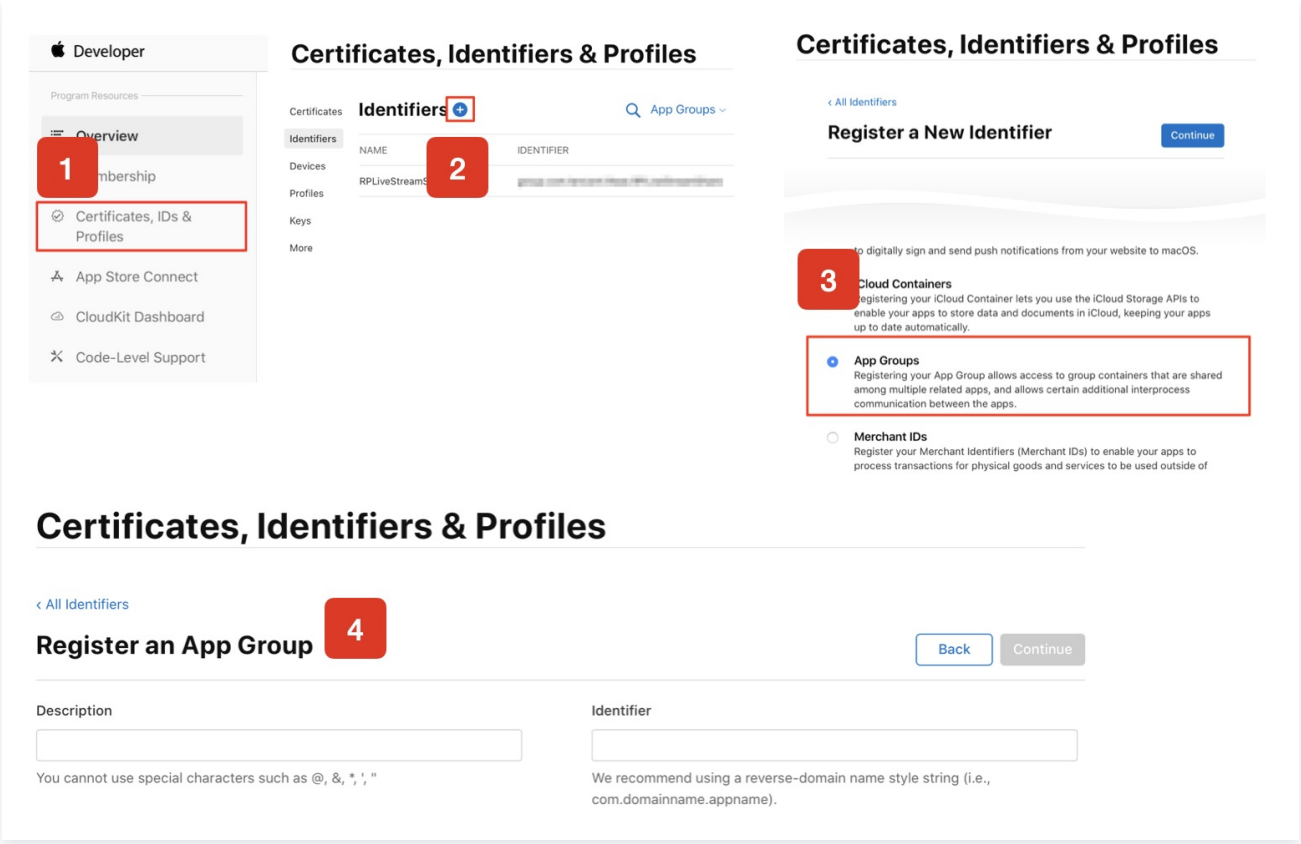
> ⚠ **注意：**
> 如果跳过 步骤1，也就是不配置 App Group（接口传 null），屏幕分享依然可以运行，但稳定性要打折扣，故虽然步骤较多，但请尽量配置正确的 App Group 以保障屏幕分享功能的稳定性。

步骤1：创建 App Group

使用您的账号登录 https://developer.apple.com/，进行以下操作，**注意完成后需要重新下载对应的 Provisioning Profile**。

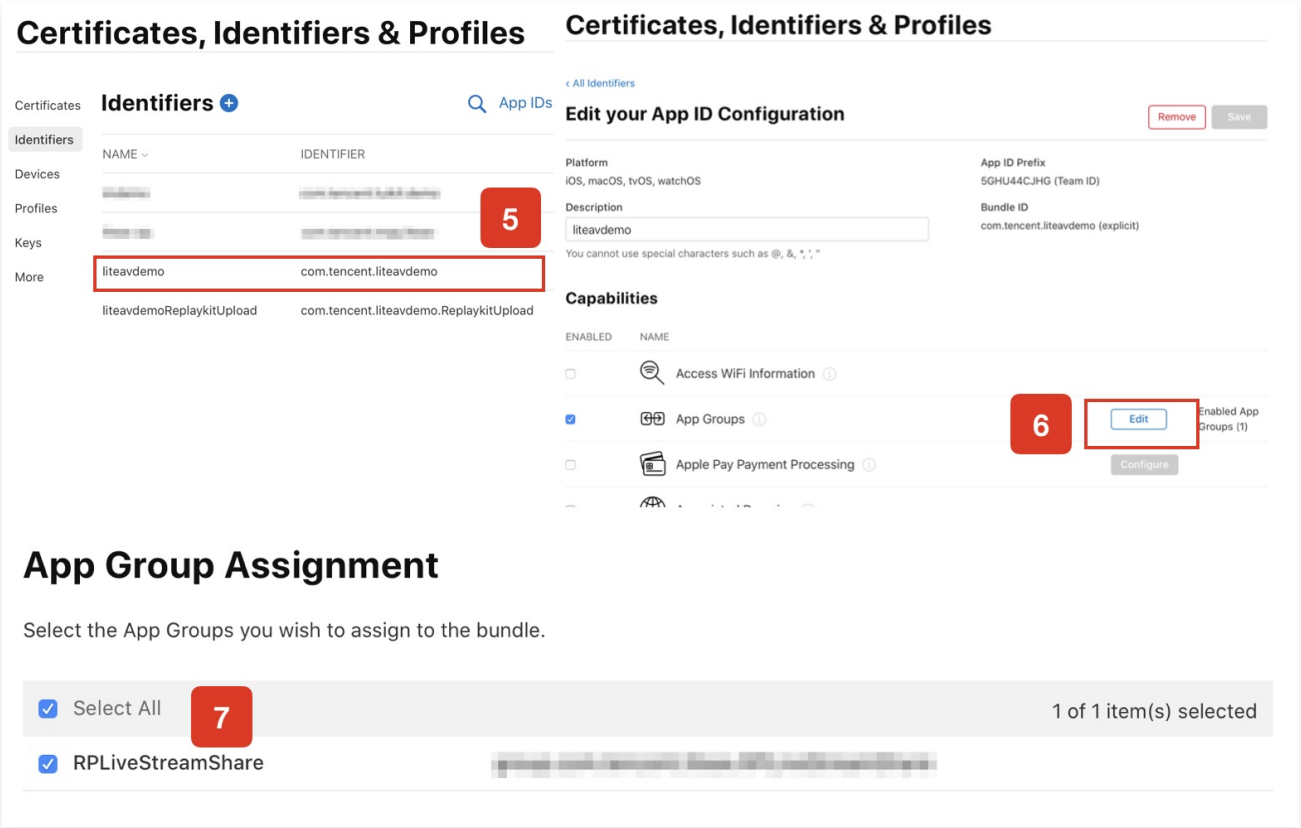
1. 单击 **Certificates, IDs & Profiles**。
2. 在右侧的界面中单击加号。
3. 选择 **App Groups**，单击 **Continue**。

4. 在弹出的表单中填写 Description 和 Identifier，其中 Identifier 需要传入接口中的对应的 AppGroup 参数。完成后单击 **Continue**。



5. 回到 Identifier 页面，左上边的菜单中选择 **App IDs**，然后单击您的 App ID（主 App 与 Extension 的 AppID 需要进行同样的配置）。
6. 选中 **App Groups** 并单击 **Edit**。

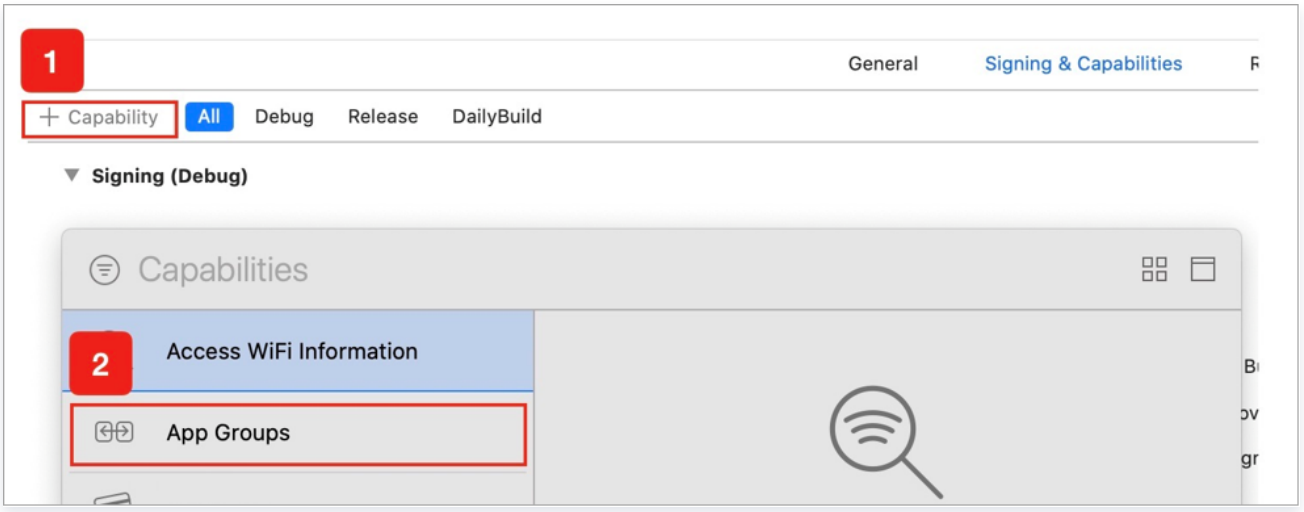
7. 在弹出的表单中选择您之前创建的 App Group，单击 **Continue** 返回编辑页，单击 **Save** 保存。



8. 重新下载 Provisioning Profile 并配置到 XCode 中。

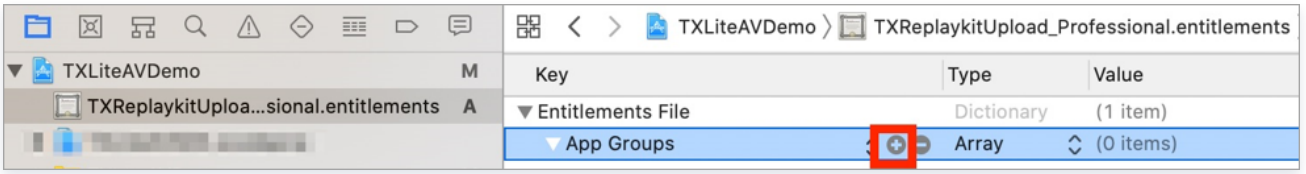
步骤2：创建 Broadcast Upload Extension

1. 在 Xcode 菜单依次单击 **File** > **New** > **Target...**，选择 **Broadcast Upload Extension**。
2. 在弹出的对话框中填写相关信息，**不用**勾选 **Include UI Extension**，单击 **Finish** 完成创建。
3. 将下载到的 SDK 压缩包中的 TXLiteAVSDK_ReplayKitExt.framework 拖动到工程中，勾选刚创建的 Target。
4. 选中新增加的 Target，依次单击 **+ Capability**，双击 **App Groups**，如下图：



操作完成后，会在文件列表中生成一个名为 `Target.entitlements` 的文件，如下图所示，选中该文件并单击 + 号填写上述步骤中的

App Group 即可。



5. 选中主 App 的 Target，**并按照上述步骤对主 App 的 Target 做同样的处理。**
6. 在新创建的 Target 中，Xcode 会自动创建一个名为 "SampleHandler.swift" 的文件，用如下代码进行替换。**需将代码中的 APPGROUP 改为上文中的创建的 App Group Identifier。**

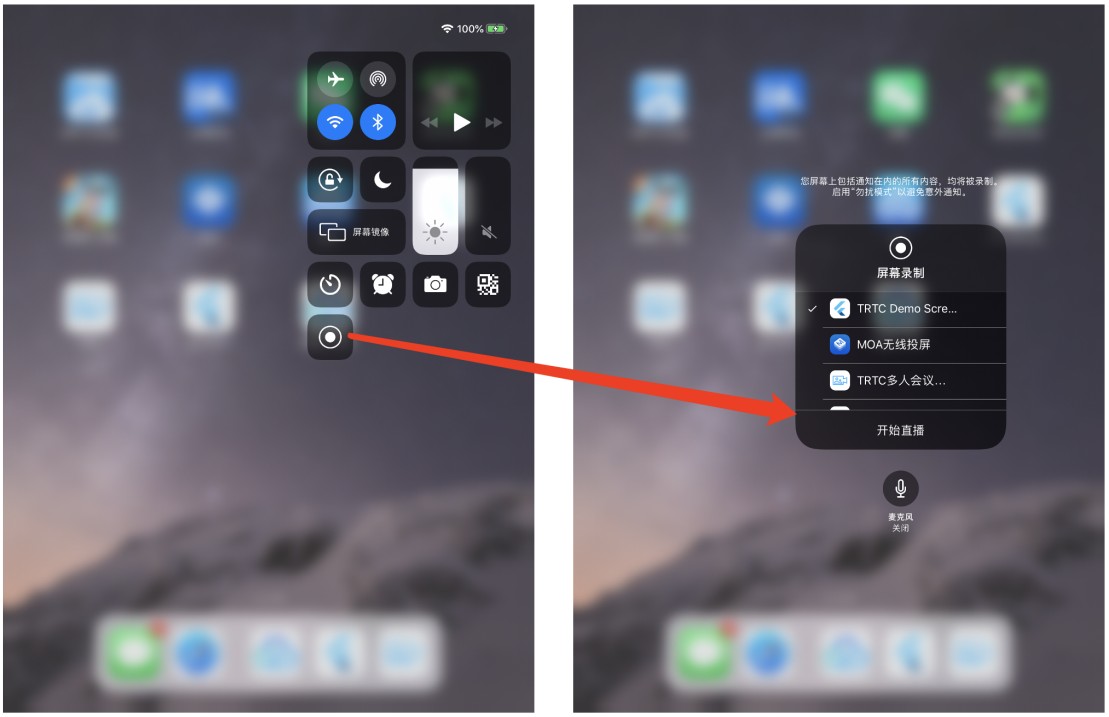
```
import ReplayKit
import TXLiteAVSDK_ReplayKitExt

let APPGROUP = "group.com.tencent.comm.trtc.demo"

class SampleHandler: RPBroadcastSampleHandler, TXReplayKitExtDelegate {

    let recordScreenKey = Notification.Name.init("TRTCRecordScreenKey")

    override func broadcastStarted(withSetupInfo setupInfo: [String : NSObject]?) {
        // User has requested to start the broadcast. Setup info from the UI extension can
be supplied but optional.
        TXReplayKitExt.sharedInstance().setup(withAppGroup: APPGROUP, delegate: self)
    }

    override func broadcastPaused() {
        // User has requested to pause the broadcast. Samples will stop being delivered.
    }

    override func broadcastResumed() {
        // User has requested to resume the broadcast. Samples delivery will resume.
    }

    override func broadcastFinished() {
        // User has requested to finish the broadcast.
        TXReplayKitExt.sharedInstance() .finishBroadcast()
    }

    func broadcastFinished(_ broadcast: TXReplayKitExt, reason: TXReplayKitExtReason) {
        var tip = ""
        switch reason {
        case TXReplayKitExtReason.requestedByMain:
            tip = "屏幕共享已结束"
            break
        case TXReplayKitExtReason.disconnected:
            tip = "应用断开"
            break
        case TXReplayKitExtReason.versionMismatch:
            tip = "集成错误（SDK 版本号不相符合）"
            break
        default:
```

```
            break
        }

        let error = NSError(domain: NSStringFromClass(self.classForCoder), code: 0,
userInfo: [NSLocalizedFailureReasonErrorKey:tip])
        finishBroadcastWithError(error)
    }

    override func processSampleBuffer(_ sampleBuffer: CMSampleBuffer, with sampleBufferType:
RPSampleBufferType) {
        switch sampleBufferType {
        case RPSampleBufferType.video:
            // Handle video sample buffer
            TXReplayKitExt.sharedInstance() .sendVideoSampleBuffer(sampleBuffer)
            break
        case RPSampleBufferType.audioApp:
            // Handle audio sample buffer for app audio
            break
        case RPSampleBufferType.audioMic:
            // Handle audio sample buffer for mic audio
            break
        @unknown default:
            // Handle other sample buffer types
            fatalError("Unknown type of sample buffer")
        }
    }
}
```

步骤3：对接主 App 端的接收逻辑

按照如下步骤，对接主 App 端的接收逻辑。也就是在用户触发屏幕分享之前，要让主 App 处于“等待”状态，以便随时接收来自 Broadcast Upload Extension 进程的录屏数据。

1. 确保 TRTCCloud 已经关闭了摄像头采集，如果尚未关闭，请调用 stopLocalPreview 关闭摄像头采集。
2. 调用 startScreenCapture 方法，并传入 步骤1 中设置的 AppGroup，让 SDK 进入“等待”状态。
3. 等待用户触发屏幕分享。如果不实现 步骤4 中的“触发按钮”，屏幕分享就需要用户在 iOS 系统的控制中心，通过长按录屏按钮来触发，这一操作步骤如下图所示：



长按“录制”按钮　　点击开始直播

4. 通过调用 stopScreenCapture 接口可以随时中止屏幕分享。

```
// 开始屏幕分享，需要将 APPGROUP 替换为上述步骤中创建的 App Group
trtcCloud.startScreenCapture(
    TRTCVideoEncParam(
        videoFps: 10,
        videoResolution: TRTCCloudDef.TRTC_VIDEO_RESOLUTION_1280_720,
        videoBitrate: 1600,
        videoResolutionMode: TRTCCloudDef.TRTC_VIDEO_RESOLUTION_MODE_PORTRAIT,
    ),
    iosAppGroup,
);

// 停止屏幕分享
await trtcCloud.stopScreenCapture();

// 屏幕分享的启动事件通知，可以通过 TRTCCloudListener 进行接收
onRtcListener(type, param){
    if (type == TRTCCloudListener.onScreenCaptureStarted) {
      //屏幕分享开始
    }
}
```

步骤4：增加屏幕分享的触发按钮（可选）

截止到 步骤3，我们的屏幕分享还必须要用户从控制中心中长按录屏按钮来手动启动。您可通过下述方法实现类似 TRTC Demo Screen 的单击按钮即可触发的效果：

1. 将 `replay_kit_launcher` 插件加入到您的工程中。
2. 在您的界面上放置一个按钮，并在按钮的响应函数中调用 `ReplayKitLauncher.launchReplayKitBroadcast(iosExtensionName);` 函数，就可以唤起屏幕分享功能了。

```
// 自定义按钮响应方法
onShareClick() async {
  if (Platform.isAndroid) {
    if (await SystemAlertWindow.requestPermissions) {
      MeetingTool.showOverlayWindow();
    }
  } else {
    //屏幕分享功能只能在真机测试
    ReplayKitLauncher.launchReplayKitBroadcast(iosExtensionName);
  }
}
```

基于 Windows 平台

Windows 平台下的屏幕分享支持主路分享和辅路分享两种方案：

- **辅路分享**
  在 TRTC 中，我们可以单独为屏幕分享开启一路上行的视频流，并称之为“辅路（substream）”。辅路分享即主播同时上行摄像头画面和屏幕画面两路画面。这是腾讯会议的使用方案，您可以在调用 `startScreenCapture` 接口时，通过将 `TRTCVideoStreamType` 参数指定为 `TRTCVideoStreamTypeSub` 来启用该模式。
- **主路分享**
  在 TRTC 中，我们一般把摄像头走的通道叫做“主路（bigstream）”，主路分享即用摄像头通道分享屏幕。该模式下，主播只有一路上

行视频流，要么上行摄像头画面，要么上行屏幕画面，两者是互斥的。您可以在调用 `startScreenCapture` 接口时，通过将 `TRTCVideoStreamType` 参数指定为 `TRTCVideoStreamTypeBig` 来启用该模式。

步骤一：获取分享目标

通过 getScreenCaptureSources 可以枚举可共享的窗口列表，列表通过出参 sourceInfoList 返回。

> **说明：**
> Windows 里的桌面屏幕也是一个窗口，叫桌面窗口（Desktop），有两台显示器时，每一台显示器都有一个对应的桌面窗口。所以，getScreenCaptureSources 返回的窗口列表里也会有 Desktop 窗口。

根据获取到的窗口信息，您可以实现一个简单的列表页面，将可以分享的目标罗列出来供用户选择。

步骤二：选择分享目标

当您通过 getScreenCaptureSources 获取到可以分享的屏幕和窗口之后，您可以调用 selectScreenCaptureTarget 接口选定期望分享的目标屏幕或目标窗口。

步骤三：开始屏幕分享

- 选取分享目标后，使用 startScreenCapture 接口可以启动屏幕分享。
- 分享过程中，您依然可以通过调用 selectScreenCaptureTarget 更换分享目标。
- pauseScreenCapture 和 stopScreenCapture 的区别在于 pause 会停止屏幕内容的采集，并以暂停那一刻的画面垫片，所以在远端看到一直都是最后一帧画面，直到 resume。

设定视频编码参数

通过设置 startScreenCapture() 中的首个参数 `encParams`，您可以指定屏幕分享的编码质量，包括分辨率、码率和帧率，我们提供如下建议参考值：

清晰度级别	分辨率	帧率	码率
超高清（HD+）	1920 × 1080	10	800kbps
高清（HD）	1280 × 720	10	600kbps
标清（SD）	960 × 720	10	400kbps

如果您指定 `encParams` 为 null，SDK 会自动使用之前设定的编码参数。

观看屏幕分享

- **观看屏幕分享**
  若用户进行屏幕分享，会通过主流进行分享。房间里的其他用户会通过 TRTCCloudListener 中的 onUserVideoAvailable 事件获得这个通知。
  希望观看屏幕分享的用户可以通过 startRemoteView 接口来启动渲染远端用户主流画面。

常见问题

一个房间里可以同时有多路屏幕分享吗？

目前一个 TRTC 音视频房间只能有一路屏幕分享。

uni-app

最近更新时间：2025-06-25 10:09:13

本文档主要介绍如何使用屏幕分享，目前一个 TRTC 音视频房间只能有一路屏幕分享。

调用指引

开启屏幕分享

要开启屏幕分享，只需调用 startScreenCapture() 接口即可。
通过设置 startScreenCapture() 中参数 `encParams` ，您可以指定屏幕分享的编码质量。如果您指定 `encParams` 为 null，SDK 会自动使用之前设定的编码参数，我们推荐的参数设定如下：

参数项	参数名称	常规推荐值	文字教学场景
分辨率	videoResolution	1280 × 720	1920 × 1080
帧率	videoFps	10 FPS	8 FPS
最高码率	videoBitrate	1600 kbps	2000 kbps
分辨率自适应	enableAdjustRes	NO	NO

- 由于屏幕分享的内容一般不会剧烈变动，所以设置较高的 FPS 并不经济，推荐10 FPS即可。
- 如果您要分享的屏幕内容包含大量文字，可以适当提高分辨率和码率设置。
- 最高码率（videoBitrate）是指画面在剧烈变化时的最高输出码率，如果屏幕内容变化较少，实际编码码率会比较低。

```
import TrtcCloud from "@/TrtcCloud/lib/index";
import { TRTCVideoStreamType, TRTCVideoResolution, TRTCVideoResolutionMode } from
'@/TrtcCloud/lib/TrtcDefines';
this.trtcCloud = TrtcCloud.createInstance();

const encParams = {
  videoResolution: TRTCVideoResolution.TRTCVideoResolution_1280_720,
  videoResolutionMode: TRTCVideoResolutionMode.TRTCVideoResolutionModePortrait,
  videoFps: 15,
  videoBitrate: 900,
  minVideoBitrate: 200,
  enableAdjustRes: false,
};
this.trtcCloud.startScreenCapture(TRTCVideoStreamType.TRTCVideoStreamTypeSub, encParams);
```

版权所有：腾讯云计算（北京）有限责任公司

分享系统声音
Android

最近更新时间：2024-12-12 18:13:13

本文档主要介绍如何分享系统声音，目前TRTC默认不采集所在应用的音频。

> ⚠ **注意：**
> 仅 Android 10.0 及以上版本支持分享系统声音。

调用指引

开启分享系统声音

步骤1: 开启屏幕分享

按照 开启屏幕分享 中的步骤 1 配置所需的 Activity，并按照步骤 2 开启屏幕分享。

步骤2: 开启分享系统声音

调用 `TRTCCloud` 中的 startSystemAudioLoopback 接口即可，这时候采集的系统声音会自动混入上行流中。

步骤3: 关闭分享系统声音

调用 `TRTCCloud` 中的 stopSystemAudioLoopback 接口即可。

Mac

最近更新时间：2023-07-12 18:07:42

场景痛点及解决方案

在屏幕分享等应用场景中，常需要共享系统音频给对方，而 Mac 电脑默认声卡不支持采集系统音频，所以在 Mac 电脑上共享系统音频比较困难。基于此，TRTC 提供了在 Mac 端录制系统音频的功能来满足该场景需求，具体接入步骤见下文。

集成说明

步骤1：集成 TRTCPrivilegedTask 库

SDK 需要使用 TRTCPrivilegedTask 库来获取 root 权限，从而将虚拟声卡插件 TRTCAudioPlugin.driver 安装至系统目录 `/Library/Audio/Plug-Ins/HAL` 。

使用CocoaPods集成

1. 打开您当前项目根目录下的 `Podfile` 文件，添加下面的内容：

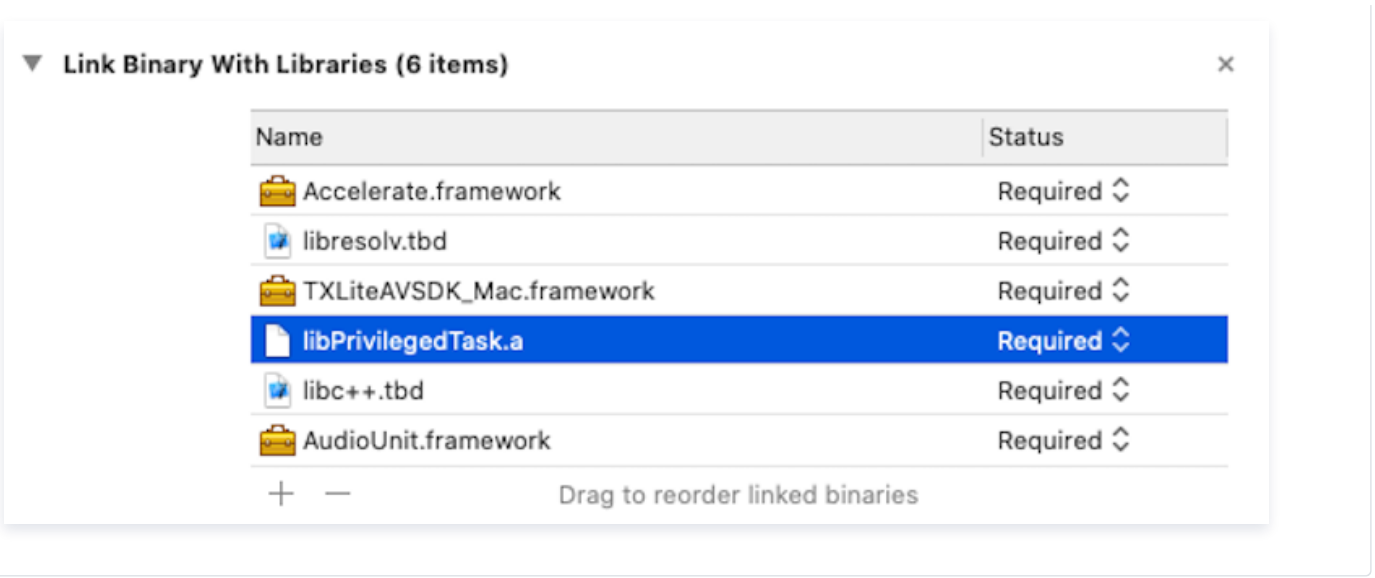
```
platform :osx, '10.10'

target 'Your Target' do
    pod 'TRTCPrivilegedTask', :podspec => 'https://pod-1252463788.cos.ap-guangzhou.myqcloud.com/liteavsdkspec/TRTCPrivilegedTask.podspec'
end
```

2. 执行 `pod install` 命令安装 TRTCPrivilegedTask 库。

> **说明**
> - 如果项目根目录下没有 `Podfile` 文件，请先执行 `pod init` 命令新建文件再添加以下内容。
> - CocoaPods 的安装方法，请参见 CocoaPods 官网安装说明 。

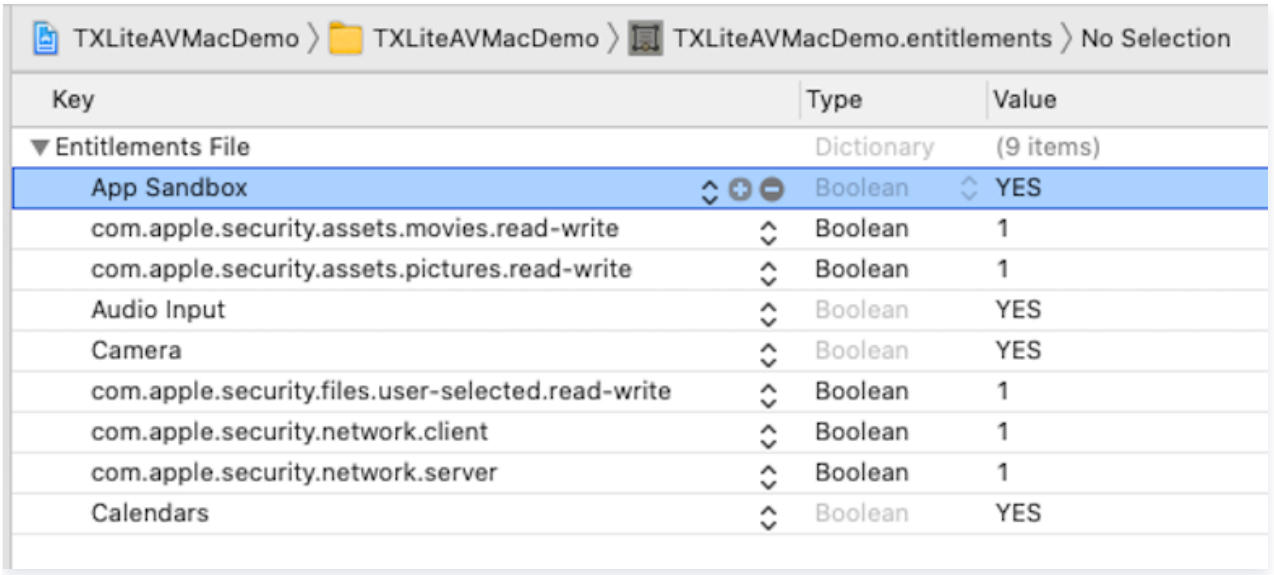
手动集成

1. 下载 TRTCPrivilegedTask 库。
2. 打开您的 Xcode 工程项目，导入解压后的文件 libPrivilegedTask.a 到您的工程。
3. 选择要运行的 target，选中 Build Phases 项，展开Link Binary with Libraries 项，单击底下的**+**，添加依赖库 `libPrivilegedTask.a` 。



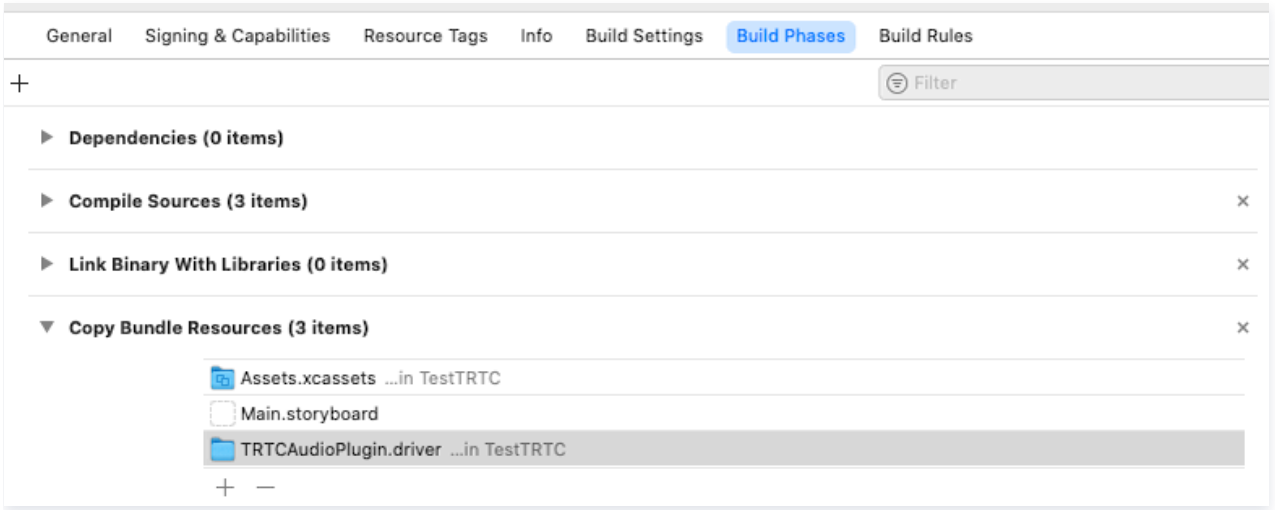
步骤2：取消 App Sandbox 功能

在 App 的 entitlements 描述文件中，删除 **App Sandbox** 条目。

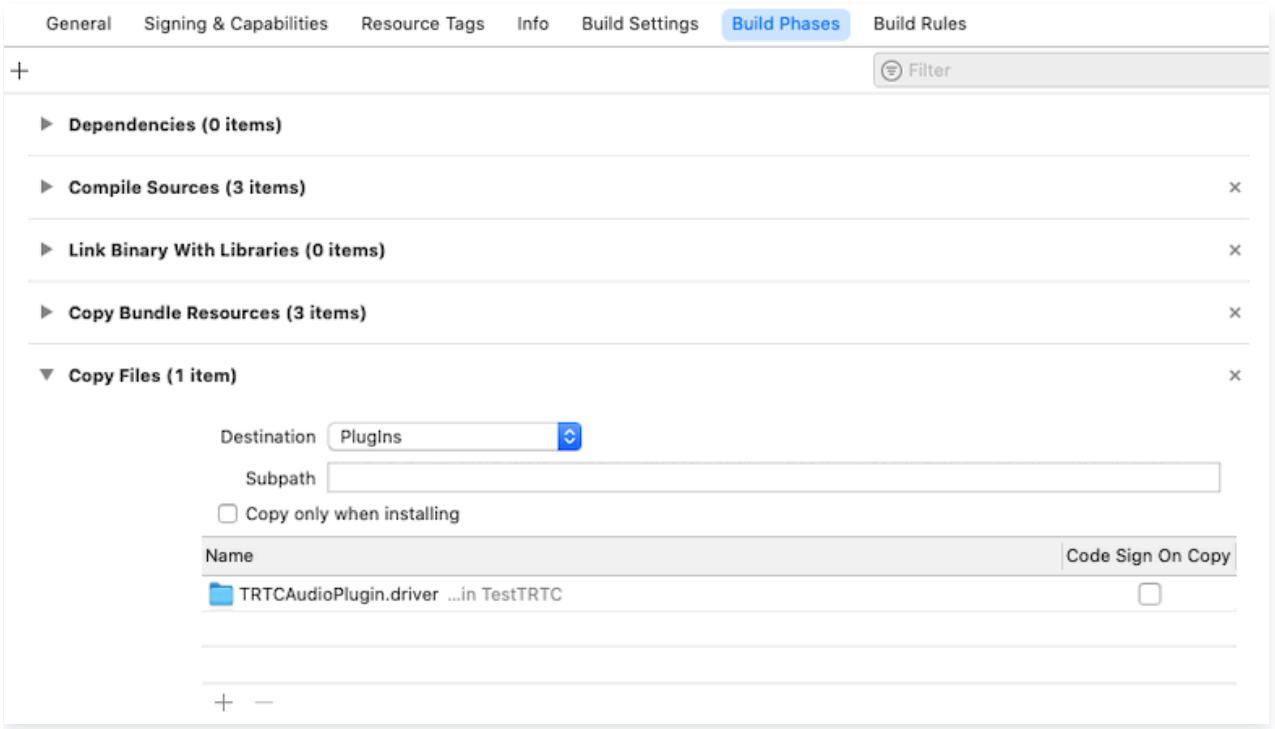


步骤3：虚拟声卡插件打包

在 集成 TRTCPrivilegedTask 库 和 取消 App Sandbox 功能 后，首次使用系统音频录制功能时，SDK 会从网络下载虚拟声卡插件并安装。如果想加速这个过程，可以将放置在 `TXLiteAVSDK_TRTC_Mac.framework` 的 PlugIns 目录下的虚拟声卡插件 `TRTCAudioPlugin.driver` 打包至 App Bundle 的 Resources 目录。如下图：



或者拷贝至 App Bundle 的 PlugIns 目录。如下图：



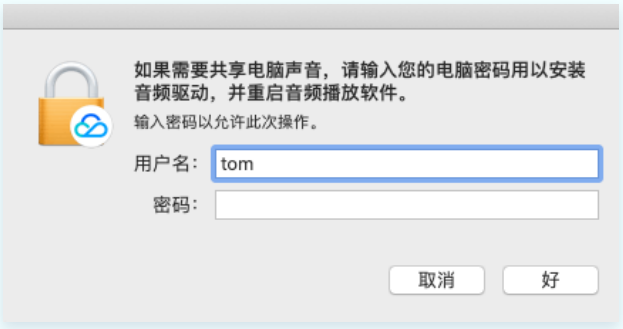
步骤4：开始系统声音采集

调用 startSystemAudioLoopback 接口开始系统声音采集，并将其混入上行音频流中，接口执行完成会通过 onSystemAudioLoopbackError 回调成功或失败的结果。

```
TRTCCloud *trtcCloud = [TRTCCloud sharedInstance];
[trtcCloud startLocalAudio];
[trtcCloud startSystemAudioLoopback];
```

注意

集成 TRTCPrivilegedTask 库和取消 App Sandbox 功能后，首次调用 startSystemAudioLoopback 会获取 root 权限。如下图：



在用户单击好后，开始自动安装虚拟声卡插件。

步骤5：停止系统声音采集

调用 stopSystemAudioLoopback 接口停止系统声音采集。

```
TRTCCloud *trtcCloud = [TRTCCloud sharedInstance];
[trtcCloud stopSystemAudioLoopback];
```

步骤6：设置系统声音采集音量

调用 setSystemAudioLoopbackVolume 接口设置系统声音的采集音量。

```
TRTCCloud *trtcCloud = [TRTCCloud sharedInstance];
[trtcCloud setSystemAudioLoopbackVolume:80];
```

集成小结

- TRTC 在 Mac 端是通过虚拟声卡插件 `TRTCAudioPlugin.driver` 来录制系统音频。这个虚拟声卡插件需要拷贝至系统目录 `/Library/Audio/Plug-Ins/HAL` 并重启音频服务后生效。可以通过 启动台 的 其他 文件夹中 音频 MIDI 设置 应用来检查虚拟声卡插件是否安装成功。在该应用的设备列表中，有名称为 “TRTC Audio Device” 的设备即表明 TRTC 的虚拟声卡插件安装成功。
- 上述步骤中的 集成 TRTCPrivilegedTask 库 和 取消 App Sandbox 功能 是为 TRTC SDK 自动安装虚拟声卡插件提供 root 权限。如不集成 TRTCPrivilegedTask 库并保留 App Sandbox 功能， SDK 不会自动安装虚拟声卡插件，但如果系统中已安装好虚拟声卡插件，录制系统音频的功能仍然可以正常使用。

> **说明**
> 除上述方案外，也可以手动安装虚拟声卡插件来集成该功能：
> 1. 将放置在 `TXLiteAVSDK_TRTC_Mac.framework` 的 PlugIns 目录下的 `TRTCAudioPlugin.driver` 拷贝至系统目录 `/Library/Audio/Plug-Ins/HAL` 。
> 2. 重启系统的音频服务。
>
> ```
> sudo cp -R TXLiteAVSDK_TRTC_Mac.framework/PlugIns/TRTCAudioPlugin.driver
> /Library/Audio/Plug-Ins/HAL
> sudo kill -9 `ps ax|grep 'coreaudio[a-z]' |awk '{print $1}'`
> ```

注意事项

- App Sandbox 功能取消后，App 内获取到的用户路径会发生变化。通过 NSSearchPathForDirectoriesInDomains 等系统方法获取到的 `~/Documents`、`~/Library` 等目录会从沙盒目录切换成用户目录 `/Users/用户名/Documents`、`/Users/用户名/Library`。
- 集成 TRTCPrivilegedTask 库，可能会使 App 无法上架到 Mac App Store。SDK 自动安装虚拟声卡插件时需要关闭 App Sandbox 功能，并获取 root 权限，App 可能会无法上架到 Mac App Store。详情请参见 App Store Review Guidelines 。如需上架 App Store 或者使用 Sandbox 功能，建议选择手动安装虚拟音频插件的方案。

Electron

最近更新时间：2023-09-18 18:08:02

场景痛点及解决方案

在屏幕分享等应用场景中，常需要共享系统音频给对方，而采用 Electron 打包 Mac 应用时，Mac 电脑默认声卡不支持采集系统音频，所以在 Mac 电脑上共享系统音频比较困难。基于此，TRTC 提供了在 Mac 端录制系统音频的功能来满足该场景需求，具体接入步骤见下文。

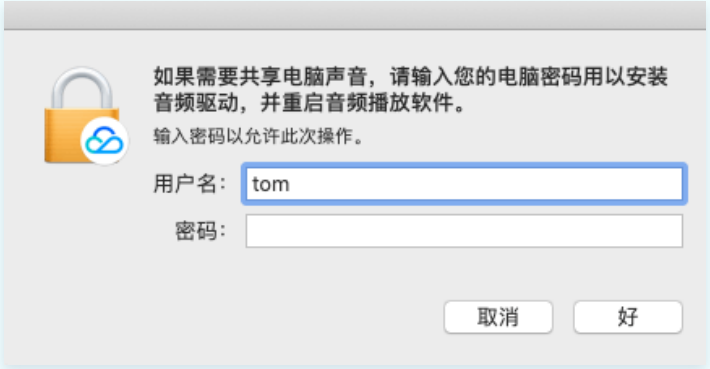
步骤1：开始系统声音采集

调用 startSystemAudioLoopback 接口开始系统声音采集，并将其混入上行音频流中，接口执行完成会通过 onSystemAudioLoopbackError 回调成功或失败的结果。

```
import TRTCCloud, { TRTCAudioQuality } from 'trtc-electron-sdk';
const rtcCloud = new TRTCCloud();
function onSystemAudioLoopbackError(errCode) {
  if (errCode === 0) {
    console.log('启动成功');
  }
  if (errCode === -1330) {
    console.log('开启系统声音录制失败，例如音频驱动插件不可用');
  }
  if (errCode === -1331) {
    console.log('未授权安装音频驱动插件');
  }
  if (errCode === -1332) {
    console.log('安装音频驱动插件失败');
  }
}

trtcCloud.on('onSystemAudioLoopbackError', onSystemAudioLoopbackError);
trtcCloud.startLocalAudio(TRTCAudioQuality.TRTCAudioQualityDefault);
trtcCloud.startSystemAudioLoopback();
```

注意：

首次调用 startSystemAudioLoopback 会获取 root 权限（如下图）， 在用户单击好后，开始自动安装虚拟声卡插件。



步骤2：停止系统声音采集

调用 stopSystemAudioLoopback 接口停止系统声音采集。

```
trtcCloud.stopSystemAudioLoopback();
```

步骤3：设置系统声音采集音量

调用 setSystemAudioLoopbackVolume 接口设置系统声音的采集音量。

```
trtcCloud.setSystemAudioLoopbackVolume(60);
```

集成小结

TRTC 在 Mac 端是通过虚拟声卡插件 `TRTCAudioPlugin.driver` 来录制系统音频。这个虚拟声卡插件需要拷贝至系统目录 `/Library/Audio/Plug-Ins/HAL` 并重启音频服务后生效。可以通过 `启动台` 的 `其他` 文件夹中 `音频 MIDI 设置` 应用来检查虚拟声卡插件是否安装成功。在该应用的设备列表中，有名称为 “TRTC Audio Device” 的设备即表明 TRTC 的虚拟声卡插件安装成功。

版权所有：腾讯云计算（北京）有限责任公司

iOS

最近更新时间：2024-08-05 14:18:41

本文档主要介绍如何分享 iOS 系统声音。

调用指引

步骤1: 开启麦克风

在 App 中，调用 startLocalAudio 开启麦克风采集，推荐使用 TRTCAudioQualityDefault 音质。

> ⚠ **注意：**
> 这一步是必要的，通过开启麦克风采集，可以让 App 在后台时依然保持运行。

步骤2: 开启屏幕分享

由于 iOS 限制，只有在屏幕录制时才能采集系统声音。因此，为实现此功能，需要先对接iOS屏幕分享功能。
按照 开启屏幕分享 中的步骤，开启屏幕录制，系统声音将自动进行采集。

> ⚠ **注意：**
> 开启屏幕录制时，不要点亮麦克风图标，人声采集已在 App 中启动。

步骤3: 中途开启和关闭系统声音

系统声音采集与屏幕录制同时进行，只能随录制开启而自动启动，随录制关闭而一起停止。无法单独对系统声音进行开关或静音操作。
TRTCCloud 提供了 setSystemAudioLoopbackVolume 方法对系统声音进行音量调节。当您不希望将系统声音输出时，可以将音量设置为0。

Web

最近更新时间：2023-10-25 14:51:31

功能描述

本文主要介绍如何在 TRTC Web SDK 实现分享系统音频功能。

> **说明：**
> - 本教程基于 5.x TRTC Web SDK 实现，若您使用 4.x 版本 SDK，可参考 此教程 。
> - 5.x 与 4.x 版本的区别 。

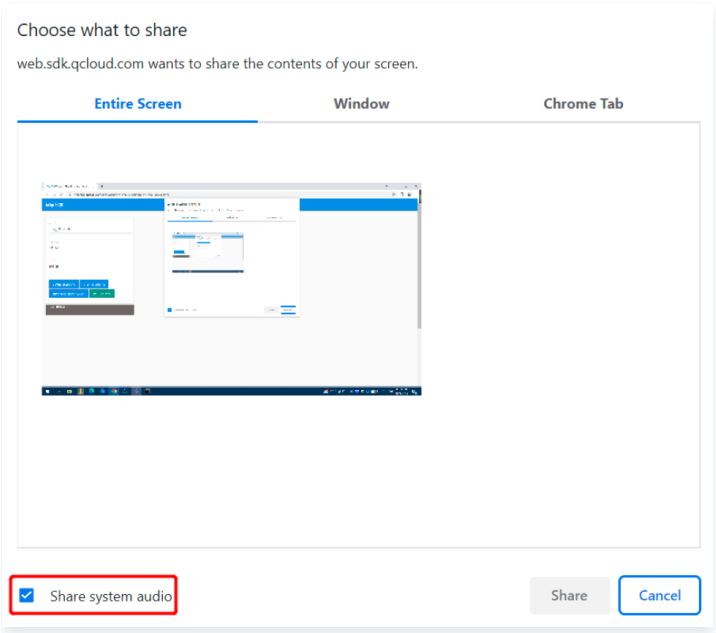
实现流程

Web 端实现分享系统音频，需要搭配屏幕分享一起使用。无法脱离屏幕分享，单独实现分享系统音频功能。

代码示例

```
await trtcA.startScreenShare({ option: { systemAudio: true }});
```

在弹出的屏幕分享选择框中勾选 **Share system audio**，点击 **Share** 即可。在进房发布后，房间内其他用户会收到 TRTC.EVENT.REMOTE_AUDIO_AVAILABLE 事件。



> **注意：**
> 若分享系统音频的同时采集了麦克风，则系统音频会与本地麦克风混音后发布。

兼容性说明

分享系统音频只支持基于 Chromium 74+ 版本的浏览器，例如 Chrome、Edge、Opera 等。其他浏览器暂不支持，例如：Safari、Firefox。

- Windows & Chrome OS 的 Chrome 支持分享系统音频 + 某个浏览器页面音频。
- MacOS & Linux 的 Chrome 只支持分享某个浏览器页面音频。

- Android & iOS 的 Chrome 不支持。

Flutter

最近更新时间：2024-02-01 21:16:21

本文档主要介绍如何分享系统声音，目前 TRTC 默认不采集所在应用的音频。

调用指引

Android

本文档主要介绍如何分享系统声音，目前TRTC默认不采集所在应用的音频。

> ⚠ **注意：**
> 仅 Android 10.0 及以上版本支持分享系统声音。

开启分享系统声音

步骤1: 开启屏幕分享

按照 开启屏幕分享-基于Android平台 中的步骤开启屏幕分享。

步骤2: 开启分享系统声音

调用 `TRTCCloud` 中的 startSystemAudioLoopback 接口即可，这时候采集的系统声音会自动混入上行流中。

步骤3: 关闭分享系统声音

调用 `TRTCCloud` 中的 stopSystemAudioLoopback 接口即可。

iOS

步骤1: 开启麦克风

在 App 中，调用 startLocalAudio 开启麦克风采集，推荐使用 TRTCAudioQualityDefault 音质。

> ⚠ **注意：**
> 这一步是必要的，通过开启麦克风采集，可以让 App 在后台时依然保持运行。

步骤2: 开启屏幕分享

由于 iOS 限制，只有在屏幕录制时才能采集系统声音。因此，为实现此功能，需要先对接iOS屏幕分享功能。
按照 开启屏幕分享-基于iOS平台 中的步骤，开启屏幕录制，系统声音将自动进行采集。

> ⚠ **注意：**
> 开启屏幕录制时，不要点亮麦克风图标，人声采集已在 App 中启动。

步骤3: 中途开启和关闭系统声音

系统声音采集与屏幕录制同时进行，只能随录制开启而自动启动，随录制关闭而一起停止。无法单独对系统声音进行开关或静音操作。TRTCCloud 提供了 setSystemAudioLoopbackVolume 方法对系统声音进行音量调节。当您不希望将系统声音输出时，可以将音量设置为0。

监听网络质量
Android&iOS&Windows&Mac

最近更新时间：2024-09-03 18:25:12

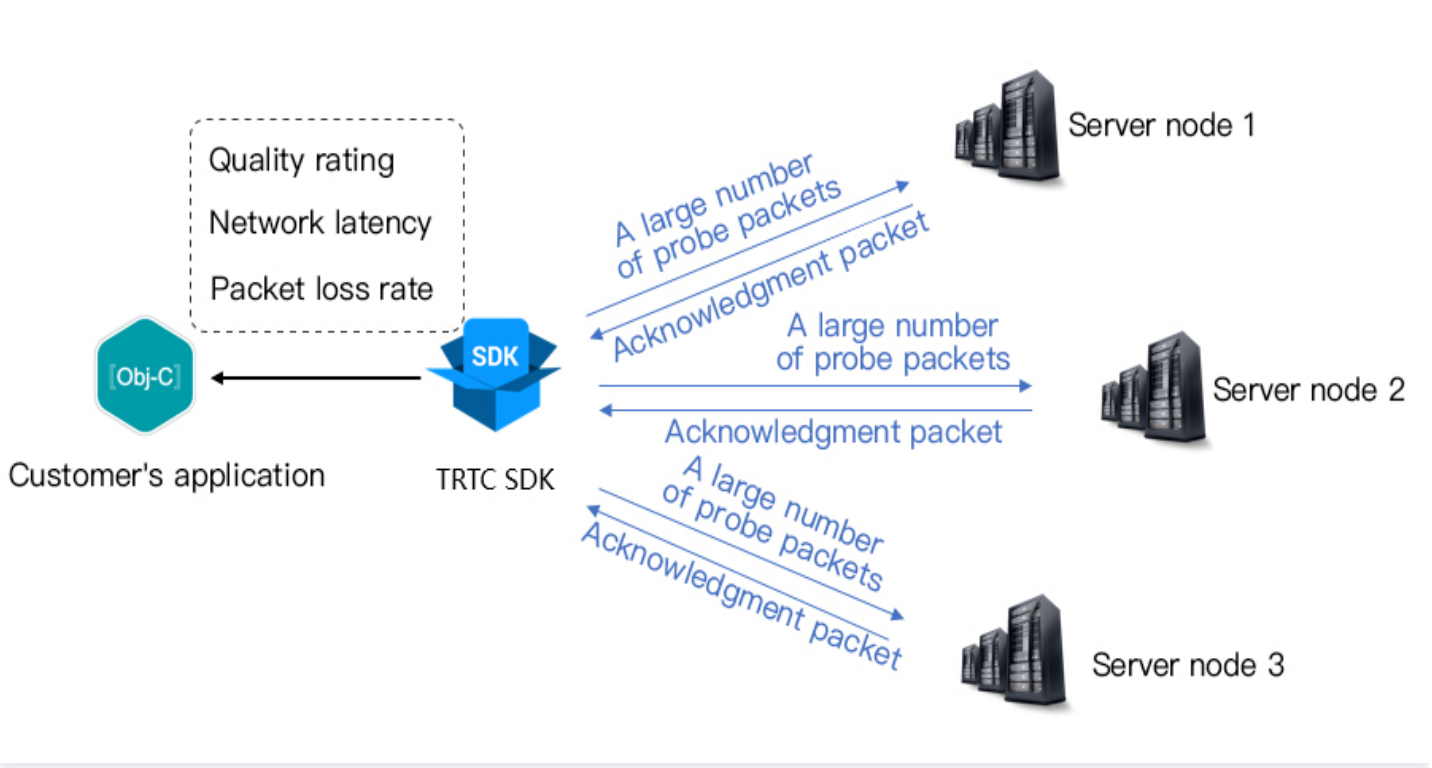
普通用户很难评估网络质量，通过视频通话前测速和通话中反馈网络质量两种方式，可以帮助用户更加直观地评估网络质量。

本文介绍了如何进行网络质量检测。

通话前网络质量检测

> **注意：**
> - 视频通话期间请勿测试，以免影响通话质量。
> - 测速本身会消耗一定的流量，从而产生极少量额外的流量费用（基本可以忽略）。

测速的原理



- 测速的原理是 SDK 向服务器节点发送一批探测包，然后统计回包的质量，并将测速的结果通过回调接口通知出来。
- 测速的结果将会用于优化 SDK 接下来的服务器选择策略，因此推荐您在用户首次通话前先进行一次测速，这将有助于我们选择最佳的服务器。同时，如果测试结果非常不理想，您可以通过醒目的 UI 提示用户选择更好的网络。
- 测速的结果（TRTCSpeedTestResult）包含如下几个字段：

字段	含义	含义说明
success	是否成功	本次测试是否成功

errMsg	错误信息	带宽测试的详细错误信息
ip	服务器 IP	测速服务器的 IP
quality	网络质量评分	通过评估算法测算出的网络质量，loss 越低，rtt 越小，得分也就越高
upLostRate	上行丢包率	范围是[0 – 1.0]，例如0.3代表每向服务器发送10个数据包，可能有3个会在中途丢失
downLostRate	下行丢包率	范围是[0 – 1.0]，例如0.2代表从服务器每收取10个数据包，可能有2个会在中途丢失
rtt	网络延时	代表 SDK 跟服务器一来一回之间所消耗的时间，这个值越小越好，正常数值在 10ms – 100ms 之间
availableUpBandwidth	上行带宽	预测的上行带宽，单位为kbps， -1表示无效值
availableDownBandwidth	下行带宽	预测的下行带宽，单位为kbps， -1表示无效值

测速方法

- **调用 API**

  通过调用 TRTCCloud 的 `startSpeedTest` 启动测速功能，测速结果将会通过回调函数返回。

Objective-C

```
// 启动网络测速的示例代码，需要 sdkAppId 和 UserSig，(获取方式参考基本功能)
// 这里以登录后开始测试为例
- (void)onLogin:(NSString *)userId userSig:(NSString *)userSid
{
    TRTCSpeedTestParams *params;
    // sdkAppID 为控制台中获取的实际应用的 AppID
    params.sdkAppID = sdkAppId;
    params.userID = userId;
    params.userSig = userSig;
    // 预期的上行带宽（kbps，取值范围： 10 ~ 5000，为 0 时不测试）
    params.expectedUpBandwidth = 5000;
    // 预期的下行带宽（kbps，取值范围： 10 ~ 5000，为 0 时不测试）
    params.expectedDownBandwidth = 5000;
    [trtcCloud startSpeedTest:params];
}
- (void)onSpeedTestResult:(TRTCSpeedTestResult *)result {
    // 测速完成后，返回测速结果
}
```

Java

```
//启动网络测速的示例代码，需要 sdkAppId 和 UserSig，(获取方式参考基本功能)
// 这里以登录后开始测试为例
```

```
public void onLogin(String userId, String userSig)
{
  TRTCCloudDef.TRTCSpeedTestParams params = new TRTCCloudDef.TRTCSpeedTestParams();
  params.sdkAppId = GenerateTestUserSig.SDKAPPID;
  params.userId = mEtUserId.getText().toString();
  params.userSig = GenerateTestUserSig.genTestUserSig(params.userId);
  params.expectedUpBandwidth = Integer.parseInt(expectUpBandwidthStr);
  params.expectedDownBandwidth = Integer.parseInt(expectDownBandwidthStr);
    // sdkAppID 为控制台中获取的实际应用的 AppID
    trtcCloud.startSpeedTest(params);
}

// 监听测速结果，继承 TRTCCloudListener 并实现如下方法
void onSpeedTestResult(TRTCCloudDef.TRTCSpeedTestResult result)
{
    // 测速完成后，会回调出测速结果
}
```

C++

```
// 启动网络测速的示例代码，需要 sdkAppId 和 UserSig，(获取方式参考基本功能)
// 这里以登录后开始测试为例
void onLogin(const char* userId, const char* userSig)
{
    TRTCSpeedTestParams params;
    // sdkAppID 为控制台中获取的实际应用的 AppID
    params.sdkAppID = sdkAppId;
    params.userId = userid;
    param.userSig = userSig;
    // 预期的上行带宽（kbps，取值范围： 10 ~ 5000，为 0 时不测试）
    param.expectedUpBandwidth = 5000;
    // 预期的下行带宽（kbps，取值范围： 10 ~ 5000，为 0 时不测试）
    param.expectedDownBandwidth = 5000;
    trtcCloud->startSpeedTest(params);
}

// 监听测速结果
void TRTCCloudCallbackImpl::onSpeedTestResult(
            const TRTCSpeedTestResult& result)
{
    // 测速完成后，会回调出测速结果
}
```

C#

```
// 启动网络测速的示例代码，需要 sdkAppId 和 UserSig，(获取方式参考基本功能)
```

```
// 这里以登录后开始测试为例
private void onLogin(string userId, string userSig)
{
    TRTCSpeedTestParams params;
    // sdkAppID 为控制台中获取的实际应用的 AppID
    params.sdkAppID = sdkAppId;
    params.userId = userid;
    param.userSig = userSig;
    // 预期的上行带宽（kbps，取值范围： 10 ~ 5000，为 0 时不测试）
    param.expectedUpBandwidth = 5000;
    // 预期的下行带宽（kbps，取值范围： 10 ~ 5000，为 0 时不测试）
    param.expectedDownBandwidth = 5000;
    mTRTCCloud.startSpeedTest(params);
}

// 监听测速结果
public void onSpeedTestResult(TRTCSpeedTestResult result)
{
    // 测速完成后，会回调出测速结果
}
```

- **测速工具**

  如果您不想通过调用接口的方式来进行网络测速，TRTC 还提供了桌面端的网络测速工具程序，帮助您快速获取详细的网络质量信息，您可以根据自己平台选择以下程序进行下载。
  该程序提供了**快速测试**和**持续测试**两种测试选择。
  Mac | Windows

指标	含义
WiFi Quality	Wi-Fi 信号质量
DNS RTT	腾讯云的测速域名解析耗时
MTR	MTR 是一款网络测试工具，能探测客户端到 TRTC 节点的丢包率与延时，还可以查看路由中每一跳的具体信息
UDP Loss	客户端到 TRTC 节点的 UDP 丢包率
UDP RTT	客户端到 TRTC 节点的 UDP 延时
Local RTT	客户端到本地网关的延时
Upload	上行预估带宽
Download	下行预估带宽

通话中网络质量检测

TRTC 提供了一个叫做 `onNetworkQuality` 的回调事件，它会每隔两秒钟一次向您汇报当前的网络质量，其参数包括 **localQuality** 和 **remoteQuality** 两个部分：

- **localQuality**：代表您当前的网络质量，分为 6 个等级，分别是 Excellent、Good、Poor、Bad、VeryBad 和 Down。
- **remoteQuality**：代表远端用户的网络质量，这是一个素组，素组中的每个元素代表一个远端用户的网络质量。

Quality	名称	说明
0	Unknown	未感知到
1	Excellent	当前网络非常好
2	Good	当前网络比较好
3	Poor	当前网络一般
4	Bad	当前网络较差，可能会出现明显的卡顿和通话延迟
5	VeryBad	当前网络很差，TRTC 只能勉强保持连接，但无法保证通讯质量
6	Down	当前网络不满足 TRTC 的最低要求，无法进行正常的音视频通话

通过对 `onNetworkQuality` 进行监听并在界面上做相应地提示即可：

Android

```
// 监听 onNetworkQuality 回调并感知当前网络状态的变化
@Override
public void onNetworkQuality(TRTCCloudDef.TRTCQuality localQuality,
                             ArrayList<trtcclouddef.trtcquality> remoteQuality)
{
    // Get your local network quality
    switch(localQuality) {
        case TRTCQuality_Unknown:
            Log.d(TAG, "SDK has not yet sensed the current network quality.");
            break;
        case TRTCQuality_Excellent:
            Log.d(TAG, "The current network is very good.");
            break;
        case TRTCQuality_Good:
            Log.d(TAG, "The current network is good.");
            break;
        case TRTCQuality_Poor:
            Log.d(TAG, "The current network quality barely meets the demand.");
            break;
        case TRTCQuality_Bad:
            Log.d(TAG, "The current network is poor, and there may be significant
freezes and call delays.");
            break;
        case TRTCQuality_VeryBad:
            Log.d(TAG, "The current network is very poor, the communication quality
cannot be guaranteed");
            break;
        case TRTCQuality_Down:
            Log.d(TAG, "The current network does not meet the minimum requirements.");
            break;
        default:
            break;
```

```
    }
    // Get the network quality of remote users
    for (TRTCCloudDef.TRTCQuality info : arrayList) {
        Log.d(TAG, "remote user : = " + info.userId + ", quality = " + info.quality);
    }
}
```

iOS & Mac

```
// 监听 onNetworkQuality 回调并感知当前网络状态的变化
- (void)onNetworkQuality:(TRTCQualityInfo *)localQuality remoteQuality:
(NSArray<trtcqualityinfo *=""> *)remoteQuality {
    // Get your local network quality
    switch(localQuality.quality) {
        case TRTCQuality_Unknown:
            NSLog(@"SDK has not yet sensed the current network quality.");
            break;
        case TRTCQuality_Excellent:
            NSLog(@"The current network is very good.");
            break;
        case TRTCQuality_Good:
            NSLog(@"The current network is good.");
            break;
        case TRTCQuality_Poor:
            NSLog(@"The current network quality barely meets the demand.");
            break;
        case TRTCQuality_Bad:
            NSLog(@"The current network is poor, and there may be significant freezes
and call delays.");
            break;
        case TRTCQuality_VeryBad:
           NSLog(@"The current network is very poor, the communication quality cannot be
guaranteed");
            break;
        case TRTCQuality_Down:
            NSLog(@"The current network does not meet the minimum requirements.");
            break;
        default:
            break;
    }
    // Get the network quality of remote users
    for (TRTCQualityInfo *info in arrayList) {
        NSLog(@"remote user : = %@, quality = %@", info.userId, @(info.quality));
    }
}
```

Windows

```
// 监听 onNetworkQuality 回调并感知当前网络状态的变化
void onNetworkQuality(liteav::TRTCQualityInfo local_quality,
    liteav::TRTCQualityInfo* remote_quality, uint32_t remote_quality_count) {
    // Get your local network quality
    switch (local_quality.quality) {
    case TRTCQuality_Unknown:
        printf("SDK has not yet sensed the current network quality.");
        break;
    case TRTCQuality_Excellent:
        printf("The current network is very good.");
        break;
    case TRTCQuality_Good:
        printf("The current network is good.");
        break;
    case TRTCQuality_Poor:
        printf("The current network quality barely meets the demand.");
        break;
    case TRTCQuality_Bad:
        printf("The current network is poor, and there may be significant freezes and
call delays.");
        break;
    case TRTCQuality_Vbad:
        printf("The current network is very poor, the communication quality cannot be
guaranteed");
        break;
    case TRTCQuality_Down:
        printf("The current network does not meet the minimum requirements.");
        break;
    default:
        break;
    }
    // Get the network quality of remote users
    for (int i = 0; i < remote_quality_count; ++i) {
        printf("remote user : = %s, quality = %d", remote_quality[i].userId,
remote_quality[i].quality);
    }
}
```

Web

最近更新时间：2023-08-16 16:41:02

在进房之前或在通话过程中，检测用户的网络质量，可以判断用户当下的网络质量情况。若用户网络质量太差，应建议用户检查网络或尝试更换网络，以保证正常通话质量。
本文主要介绍如何基于 NETWORK_QUALITY 事件实现通话前网络质量检测。通话过程中感知网络质量，只需监听 NETWORK_QUALITY 事件即可。

> **说明：**
> - 本教程基于 5.x TRTC Web SDK 实现，若您使用 4.x 版本 SDK，可参考 此教程 。
> - 5.x 与 4.x 版本的区别 。

通话过程中的网络质量检测

```
const trtc = TRTC.create();
trtc.on(TRTC.EVENT.NETWORK_QUALITY, event => {
   console.log(`network-quality, uplinkNetworkQuality:${event.uplinkNetworkQuality},
downlinkNetworkQuality: ${event.downlinkNetworkQuality}`)
   console.log(`uplink rtt:${event.uplinkRTT} loss:${event.uplinkLoss}`)
   console.log(`downlink rtt:${event.downlinkRTT} loss:${event.downlinkLoss}`)
})
```

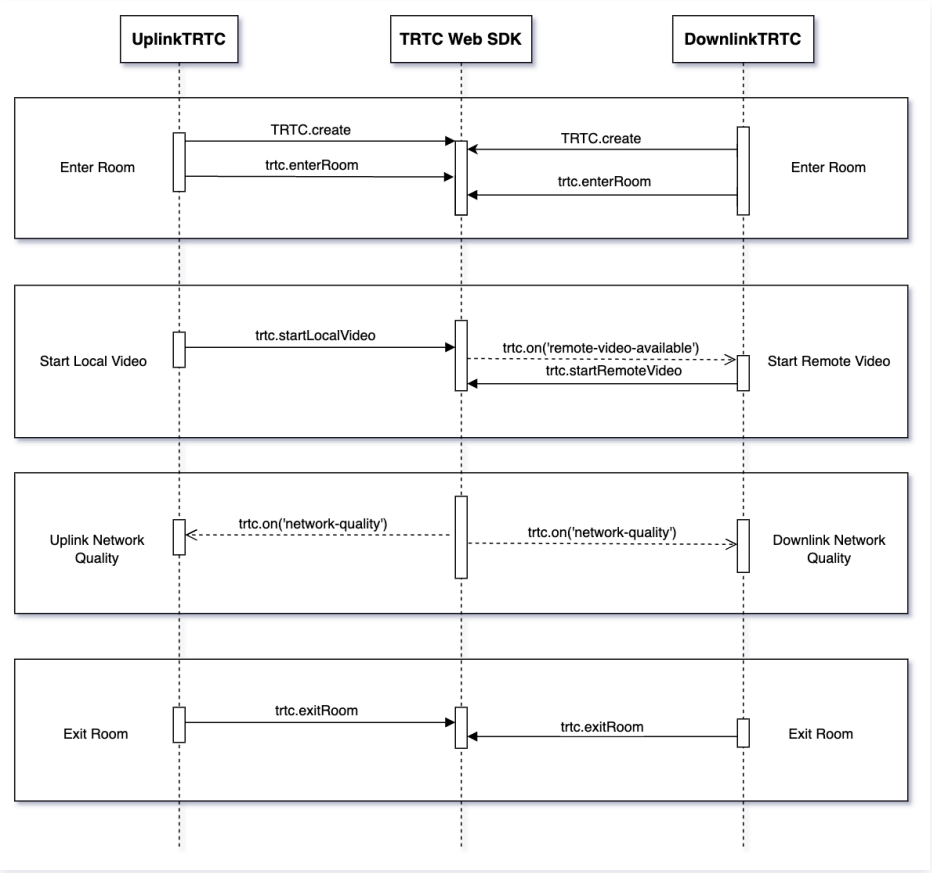
通话前的网络质量检测

实现流程

1. 调用 `TRTC.create()` 方法创建两个 `TRTC` ，分别称为 `uplinkTRTC` 和 `downlinkTRTC` 。
2. 两个 `TRTC` 都进入同一个房间。
3. 使用 `uplinkTRTC` 进行推流，监听 `TRTC.EVENT.NETWORK_QUALITY` 事件来检测上行网络质量。
4. 使用 `downlinkTRTC` 进行拉流，监听 `TRTC.EVENT.NETWORK_QUALITY` 事件来检测下行网络质量。
5. 整个过程可持续15s左右，最后取平均网络质量，从而大致判断出上下行网络情况。

> **注意：**
> 检测过程将产生少量的 基础服务费用 。如果未指定推流分辨率，则默认以640*480的分辨率推流。

API 调用时序



代码示例

```
let uplinkTRTC = null; // 用于检测上行网络质量
let downlinkTRTC = null; // 用于检测下行网络质量
let localStream = null; // 用于测试的流
let testResult = {
  // 记录上行网络质量数据
  uplinkNetworkQualities: [],
  // 记录下行网络质量数据
  downlinkNetworkQualities: [],
  average: {
    uplinkNetworkQuality: 0,
    downlinkNetworkQuality: 0
  }
}

// 1. 检测上行网络质量
async function testUplinkNetworkQuality() {
  uplinkTRTC = TRTC.create();

  uplinkTRTC.enterRoom({
    roomId: 8080,
    sdkAppId: 0, // 填写 sdkAppId
    userId: 'user_uplink_test',
    userSig: '', // uplink_test 的 userSig
    scene: 'rtc'
  })
```

```
  uplinkTRTC.on(TRTC.EVENT.NETWORK_QUALITY, event => {
    const { uplinkNetworkQuality } = event;
    testResult.uplinkNetworkQualities.push(uplinkNetworkQuality);
  });

}

// 2. 检测下行网络质量
async function testDownlinkNetworkQuality() {
  downlinkTRTC = TRTC.create();
  downlinkTRTC.enterRoom({
    roomId: 8080,
    sdkAppId: 0, // 填写 sdkAppId
    userId: 'user_downlink_test',
    userSig: '', // userSig
    scene: 'rtc'
  });

  downlinkTRTC.on(TRTC.EVENT.NETWORK_QUALITY, event => {
      const { downlinkNetworkQuality } = event;
      testResult.downlinkNetworkQualities.push(downlinkNetworkQuality);
  })
}

// 3. 开始检测
testUplinkNetworkQuality();
testDownlinkNetworkQuality();

// 4. 15s 后停止检测，计算平均网络质量
setTimeout(() => {
  // 计算上行平均网络质量
  if (testResult.uplinkNetworkQualities.length > 0) {
    testResult.average.uplinkNetworkQuality = Math.ceil(
      testResult.uplinkNetworkQualities.reduce((value, current) => value + current, 0) /
testResult.uplinkNetworkQualities.length
    );
  }

  if (testResult.downlinkNetworkQualities.length > 0) {
    // 计算下行平均网络质量
    testResult.average.downlinkNetworkQuality = Math.ceil(
      testResult.downlinkNetworkQualities.reduce((value, current) => value + current, 0) /
testResult.downlinkNetworkQualities.length
    );
  }

  // 检测结束，清理相关状态。
  uplinkTRTC.exitRoom();
  downlinkTRTC.exitRoom();
}, 15 * 1000);
```

结果分析

经过上述步骤，可以拿到上行平均网络质量、下行平均网络质量。网络质量的枚举值如下所示：

数值	含义
0	网络状况未知，表示当前 TRTC 实例还没有建立上行/下行连接
1	网络状况极佳
2	网络状况较好
3	网络状况一般
4	网络状况差
5	网络状况极差
6	网络连接已断开 **注意：若下行网络质量为此值，则表示所有下行连接都断开了**

建议：当网络质量大于3时，应引导用户检查网络并尝试更换网络环境，否则难以保证正常的音视频通话。
也可通过下述策略来降低带宽消耗：

- **若上行网络质量大于3，则可通过 TRTC.updateLocalVideo() 接口降低码率或 TRTC.stopLocalVideo() 方式关闭视频，以降低上行带宽消耗。**
- **若下行网络质量大于3，则可通过订阅小流（参考： 开启大小流传输 ）或者只订阅音频的方式，以降低下行带宽消耗。**

Electron

最近更新时间：2023-09-18 18:08:02

本文档主要介绍如何感知当前网络的好与坏。
当我们在使用微信视频通话的时候，如果遇到网络环境较差的情况（比如在进入电梯以后），微信会在视频通话的界面上提示”您当前的网络质量较差“。本文档主要介绍如何通过 TRTC 完成同样的交互。

调用指引
TRTC 提供了一个叫做 **onNetworkQuality** 的回调事件，它会每隔两秒钟一次向您汇报当前的网络质量，其参数包括 localQuality 和 remoteQuality 两个部分：

- **localQuality**：代表您当前的网络质量，分为 6 个等级，分别是 Excellent、Good、Poor、Bad、VeryBad 和 Down。
- **remoteQuality**：代表远端用户的网络质量，这是一个素组，素组中的每个元素代表一个远端用户的网络质量。

Quality	名称	说明
0	Unknown	未感知到
1	Excellent	当前网络非常好
2	Good	当前网络比较好
3	Poor	当前网络一般
4	Bad	当前网络较差，可能会出现明显的卡顿和通话延迟
5	VeryBad	当前网络很差，TRTC 只能勉强保持连接，但无法保证通讯质量
6	Down	当前网络不满足 TRTC 的最低要求，无法进行正常的音视频通话

您只需要监听 TRTC 的 onNetworkQuality 并在界面上做相应地提示即可：

```
import TRTCCloud, { TRTCQuality } from 'trtc-electron-sdk';
```

```
const rtcCloud = new TRTCCloud();

function onNetworkQuality(localQuality, remoteQuality) {
  switch(localQuality.quality) {
    case TRTCQuality.TRTCQuality_Unknown:
      console.log('SDK has not yet sensed the current network quality.');
      break;
    case TRTCQuality.TRTCQuality_Excellent:
      console.log('The current network is very good.');
      break;
    case TRTCQuality.TRTCQuality_Good:
      console.log('The current network is good.');
      break;
    case TRTCQuality.TRTCQuality_Poor:
      console.log('The current network quality barely meets the demand.');
      break;
    case TRTCQuality.TRTCQuality_Bad:
      console.log('The current network is poor, and there may be significant freezes and
call delays.');
      break;
    case TRTCQuality.TRTCQuality_Vbad:
      console.log('The current network is very poor, the communication quality cannot be
guaranteed.');
      break;
    case TRTCQuality.TRTCQuality_Down:
      console.log('The current network does not meet the minimum requirements.');
      break;
    default:
      break;
  }
  for (let i = 0; i < remoteQuality.length; i++) {
    console.log(`remote user: ${remoteQuality[i].userId}, quality:
${remoteQuality[i].quality}`);
  }
}

rtcCloud.on('onNetworkQuality', onNetworkQuality);
```

Flutter

最近更新时间：2024-02-21 15:49:01

本文档主要介绍如何感知当前网络的好与坏。
当我们在使用微信视频通话的时候，如果遇到网络环境较差的情况（例如在进入电梯以后），微信会在视频通话的界面上提示“您当前的网络质量较差”。本文档主要介绍如何通过 TRTC 完成同样的交互。

调用指引

TRTC 提供了一个叫做 **onNetworkQuality** 的回调事件，它会每隔两秒钟一次向您汇报当前的网络质量，其参数包括 localQuality 和 remoteQuality 两个部分：

- **localQuality**：代表您当前的网络质量，分为 6 个等级，分别是 Excellent、Good、Poor、Bad、VeryBad 和 Down。
- **remoteQuality**：代表远端用户的网络质量，这是一个素组，素组中的每个元素代表一个远端用户的网络质量。

Quality	名称	说明
0	Unknown	未感知到
1	Excellent	当前网络非常好
2	Good	当前网络比较好
3	Poor	当前网络一般
4	Bad	当前网络较差，可能会出现明显的卡顿和通话延迟

5	VeryBad	当前网络很差，TRTC 只能勉强保持连接，但无法保证通讯质量
6	Down	当前网络不满足 TRTC 的最低要求，无法进行正常的音视频通话

您只需要监听 TRTC 的 onNetworkQuality 并在界面上做相应地提示即可：

```
// 监听 onNetworkQuality 回调并感知当前网络状态的变化
if (type == TRTCCloudListener.onNetworkQuality) {
    if (type == TRTCCloudDef.TRTC_QUALITY_UNKNOWN) {
      // TODO
    } else if (type == TRTCCloudDef.TRTC_QUALITY_Excellent) {
      // TODO
    } else if (type == TRTCCloudDef.TRTC_QUALITY_Good) {
      // TODO
    } else if (type == TRTCCloudDef.TRTC_QUALITY_Poor) {
      // TODO
    } else if (type == TRTCCloudDef.TRTC_QUALITY_Bad) {
      // TODO
    } else if (type == TRTCCloudDef.TRTC_QUALITY_Vbad) {
      // TODO
    } else if (type == TRTCCloudDef.TRTC_QUALITY_Down) {
      // TODO
    }
    // Get the network quality of remote users
    for (var info in param['remoteQuality']) {
        // TODO
    }
}
```

常见问题
全平台

最近更新时间：2023-08-16 16:41:02

什么是 UserSig?

UserSig 是腾讯云设计的一种安全保护签名，目的是为了阻止恶意攻击者盗用您的云服务使用权。
目前，腾讯云的实时音视频（TRTC）、即时通信（IM）以及移动直播（MLVB）等服务都采用了该套安全保护机制。要使用这些服务，您都需要在相应 SDK 的初始化或登录函数中提供 SDKAppID，UserID 和 UserSig 三个关键信息。
其中 SDKAppID 用于标识您的应用，UserID 用于标识您的用户，而 UserSig 则是基于前两者计算出的安全签名，它由 HMAC SHA256 加密算法计算得出。只要攻击者不能伪造 UserSig，就无法盗用您的云服务流量。
UserSig 的计算原理如下图所示，其本质就是对 SDKAppID、UserID、ExpireTime 等关键信息进行了一次哈希加密：

```
//UserSig 计算公式，其中 secretkey 为计算 usersig 用的加密密钥

usersig = hmacsha256(secretkey, (userid + sdkappid + currtime + expire +
                                 base64(userid + sdkappid + currtime + expire)))
```

说明

- `currtime` 为当前系统的时间，`expire` 为签名过期的时间。
- 如需了解 UserSig 具体计算获取方法，请参见 UserSig 详情说明 。

实时音视频最多可以同时创建多少个房间?

支持同时并发存在4294967294个房间，累计房间数量无限制。

实时音视频延时大约多少?

全球端到端平均延时小于300ms。

实时音视频接入 PC 端是否支持屏幕分享功能?

支持，您可以参考如下文档：

- 屏幕分享（Windows）
- 屏幕分享（Mac）
- 屏幕分享（Web）

屏幕分享接口详情请参见 Windows（C++）API 或 Windows（C#）API 。另外，您也可以使用 Electron 接口 。

TRTC 支持哪些平台?

支持的平台包括 iOS、Android、Windows(C++)、Windows(C#)、Mac、Web、Electron、微信小程序，更多详情请参见 平台支持 。

Web 端有哪些常见问题?

请参见：Web 端常见问题 。

实时音视频最多可以支持多少个人同时通话?

- 通话模式下，单个房间最多支持300人同时在线，最多支持50人同时开启摄像头或麦克风。

- 直播模式下，单个房间支持10万人以观众身份在线观看，最多支持50人以主播身份开启摄像头或麦克风。

TRTC 怎么实现直播场景类应用?

TRTC 专门针对在线直播场景推出了10万人低延时互动直播解决方案，能保证主播与连麦主播的最低延时到200ms，普通观众的延时在1s以内，并且超强的抗弱网能力适应移动端复杂的网络环境。
具体操作指引请参考 跑通直播模式 。

TRTC 直播支持什么角色? 有什么区别?

直播场景（TRTCAppSceneLIVE 和 TRTCAppSceneVoiceChatRoom）支持 TRTCRoleAnchor（主播）和 TRTCRoleAudience（观众）两种角色，区别是主播角色可以同时上行、下行音视频数据，观众角色只支持下行播放其他人的数据。您可以通过调用 switchRole() 进行角色切换。

TRTC 房间支不支持踢人、禁止发言、静音?

支持。

- 如果是简单的信令操作，可以使用 TRTC 的自定义信令接口 sendCustomCmdMsg，开发者自己定义相应的控制信令，收到控制信令的通话方执行对应操作即可。例如，踢人就是定义一个踢人的信令，收到此信令的用户就自行退出房间。
- 如果是需要实现更完善的操作逻辑，建议开发者通过 即时通信 IM 来实现相关逻辑，将 TRTC 的房间与 IM 群组进行映射，在 IM 群组中收发自定义消息来实现相应的操作。

TRTC 音视频流是否支持通过 CDN 拉流观看?

支持，详情请参见 实现 CDN 直播观看 。

在 iOS 端是否支持 Swift 集成?

支持，直接按照支持集成三方库的流程集成 SDK 即可，还可以参考 跑通 Demo（iOS&Mac） 。

直播、互动直播、实时音视频以及旁路直播有什么区别和关系?

- **直播**（关键词：一对多，RTMP/HLS/HTTP-FLV，CDN）
  直播分为推流端、播放端以及直播云服务，云服务使用 CDN 进行直播流的分发。推流使用的是通用标准的协议 RTMP，经过 CDN 分发后，播放时一般可以选择 RTMP、HTTP-FLV 或 HLS（H5 支持）等方式进行观看。
- **互动直播**（关键词：连麦、PK）
  互动直播是一种业务形式，指主播与观众之间进行互动连麦，主播与主播之间进行互动PK的一种直播类型。
- **实时音视频**（关键词：多人互动，UDP 私有协议，低延时）
  实时音视频（Tencent Real-Time Communication，TRTC）主要应用场景是音视频互动和低延时直播，使用基于 UDP 的私有协议，其延迟可低至100ms，典型的场景就是 QQ 电话、腾讯会议、大班课等。 腾讯云实时音视频（TRTC）覆盖全平台，除了 iOS/Android/Windows 之外，还支持小程序以及 WebRTC 互通，并且支持通过云端混流的方式将画面旁路直播到 CDN。
- **旁路直播**（关键词：云端混流，RTC 旁路转推，CDN）
  旁路直播是一种技术，指的是将低延时连麦房间里的多路推流画面复制出来，在云端将画面混合成一路，并将混流后的画面推流给直播 CDN 进行分发播放。

TRTC 如何查看通话时长和使用量?

可在实时音视频控制台的 用量统计 页面查看。

TRTC 出现卡顿怎么排查?

可以通过对应的 RoomID、UserID 在实时音视频控制台的“监控仪表盘”页面查看通话质量：

- 通过接收端视角查看发送端和接收端用户情况。
- 查看发送端和接收端是否丢包率比较高，如果丢包率过高一般是网络状况不稳定导致卡顿。
- 查看帧率和 CPU 占用率，帧率比较低和 CPU 使用率过高都会导致卡顿现象。

版权所有：腾讯云计算（北京）有限责任公司

TRTC 出现画质不佳，模糊、马赛克等现象怎么排查?

- 清晰度主要和码率有关，检查 SDK 码率是否配置的比较低，如果高分辨率低码率容易产生马赛克现象。
- TRTC 会通过云端 QOS 流控策略，根据网络状况动态调整码率、分辨率，网络比较差时容易降低码率导致清晰度下降。
- 检查进房时使用的 VideoCall 模式还是 Live 模式，针对通话场景 VideoCall 模式主打低延时和保流畅，所以在弱网情况下会更容易牺牲画质确保流畅，对画质更加看重的场景建议使用 Live 模式。

如何查询 SDK 最新版本号?

- 若您使用自动加载的方法，`latest.release` 为匹配最新版并进行自动加载，不需要对版本号进行修改。具体集成方法请参见 一分钟集成 SDK 。
- 当前 SDK 最新版本号可通过发布日志查看，具体请参见：
  - iOS & Android 端，请参见 发布日志（App）。
  - Web 端，请参见 发布日志（Web）。
  - Electron 端，请参见 发布日志（Electron）。

小程序

最近更新时间：2025-01-03 14:56:33

环境问题

小程序的环境要求是怎样的?

- 微信 App iOS 最低版本要求：7.0.9
- 微信 App Android 最低版本要求：7.0.8
- 小程序基础库最低版本要求：2.10.0
- 由于小程序测试号不具备 <live-pusher> 和 <live-player> 的使用权限，请使用企业小程序账号申请相关权限进行开发。
- 由于微信开发者工具不支持原生组件（即 <live-pusher> 和 <live-player> 标签），需要在真机上进行运行体验。
- 不支持 uniapp 开发环境，请使用原生小程序开发环境。

更多详情请参见 快速集成(小程序) 。

小程序端如果需要上线或者部署正式环境怎么办?

1. 请申请域名并做好备案工作。
2. 请将服务端代码部署到申请的服务器上。
3. 请将推流域名及 IM 受信域名 配置到小程序控制台 request 合法域名里面：
   - `https://official.opensso.tencent-cloud.com`
   - `https://yun.tim.qq.com`
   - `https://cloud.tencent.com`
   - `https://webim.tim.qq.com`
   - `https://query.tencent-cloud.com`
   - `wss://wss.im.qcloud.com`
   - `wss://wss.tim.qq.com`
   - `https://web.sdk.qcloud.com`

小程序是否支持uniapp、taro等开发环境?

TUICalling 、TUIVoiceRoom 等原生组件暂不支持。

小程序端可以禁用重力感应吗?

小程序暂未开放重力感应设置接口。

小程序支持做最小化悬浮窗吗?

目前在页面存在 mode='RTC' 的 live-pusher 和至少一个 live-player 时，小程序在后台运行的情况下可以正常采集和播放音频，否则小程序在切后台时会终止音视频通话。

集成问题

小程序端有没有区分退房事件类型的，例如主动退房、被踢、解散房间?

EVENT.KICKED_OUT 表示服务端踢人或房间被解散退房，LOCAL_LEAVE 表示本地退房。

小程序端横屏推流的时候，为什么画面被裁剪了?

iOS 端可以通过打开竖屏方向锁定，Android 端暂时没有办法规避，需要小程序底层来修改。

版权所有：腾讯云计算（北京）有限责任公司

小程序网络波动通过哪个值来判断?

通过 netQualityLevel 来判断:

- 0：未定义
- 1：最好
- 2：好
- 3：一般
- 4：差
- 5：很差
- 6：不可用

小程序端和 Web 端支持自定义流 ID 吗?

- 小程序端从微信7.0.12版本开始支持自定义流 ID，在 rtcConfig 构造时字段填写自定义流 ID，具体可以参见 Demo 实现。
- Web 端4.3.8以上版本已支持自定义流 ID，在 createClient 时自定义该字段。

小程序端调试时为什么要开启调试模式?

开启调试后，可以略过把“request 合法域名”加入小程序白名单的操作，避免遇到登录失败，通话无法连接的问题。

小程序端为什么会出现黑屏/画面卡住?

您可以检查小程序 Demo 左下方的控制面板，打开**调试模式**即可在界面上看到详细的推拉流信息，如果没有推拉流信息则表示未成功进房或 live-pusher，live-player 创建失败。

小程序端集成实时音视频 SDK 前需要做哪些准备工作?

1. 创建腾讯云实时音视频应用，购买相应的套餐，并获取到 SDKAppID 和密钥信息。
2. 小程序服务器域名配置。
3. 开通小程序类目与推拉流标签权限。（如不开通则无法正常使用）
   出于政策和合规的考虑，微信暂未放开所有小程序对实时音视频功能（即 <live-pusher> 和 <live-player> 标签）的支持:
   - 小程序推拉流标签不支持个人小程序，只支持企业类小程序。
   - 小程序推拉流标签使用权限暂时只开放给有限 类目 。

- 符合类目要求的小程序，需要在 微信公众平台> 开发 > 接口设置中自助开通该组件权限，如下图所示：

更多详情请参见 跑通 Demo(小程序) 和 快速集成(小程序) 。

小程序端进入多人音视频看不到画面，该如何处理?

1. 请使用手机真机运行，微信开发者工具内部的模拟器目前暂不支持直接运行。
2. 请通过 wx.getSystemInfo 查询小程序基础库版本，小程序基础库最低版本要求为2.10.0。
3. 请确认小程序所属的类目，由于监管要求，小程序推拉流标签使用权限暂时只开放给有限 类目 。

如有更多需求，或希望深度合作，可以 提交工单 或致电4009100100联系我们。

小程序端运行出错，该如何排查?

1. 请检查开通的小程序类目是否正确，<live-pusher> 和 <live-player> 标签是否已开启。
2. 请确认已将 小程序域名白名单 添加到小程序 request 合法域名，或已开启调试模式。
3. 请重新解压小程序端 Demo 直接运行，若运行正常，建议参考 快速集成(小程序) 重新集成 SDK。
4. 若问题依然存在，可以登录 微信小程序开发者社区 查找相关资料，也可以 提交工单 或致电 4009100100 联系我们。

<live-pusher> 和 <live-player> 标签使用及错误码?

- live-pusher 错误码
- live-player 错误码
- livePusherContext
- livePlayerContext

是否能监听小程序缩小到后台?

可以。监听小程序的 onHide 方法，即可查看用户是否缩小到后台。

为什么拨打不通，或者被踢下线?

组件暂不支持多实例登入，不支持**离线推送信令**功能，请您确认账号登入的唯一性。

- 多实例：一个 userID 重复登入，或在不同端登入，将会引起信令的混乱。
- 离线推送：实例在线才能接收消息，实例离线时接收到的信令不会在上线后重新推送。即，小程序在后台与离线状态下，无法收到呼入提醒或来电提醒。

怎么区分主播观众?

在接入侧不需要设置主播/观众身份，SDK本身是通过是否有上行流来区分的，pusherAttributes的属性中 `enableCamera || enableMic` 为 true 的情况下是主播，音视频都没有上行的情况下是观众。

小程序怎么接收 SEI 消息?

可以参照 微信小程序 API LivePusherContext.sendMessage 进行 SEI 发送，通过监听 [live-player] bindstatechange 2012 事件进行接收，代码参考如下：

```
playerStateChange(e) {
  const { code, message } = e.detail
  console.log('playerStateChange', code, e)
  switch (code) {
    case 2012:
    {
      wx.showToast({ title: `收到SEI消息:${message}`, icon: 'none' })
      console.log('收到SEI消息: ', code, message)
      break
    }
  }
},
```

小程序怎么使用字符串房间号进房?

在进房时使用 strRoomID ，该参数的优先级会高于 roomID。

怎么播放背景音乐?

pusherInstance 中提供了 playBGM 的方法，具体请参见 pusherInstance 。如果使用微信的原生标签进行播放可能会在某些机型上不兼容，导致音量模式异常或听筒扬声器播放的异常。

怎样在小程序端暂停推流?

在小程序端，音频流与视频流可分别进行单独的暂停推流设置， pusherInstance 中相关方法如下：

- 暂停视频流：pusherInstance 中提供了 pause 的方法，调用这个接口，远端会显示黑屏帧。
- 暂停音频流：pusherInstance 中提供了 setMICVolume 的方法，调用这个接口，可以将本地采集的音量设为0，远端不会收到声音。

例如在需要同时暂停两个流的场景，可以在调用 pause 暂停视频流的同时，调用 setMICVolume 将音量设为 0，暂停音频流。
以上两个方法均会保留摄像头和麦克风设备的占用，是推荐的暂停推流的方式。
pusherAttributes 中也提供了直接关闭设备的属性：enableMic、enableCamera，当这两个属性的变化时，远端会收到状态变化事件，具体现象可见下表：

状态变化调用	enableMic 当前值	enableCamera 当前值	小程序现象	Web 端现象
enableMic: false	true	true	REMOTE_AUDIO_REMOVE 事件	收到 PLAYER_STATE_CHANGED 中 mute audio 事件
	true	false	REMOTE_AUDIO_REMOVE 事件	收到 stream-removed 事件
enableCamera: false	true	true	REMOTE_VIDEO_REMOVE 事件	收到 PLAYER_STATE_CHANGED 中 mute video 事件
	false	true	REMOTE_VIDEO_REMOVE 事件	收到 stream-removed 事件

实时音视频的小程序端、桌面浏览器端、PC 端是不是互通的?

是的，实时音视频支持全平台互通。

开通微信小程序SDK能力

自2023年02月15日起，如需使用微信小程序 SDK 音视频通话和互动直播，需开通订阅 TRTC 包月套餐以解锁小程序 SDK 能力，详见包月套餐 功能与计费说明 。

> **注意：**
> 2023年02月15日前创建过 TRTC 应用的腾讯云账号下的所有应用（SdkAppId），作为缓冲期可免费使用微信小程序 SDK 能力至2023年04月01日。

集成实时音视频 SDK 前需要做哪些准备工作?

1. 创建腾讯云实时音视频应用，购买相应的套餐，并获取到 SDKAppID 和密钥信息。
2. 小程序服务器域名配置。
3. 开通小程序类目与推拉流标签权限。
4. 出于政策和合规的考虑，微信暂未放开所有小程序对实时音视频功能（即 <live-pusher> 和 <live-player> 标签）的支持：
   - 小程序推拉流标签不支持个人小程序，只支持企业类小程序。
   - 小程序推拉流标签使用权限暂时只开放给有限 类目 。
   - 符合类目要求的小程序，需要在 **微信公众平台 > 开发 > 开发管理 > 接口设置**中自助开通该组件权限，如下图所示：

更多详情请参见 跑通 Demo (小程序) 和 快速集成(小程序) 。

运行出错，该如何排查?

1. 请检查开通的小程序类目是否正确，<live-pusher> 和 <live-player> 标签是否已开启。
2. 请确认已将 小程序域名白名单 添加到小程序 request 合法域名，或已开启调试模式。
3. 请重新解压小程序端 Demo 直接运行，若运行正常，建议参考 快速集成(小程序) 重新集成 SDK。
4. 若问题依然存在，可以登录 微信小程序开发者社区 查找相关资料，也可以 提交工单 或致电 4009100100 联系我们。

Android&iOS

最近更新时间：2024-12-13 21:47:43

移动端（Android/iOS）支持哪几种系统音量模式?

支持2种系统音量类型，即通话音量类型和媒体音量类型：

- 通话音量，手机专门为通话场景设计的音量类型，使用手机自带的回声抵消功能，音质相比媒体音量类型较差， 无法通过音量按键将音量调成零，但是支持蓝牙耳机上的麦克风。
- 媒体音量，手机专门为音乐场景设计的音量类型，音质相比于通话音量类型要好，通过音量按键可以将音量调成零。 使用媒体音量类型时，如果要开启回声抵消（AEC）功能，SDK 会开启内置的声学处理算法对声音进行二次处理。 在媒体音量模式下，蓝牙耳机无法使用自带的麦克风采集声音，只能使用手机上的麦克风进行声音采集。

移动端 SDK 推流怎么设置1080p分辨率?

1080P在 TX_Enum_Type_VideoResolution 定义是114，直接设置分辨率传枚举值即可。详情请参见 设置画面质量 。

> ⚠ **注意：**
> 目前 Android 和 iOS 暂不支持2K及以上的分辨率设置。

在小程序端创建了一个房间，移动端能否进入该房间?

可以，实时音视频支持全平台互通。

TRTC 移动端怎么实现录屏（屏幕分享）?

- **Android 端**：Version 7.2 及以上版本支持手机录屏，具体实践方法请参见 实时屏幕分享（Android）。
- **iOS 端**：Version 7.2 及以上版本支持 App 内录屏；Version 7.6 及以上版本支持手机录屏和 App 内录屏。具体实践方法请参见 实时屏幕分享（iOS）。

TRTC Android 端能不能支持64位的 arm64-v8a 架构?

TRTC 6.3 版本开始已提供 arm64-v8a 架构 ABI 支持。

在 Android 端怎么实现动态加载 so 库?

具体的操作步骤请参考 Android 端实现动态加载 so 库 。

在 Android 端为什么蓝牙耳机连接成功仍然无声或外放?

- **场景1：**
  - **问题：**Android 12 及以上版本设备连接蓝牙后，有蓝牙图标，可是蓝牙外放或无声，换成 Android 12 以下的版本正常。
  - **解决方案：**需要在 Manifest 中申请权限 `Manifest.permission.BLUETOOTH_CONNECT`，并且也需要动态申请。详情请参考 Android 官方文档。如果确认已经添加权限，可以查看 `TRTCCloudListener.onWarning` 的 `warningCode` 是否有收到 `TXLiteAVCode.WARNING_BLUETOOTH_DEVICE_CONNECT_FAIL`，如果有，就表示被其他应用影响了，可以退出其他应用试试。
- **场景2：**
  - **问题：**出现蓝牙外放或无声问题后，清空后台应用后恢复正常，可能是被三方 App 影响导致。
  - **解决方案：**SDK 版本更新至10.7+。
- **场景3：**
  - **问题：**使用有类似入耳检测功能的耳机概率性出现蓝牙外放或无声。
  - **解决方案：**SDK 版本更新至10.3+。
- **场景4：**

- **问题：**在重进房或修改音质时概率性出现蓝牙外放或无声，比如调用频繁调用 exitRoom、enterRoom、setAudioQuality。
- **解决方案：**SDK 版本更新至10.3+。

使用 startPlayMusic 接口播放背景音乐报错 -4001 是什么含义?

startPlayMusic 接口在遇到错误时会返回以下错误码：

- **-4001：**打开文件失败，如音频文件格式不支持、本地音频文件不存在、网络音频文件无法访问等。
- **-4002：**解码失败，如音频文件损坏、网络音频文件服务器无法访问等。

在遇到这类错误码时，可以尝试切换网络或者更换音频资源来解决问题。

在 iOS 端是否支持 Swift 集成?

支持，直接按照支持集成三方库的流程集成 SDK 即可，详情请参见 跑通Demo(iOS&Mac) 。

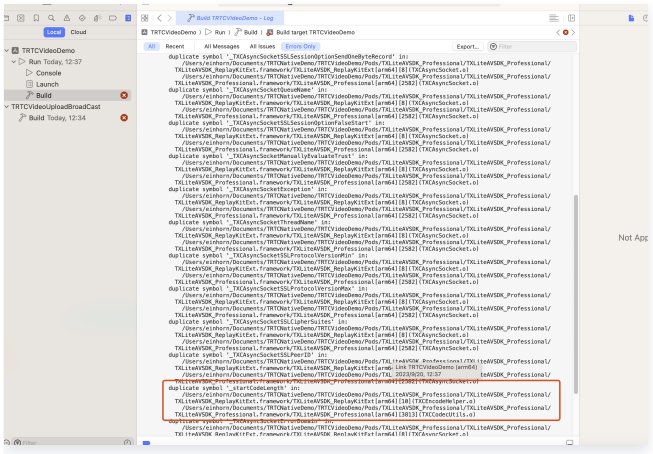
iOS 端 SDK 与其它三方库冲突报错问题该如何解决?

详情请参见 iOS 端 TXLiteAVSDK 与其它三方库冲突报错问题 。

在 Xcode 15 集成 iOS 端 SDK 遇到编译报错、提交 AppStore 报错问题该如何解决?

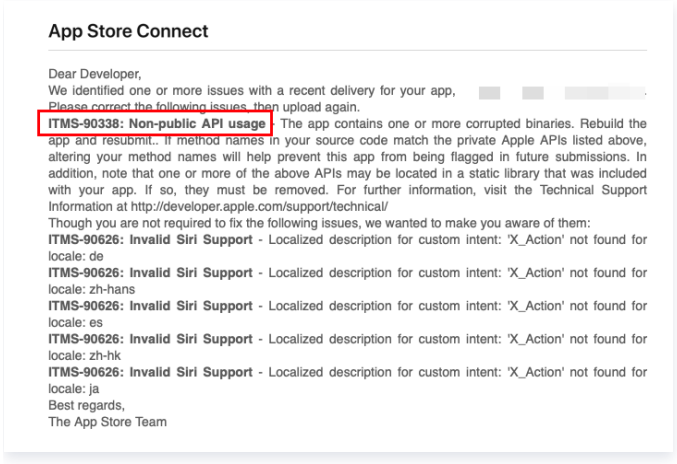
出现此类问题的场景一般有以下几种。

场景1：编译时报 `duplicate symbol 'xxx'` 符号冲突错误，例如打印类似下图的信息。



场景2：编译成功，在特定系统版本或者特定的机型出现异常崩溃，例如 iOS 11 及以下系统的真机运行崩溃。

场景3：上传到 TestFlight 苹果检测误报“Non-public API usage”，例如下图所示的检测误报。



App Store Connect

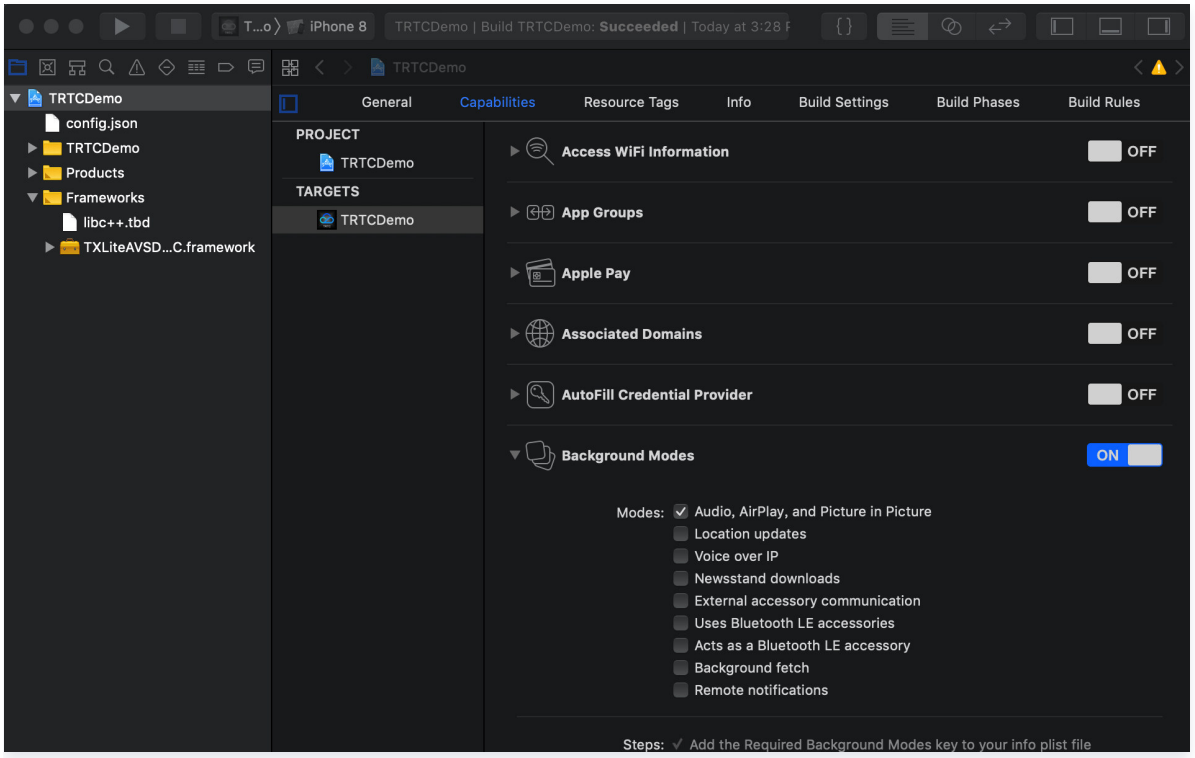
Dear Developer,
We identified one or more issues with a recent delivery for your app,
Please correct the following issues, then upload again.
ITMS-90338: Non-public API usage - The app contains one or more corrupted binaries. Rebuild the app and resubmit.. If method names in your source code match the private Apple APIs listed above, altering your method names will help prevent this app from being flagged in future submissions. In addition, note that one or more of the above APIs may be located in a static library that was included with your app. If so, they must be removed. For further information, visit the Technical Support Information at http://developer.apple.com/support/technical/
Though you are not required to fix the following issues, we wanted to make you aware of them:
ITMS-90626: Invalid Siri Support - Localized description for custom intent: 'X_Action' not found for locale: de
ITMS-90626: Invalid Siri Support - Localized description for custom intent: 'X_Action' not found for locale: zh-hans
ITMS-90626: Invalid Siri Support - Localized description for custom intent: 'X_Action' not found for locale: es
ITMS-90626: Invalid Siri Support - Localized description for custom intent: 'X_Action' not found for locale: zh-hk
ITMS-90626: Invalid Siri Support - Localized description for custom intent: 'X_Action' not found for locale: ja
Best regards,
The App Store Team

问题原因：

Xcode 15 项目中的 `-ObjC` 符号加载行为被调整为类似于 `-all_load` 的效果，导致重复加载相同符号，从而出现上述问题，解决的办法是将 **Other Linker Flags** 配置为 `-ld_classic`。

TRTC SDK 是否支持 iOS 后台运行?

支持，您只需选中当前工程项目，在 Capabilities 下的设置 Background Modes 为 ON，并勾选 Audio，AirPlay and Picture in Picture即可实现后台运行，详情如下图所示：



iOS 端是否可以监听远端离开房间的事件?

可以使用 onRemoteUserLeaveRoom 来监听用户离开房间事件，且该接口仅在 VideoCall 的所有用户和 LIVE 模式下的主播离开房间时会触发回调，观众离开房间不会有回调。

手机锁屏状态、App 在后台或 App 被关闭，音视频如何拨通?

实现离线接听等功能，详情请参见 实现离线接听 。

Android 端 App 退后台、锁屏等情况下麦克风采集无声，onWarning 回调 -1208，该如何处理?

场景1：被第三方应用打断（如系统电话、语音助手等）。
解决方案：关闭第三方应用后尝试 startLocalAudio 重启采集。

场景2：项目未启动 foregroundService 前台服务。
解决方案：请在项目中启动 foregroundService 前台服务，确保后台时 App 可以保持麦克风采集。

场景3：项目的 targetSdkVersion 设置为 30 及以上，或问题机型涉及到 Android 10 及更高系统版本。
解决方案：
若设置项目的 targetSdkVersion 30 及以上，除启动 foregroundService 外，还要求前台服务类型指定为 `android:foregroundServiceType:microphone`，以确保退后台可以保持麦克风正常采集。
若设置项目 targetSdkVersion 34 及以上，还需在 manifest 中配置 FOREGROUND_SERVICE_MICROPHONE 权限：

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

是否支持 Android 和 Web 端互通?

支持。使用相同的 SDKAppID，并进入同一个房间进行通话。详情请参见下列文档链接配置 Demo：

- 跑通 Demo（Android）

- 跑通 Demo（Web）

主播和粉丝在直播过程中连麦，是否双方都可以主动发起连麦?

双方都可以主动发起，观众和主播发起逻辑一致，具体操作请参见 跑通直播模式(Android) 。

多人音视频房间中，移动端和 Web 端是否可以进入同一房间?

可以。需保证 SDKAppID 和房间号一致，且用户 ID 不一致。

同一个页面中，是否可以创建 N 个 TRTC 对象，通过 N 个 UserID，分别登录到 N 个房间?

可以。Version 7.6 版本 开始支持一个用户进入多个房间了。

如何查询 SDK 最新版本号?

- 若您使用自动加载的方法，`latest.release` 为匹配最新版并进行自动加载，不需要对版本号进行修改。具体集成方法请参见 一分钟集成 SDK 。
- 当前 SDK 最新版本号可通过发布日志查看，具体请参见：
  - iOS & Android 端，请参见 发布日志（App）。
  - Web 端，请参见 发布日志（Web）。
  - Electron 端，请参见 发布日志（Electron）。

集成 Android 端 SDK 出现 <queries> 元素编译报错该如何解决?

解决方案：
Android 11 版本新增了软件包可见性过滤条件，需要在 AndroidManifest 添加 <queries> 元素来查看系统上的应用列表，但是 <queries> 元素声明与老版本 Gradle 插件不兼容，所以需要您升级项目中的 Gradle 插件版本，具体版本请查阅 Android Gradle 插件 。

uni-app

最近更新时间：2025-06-23 18:04:01

【官方】腾讯云实时音视频 SDK 支持悬浮窗吗?

【官方】腾讯云实时音视频 SDK 仅提供基本的音视频通话能力，悬浮窗的实现需要参考 uni-app 相关的官文文档自己实现。

【官方】腾讯云实时音视频 SDK 和【官方】腾讯云原生音视频插件 TencentCloud-TUICallKit 的区别?

- 【官方】腾讯云实时音视频 SDK 就是 `TRTC SDK`。【官方】腾讯云原生音视频插件 TencentCloud-TUICallKit 就是 TUICallKit。
- `TRTC SDK` 是音视频基础能力，无 UI 和逻辑。适用于比较复杂的场景，或者定制化比较强的场景。用现有的 TUICallKit 无法满足的需求，就可以使用 `TRTC SDK` 自己搭建。
- TUICallKit 插件包含 UI 和逻辑，专用于 1v1、多人音视频通话场景，接入简单。

【官方】腾讯云实时音视频 SDK 支持小程序吗?

建议请参见 TRTCSimpleDemoUniApp。

实例里面 app.vue 中的 aegis-weex-sdk 有什么用?

Demo 里的 `aegis-weex-sdk` 主要是用来对 Demo 做性能监控统计。

创建实例 this.trtcCloud = TrtcCloud.createInstance(); 和 this.trtcCloud = new TrtcCloud(); 有什么区别?

创建 trtcCloud 单例，只能通过 `TrtcCloud.createInstance()` 实例化一个 TrtcCloud 对象。

startRemoteView 在什么时候调用?

建议在 onUserVideoAvailable 事件回调中调用 startRemoteView。

本地预览如何自定义样式?

建议通过 view 包裹 `trtc-local-view`、`trtc-remote-view`。然后设置 view 的样式即可。

startLocalPreview 不传 viewId 可以吗?

startLocalPreview 可以不传 viewId。但是页面需要有 trtc-local-view 组件，并且它的 viewId 绑定的是本地用户 ID。

startLocalPreview 报 -2 错误?

- 错误原因：没有实例化本地预览的 view。
- 解决方案：
  - 首先必须是 .nvue 文件，详见。
  - 页面必须使用 `trtc-local-view` 并且绑定了 viewId。

【官方】腾讯云实时音视频 SDK 和 livepusher 推流模块同时使用，打包冲突?

结论：

目前【官方】腾讯云实时音视频 SDK（即：TRTC SDK）和 livepusher 无法同时使用。

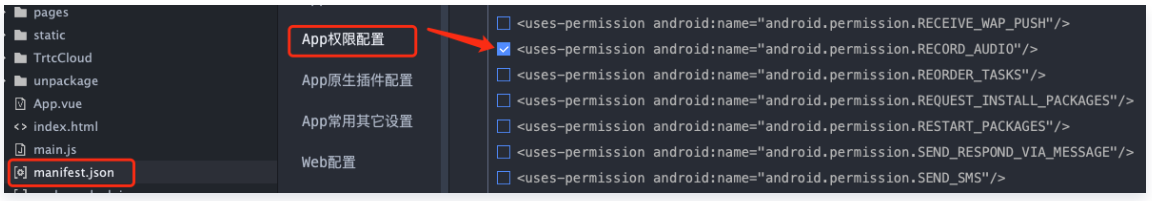
原因：

uni-app 官网文档介绍 第三方 SDK 使用 LiteAVSDK（weex_livepusher-release.aar）做直播推流，和【官方】腾讯云实时音视频 SDK 使用的 LiteAVSDK_TRTC 类冲突。和 uni-app 技术沟通后，对方反馈不清楚 SDK 属于哪个版本类型，因此 uni-app 他们无法对 livepusher 进行升级。

解决方案：

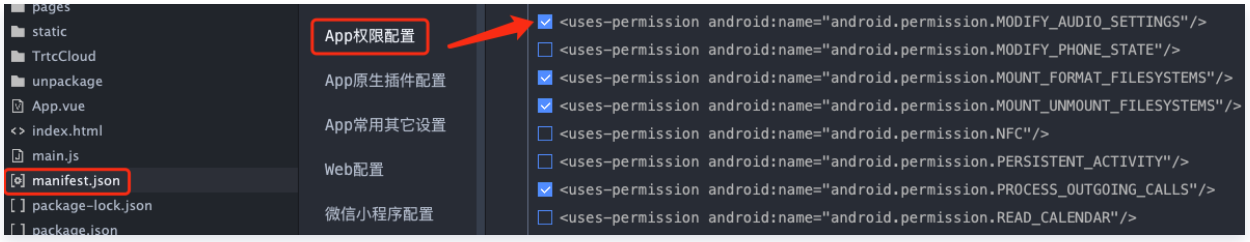
Api-Example 打开音频没作用?

检查打基座是否勾选 `android.permission.RECORD_AUDIO`，勾选后重新打基座。



Api-Example 音频路由 setAudioRoute 没作用?

检查打基座是否勾选 `android.permission.MODIFY_AUDIO_SETTINGS`，勾选后重新打基座。



【官方】腾讯云实时音视频 SDK 支持将插件 android 里的 .aar 放到 Android Studio 中离线打包吗?

不支持。目前仅支持 HBuilder 中采用 “本地插件” 或 “云端插件” 进行打包。

如何实现大小窗切换实现方案?

- 实现思路：用户在代码中写两个 <trtc-remote-view /> 标签，分别对应两个不同的 viewId(eg. 大窗时的 viewId='remoteBigID'，小窗时的 viewId='remoteSmallID')，在不同时刻渲染对应的 view。
- 快速体验：请下载 大小窗切换实现方案，参考 README.md 跑通。
- 实现逻辑

```
    <view class="trtc-box">
        <view v-if="isLocalSmall" class="big-view">
            <trtc-remote-view class="big-view" :userId="remoteUserId" viewId="remoteBigID">
</trtc-remote-view>
        </view>

        <view :class="[isLocalSmall ? 'small-view' : 'big-view']" >
            <trtc-local-view :class="[isLocalSmall ? 'small-view' : 'big-view']"
:viewId="userId"></trtc-local-view>
        </view>

        <view v-if="!isLocalSmall" class="small-view">
            <trtc-remote-view class="small-view" :userId="remoteUserId"
viewId="remoteSmallID"></trtc-remote-view>
        </view>
    </view>

    <button @click="toggleWindowSize"> toggleWindowSize </button>
    <button @click="() => startRemoteView('remoteBigID')">Start Remote View</button>
```

```
toggleWindowSize() {
    this.isLocalSmall = !this.isLocalSmall;
    if (this.isLocalSmall) {
        this.startRemoteView('remoteBigID');
    } else {
```

```
        this.startRemoteView('remoteSmallID');
    }
},
```

```
.trtc-box {
    width: 300px;
    height: 500px;
    border: 1px solid black;
    position: relative;
}

.big-view {
    width: 250px;
    height: 200px;
    position: absolute;
    background-color: red;
}

.small-view {
    width: 100px;
    height: 100px;
    position: absolute;
    right: 0;
    top: 0;
    background-color: green;
}
```

Web

最近更新时间：2024-11-08 10:36:22

一、基础环境问题

Web 端 5.x 版本和 4.x 版本有什么区别?

TRTC Web SDK 5.x 版本是 Web 端全新升级版，提供扁平化接口、大幅简化 API、降低您的接入成本；在多人音视频场景下，具有更好的性能表现及弱网抗性。
如何选择使用哪个版本?

- 若您是新接入的客户，建议您使用 5.x 版本，以获得更好的支持。
  相关资料：5.x 版本的 SDK 文档 、npm 地址 、发布日志 。
- 若您已经接入了 4.x 版本，建议尽快升级到 5.x 版本（参考：升级指引 ）。4.x 版本会持续维护，但不再迭代新功能。
  相关资料：4.x 版本的 SDK 文档 、npm 地址 、发布日志 。

Web 端 SDK 支持哪些浏览器?

TRTC Web SDK 对浏览器的详细支持度，请参见 TRTC Web SDK 对浏览器支持情况 。
对于上述没有列出的环境，您可以在当前浏览器打开 TRTC 能力测试 测试是否完整的支持 WebRTC 的功能。

通话前音视频设备测试?

您可以查看 通话前环境与设备检测 。

如何实时检测当前网络的情况?

具体请参见 通话前的网络质量检测 。

为什么本地开发测试能正常使用 TRTC Web SDK，但是部署到线上无法使用?

出于对用户安全、隐私等问题的考虑，浏览器限制网页只有在安全的环境下（例如 `https` 、 `localhost` 、 `file://` 等协议），才能采集麦克风、摄像头。HTTP 协议是不安全的，浏览器会禁止在 HTTP 协议下采集媒体设备。
若您在本地开发测试一切正常，但是页面部署后，却无法正常采集摄像头、麦克风。则请检查您的网页是否部署到了 HTTP 协议上，若是，请使用 HTTPS 部署您的网页，并确保具备合格的 HTTPS 安全证书。
更多详情请参见 URL域名及协议限制说明 。

是否支持大小流、水印?

详情请参见 大小流 、水印 文档实现高级功能。

WebRTC 有哪些已知问题?

具体请参见 WebRTC 已知问题及规避方案 。

二、推拉流问题

Web 端 SDK 日志中报错 NotFoundError、NotAllowedError、NotReadableError、OverConstrainedError 以及 AbortError 分别是什么意思?

这是设备采集失败时的错误，详情请参考：DEVICE_ERROR 。

部分手机上的浏览器无法正常运行 TRTC 进行推拉流?

TRTC Web SDK 对浏览器的详细支持度，请参见 TRTC Web SDK 对浏览器支持情况 。
对于上述没有列出的环境，您可以在当前浏览器打开 TRTC 能力测试 测试是否完整的支持 WebRTC 的功能。

Web 端用宽高设置推流的分辨率是所有浏览器都适用吗?

由于设备和浏览器的限制，视频分辨率不一定能够完全匹配，在这种情况下，浏览器会自动调整分辨率使其接近 Profile 对应的分辨率。详情请参见 trtc.startLocalVideo 的 profile 参数 。

Web 端支持哪些视频画面分辨率设置?

目前 Web 端支持设置 720p、1080p、2k 和 4k 的视频画面分辨率，更多设置详情请见Web端画面质量设置 。

为什么摄像头支持采集 1080p，而采集出来的分辨率不到 1080p?

这种情况一般是由于摄像头提前被低分辨率采集占用，导致 SDK 无法采集到目标分辨率。
解决方案：检查业务侧逻辑，如果提前采集了摄像头，请在无需使用时，及时释放摄像头采集。

Web 端屏幕分享的样式支持修改吗?

屏幕分享的样式由浏览器控制，目前不能修改。

Web 端 SDK 在使用的过程中拔掉摄像头，怎么清除摄像头列表里面的数据?

监听 DEVICE_CHANGED 事件，SDK 会在设备插拔时触发该事件，此时重新获取一次摄像头列表即可。

iOS 的微信内嵌浏览器不能正常推流?

iOS 14.3+ 版本的微信内嵌浏览器，可以正常推流。
请参见 浏览器支持情况，查看 iOS 上的微信内嵌浏览器对推拉流的支持情况。

三、播放问题

通话过程中出现有画面但是听不到声音?

- 因浏览器自动播放策略限制，在用户没有与页面产生交互的情况下，音频播放可能会被浏览器拦截，请参见 自动播放受限处理建议 。
- 其他异常，可通过监控仪表盘查询收发两端的音量值进行排查。

Android 端采集麦克风后，扬声器声音变成从听筒播放了?

Android 端一般会有多个麦克风，label 列表为： ['default', 'Speakerphone', 'Headset earpiece']，如果在 trtc.startLocalAudio 时，不指定麦克风，浏览器默认麦克风可能会采集到 Headset earpiece，此时声音会从听筒放出。若需要通过扬声器外放，则需要指定采集 label 为 Speakerphone 的麦克风。

四、其他

2.x、3.x版本的 SDK，在 Chrome 96+ 版本无法正常通话该如何处理?

最新版本的 Chrome 96 废弃了 Plan-B，将会导致 TRTC 实时音视频老版本的(2.x, 3.x) Web SDK 会出现无法通话的情况，请您尽快将 Web SDK 升级至我们的最新版本(5.x)。5.x 版本 SDK 的接口与老版本(2.x, 3.x)不兼容，请参考 快速集成(Web) 升级接入 5.x 版本 SDK。

通话一直没办法互通，切换网络后就正常了?

一般是由于网络防火墙导致，请检查防火墙配置。具体请参见 应对防火墙限制相关 进行处理。

Web 端 SDK 可以获取当前音量大小吗?

可以通过 AUDIO_VOLUME 事件 获取当前音量大小，具体请参见 音量大小检测 。

什么情况会收到 KICKED_OUT 事件?

当用户被移出时会触发该事件，例如：使用同名用户同时登录、调用后台 RESTAPI 移除用户 将用户移出房间、调用后台 RESTAPI 解散房间 。

> ⚠ **注意：**
> 同名用户同时登录是不允许的行为，可能会导致双方通话异常，业务层应避免出现同名用户同时登录。

Web 端是否可以监听远端离开房间?

支持监听远端退房事件，建议使用客户端事件中的 REMOTE_USER_EXIT 事件实现远端用户退房通知。

TRTC 的 Web 端、小程序端、PC 端是不是互通的?

是的，实时音视频支持全平台互通。

收到 trtc.on('error') 事件该如何处理?

这个表示 SDK 遇到不可恢复错误，在 SDK 经过多次重试后仍然无法恢复时，会抛出该事件。此时可以通过重新进房或者刷新页面来恢复。

Web 端如何在屏幕分享的时候采集系统声音?

具体操作请参见 实现屏幕分享 。

Web 端如何切换摄像头和麦克风?

参考：切换摄像头和麦克风 。

在 iframe 使用 TRTC Web SDK 报错 Permission denied?

在 iframe 中使用 WebRTC 需要给 iframe 标签增加属性来开启相关权限，具体参考如下。
麦克风、摄像头、屏幕分享权限：

```
<iframe allow="microphone; camera; display-capture;">
```

在微信 H5 无法采集摄像头和麦克风?

出现这种情况，可能是您的微信清理了数据、或者是首次安装的微信，这种情况下微信使用安装时自带的浏览器内核是不支持设备采集的。
微信会在 Wi-Fi 联网的情况下，自动下载安装支持设备采集的 XWeb 内核。您可以在 Wi-Fi 联网一段时间（例如半小时）后再次重试。

使用 Vue 3 有哪些注意事项?

在 Vue3 框架使用时，需要注意使用 markRaw 接口标记 trtc 实例，避免 Vue 将 trtc 实例转成 Proxy 对象，导致不可预知的问题。

```
import { markRaw } from 'vue';
const trtc = markRaw(TRTC.create());
```

Flutter

最近更新时间：2025-03-11 17:16:22

两台手机同时运行 Demo，为什么看不到彼此的画面?

请确保两台手机在运行 Demo 时使用的是不同的 UserID，TRTC 不支持同一个 UserID（除非 SDKAppID 不同）在两个终端同时使用。

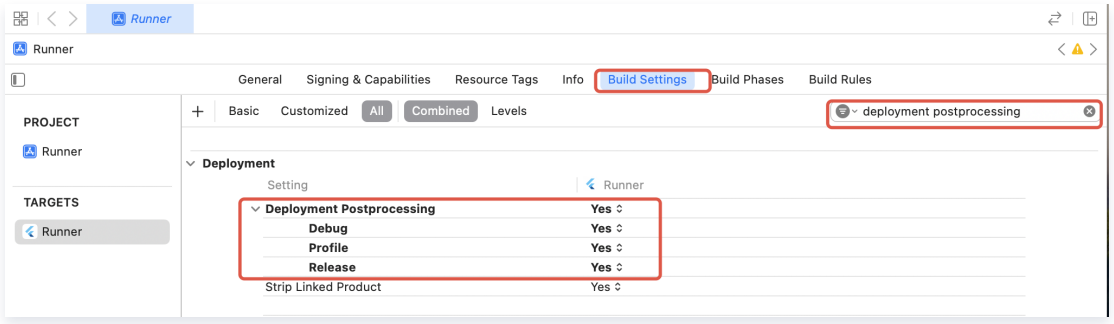
防火墙有什么限制?

由于 SDK 使用 UDP 协议进行音视频传输，所以在对 UDP 有拦截的办公网络下无法使用。如遇到类似问题，请参见 应对公司防火墙限制 排查并解决。

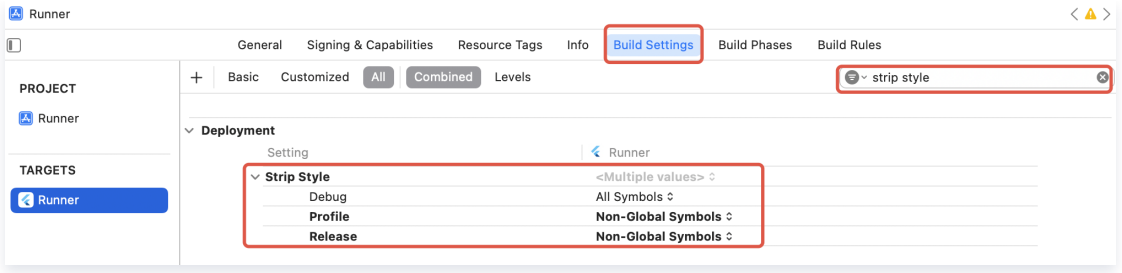
iOS release 包运行时 [symbol not found]?

由于 tencent_rtc_sdk 通过 Flutter FFI 调用接口，iOS Release 构建时 Xcode 的符号裁剪优化可能误移除 TRTC 的 C 符号，引发 `symbol not found` 错误。解决方案如下：

1. 在项目的 Build Settings中找到 `deployment postprocessing`，将其设置为 **Yes** 。



2. 在项目的 Build Settings中找到 `strip style`，将其中 Release 的值设置为 **Non-Global Symbols** 。



iOS 打包运行 Crash?

请排查是否 iOS14 以上的 debug 模式问题，具体请参见 官方说明 。

iOS 无法显示视频（Android 正常）?

请确认在您项目的 `info.plist` 中 `io.flutter.embedded_views_preview` 值为 YES。

更新 SDK 版本后，iOS CocoaPods 运行报错?

1. 删除 iOS 目录下 `Podfile.lock` 文件。
2. 执行 `pod repo update` 。
3. 执行 `pod install` 。
4. 重新运行。

Android Manifest merge failed 编译失败?

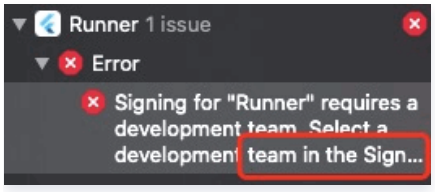
1. 请打开 `/example/android/app/src/main/AndroidManifest.xml` 文件。
2. 将 `xmlns:tools="http://schemas.android.com/tools"` 加入到 manifest 中。
3. 将 `tools:replace="android:label"` 加入到 application 中。

android > app > src > main > AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.example.mlp">
    <!-- io.flutter.app.FlutterApplication is an android.app.Application that
         calls FlutterMain.startInitialization(this); in its onCreate method.
         In most cases you can leave this as-is, but you if you want to provide
         additional functionality it is fine to subclass or reimplement
         FlutterApplication and put your custom class here. -->
    <application
        tools:replace="android:label"
        android:name="io.flutter.app.FlutterApplication"
        android:label="mlp"
        android:icon="@mipmap/ic_launcher">
```

因为没有签名，真机调试报错?

若报错信息如下图所示：



1. 您需购买苹果证书，并进行配置、签名操作后，即可在真机上调试。

2. 证书购买成功后，在 `target > signing & capabilities` 中进行配置。

对插件内的 swift 文件做了增删后，build 时查找不到对应文件?

在主工程目录的 `/ios` 文件路径下 `pod install` 即可。

Run 报错“Info.plit, error: No value at that key path or invalid key path: NSBonjourServices”?

执行 `flutter clean` 后，重新运行即可。

Pod install 报错?

若报错信息如下图所示：

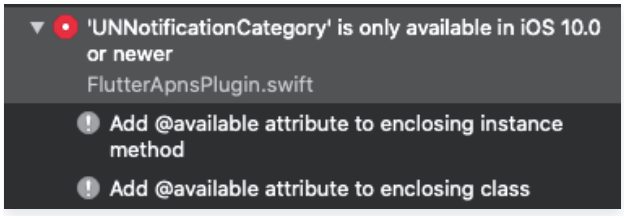
```
linzhi@MacBook-Pro ios % pod install

[!] Invalid `Podfile` file: /Users/linzhi/Desktop/source/trtc-flutter-plugin-demo
/trtc-flutter-plugin/example/ios/Flutter/Generated.xcconfig must exist. If you're
 running pod install manually, make sure flutter pub get is executed first.
```

报错信息里面提示 pod install 的时候没有 `generated.xconfig` 文件，因此运行报错，您根据提示**需要执行 flutter pub get** 解决。

> **说明：**
> 该问题是 flutter 编译后的问题，新项目或者执行了 `flutter clean` 后，都不存在这个问题。

Run 的时候 iOS 版本依赖报错?

若报错信息如下图所示：



可能是 pods 的 target 版本无法满足所依赖的插件，因此造成报错。因此您需修改报错 pods 中的 target 到对应的版本。

Flutter 支持自定义采集和渲染吗?

目前不支持。自定义采集和渲染平台支持详情，请参见 支持的平台 。

Electron

最近更新时间：2025-03-07 12:06:33

安装相关

trtc-electron-sdk 是否兼容官方 Electron v12.0.1 版本?

兼容的，trtc-electron-sdk 没有特别依赖 elecron 自身的 SDK，所以没有相关的版本依赖。

Electron 下载慢甚至卡住不动?

当开始下载 `tmp-3320-1-SHASUMS256.txt-6.1.9` 文件或其它文件时，可能会特别慢，甚至在辛苦等待了很长时间后，等到的却是 npm 的 Timeout 错误：

```
Downloading tmp-3320-1-SHASUMS256.txt-6.1.9
[=>                                   ] 1.0% of 5.56 kB (0 B/s)
```

- **解决方案 A：**如果您是在家中办公，可以切换到国内的 npm 镜像。

  ```
  # 指定 npm 国内镜像
  npm config set registry http://mirrors.tencent.com/npm/

  # 指定 Electron 的国内镜像地址
  npm config set ELECTRON_MIRROR=https://registry.npmmirror.com/mirrors/electron/
  npm install
  ```

- **解决方案 B：**如果您是在公司办公，那么您公司的网络管理员可能已经设置了代理，需要确认 npm 的 proxy 配置是否指向了公司的代理服务器，以及是否配置了环境变量 `ELECTRON_GET_USE_PROXY` ，如均无配置，请按以下步骤执行。

  1.1 设置 npm 代理： `npm config set all_proxy=[您的代理地址]` 。

  1.2 配置 `ELECTRON_GET_USE_PROXY` 环境变量，这样 Electron 的安装脚本就会通过 npm 的代理下载。

- **解决方案 C：**如果您是 Mac 环境。

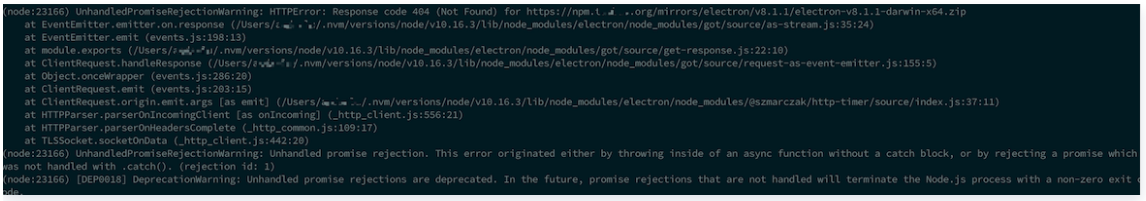
  ```
  export ELECTRON_GET_USE_PROXY=true
  ```

- **解决方案 D：**如果您是 Windows 环境。

  1.1 右键单击**计算机 > 属性 > 高级系统设置 > 环境变量**。

  1.2 按下图操作设置环境变量 ELECTRON_GET_USE_PROXY ，然后执行 `npm install` 或 `npm install --proxy=[您的代理地址]` ：

下载 Electron 时出现 404 错误?



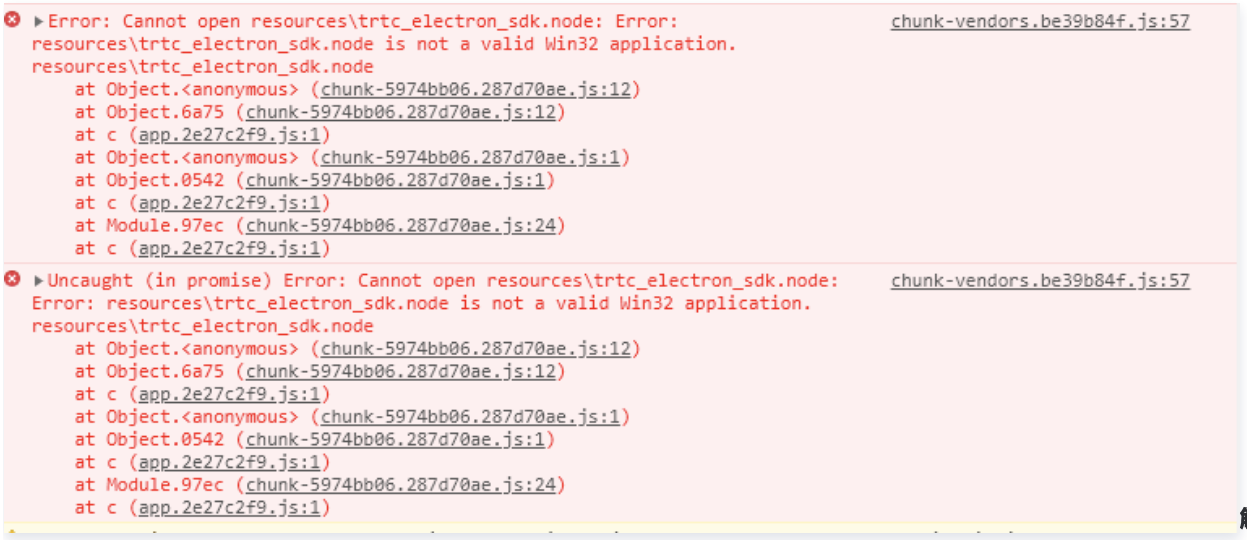
在终端中输入如下指令：

```
npm config set electron_custom_dir 8.1.1 # 根据版本号来决定
```

运行相关

Windows 32 系统运行报错

`Error: resource\trtc_electron_sdk.node is not a valid Win32 application`，提示需要 32 位的 trtc_electron_sdk.node?



解决方法：

1. 进入到工程目录下的 trtc-electron-sdk 库目录下（xxx/node_modules/trtc-electron-sdk）。执行：

```
npm run install -- arch=ia32
```

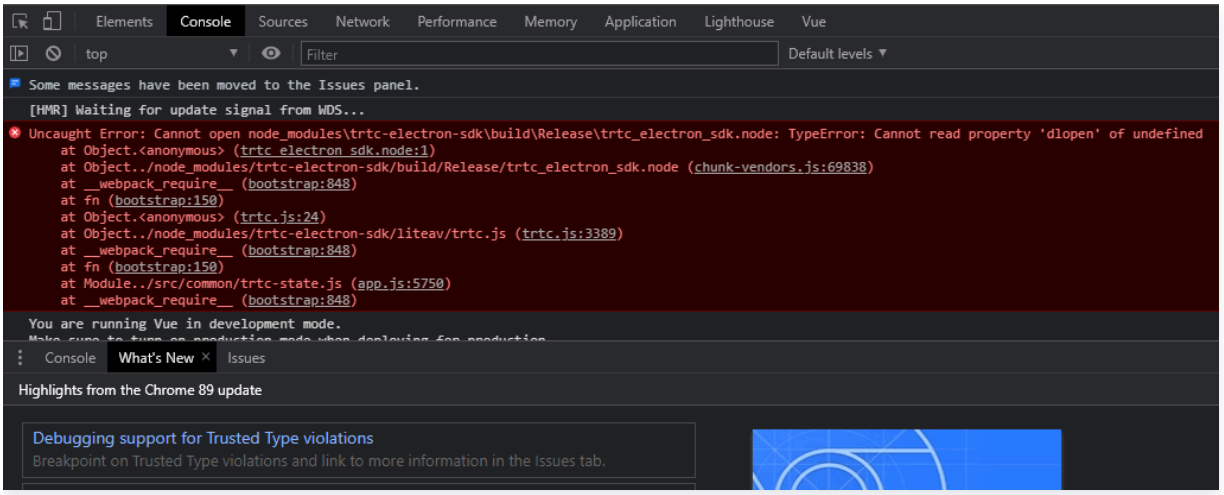
2. 下载完 32 位的 `trtc_electron_sdk.node` 后，重新对项目进行打包。

vscode terminal 启动 Electron Demo，进入房间后白屏?

vscode 需有摄像头权限, 可采用如下方式进行权限添加。

```
cd ~/Library/Application\ Support/com.apple.TCC/
cp TCC.db TCC.db.bak
sqlite3 TCC.db    # sqlite> prompt appears.

# for Mojave, Catalina
INSERT into access
VALUES('kTCCServiceCamera',"com.microsoft.VSCode",0,1,1,NULL,NULL,NULL,'UNUSED',NULL,0,15414
40109);

# for BigSur
INSERT into access
VALUES('kTCCServiceCamera',"com.microsoft.VSCode",0,1,1,1,NULL,NULL,NULL,'UNUSED',NULL,0,154
1440109);
```

跑 Demo 抛出空指针未定义的错误：`“cannot read property 'dlopen' of undefined”`?



解决方法：

Electron 12 版本上下文隔离默认启用，可设置 contextIsolation 为 false。

```
let win = new BrowserWindow({
    width: 1366,
    height: 1024,
    minWidth: 800,
    minHeight: 600,
    webPreferences: {
    nodeIntegration: true,
    contextIsolation: false
        },
});
```

Electron 多次出现重新进房问题?

需要具体 case 进行分析，大致原因如下：

- 客户端网络状态不好（断网会触发重进房）。
- 连着发两次进房信令也会重进房的。
- 有可能是设备负载过高，导致解码失败的重进房。
- 同一个 UID 多端登录互踢导致的重进房。

终端出现提示“Electron failed to install correctly”？

当看似安装完成，运行项目时，终端上出现以下错误：

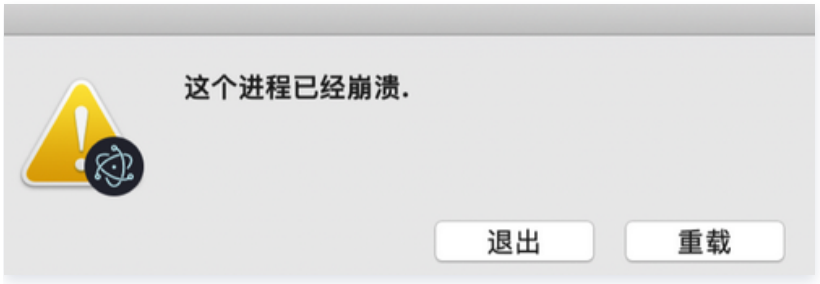
```
Error: Electron failed to install correctly, please delete node_modules/electron and try
installing again
```

按照如下三个步骤进行**手动下载**：

1. 执行 `npm config get cache` 查看缓存目录。
2. 手动下载 Electron ，并放到缓存目录中。
3. 重新执行 `npm install` 。

调用摄像头或麦克风时直接崩溃?

使用 vscode 终端启动项目，当 trtc-electron-sdk 启动摄像头和麦克风时，程序直接崩溃：



- **解决方案 A**：使用有授权的终端运行项目。
- **解决方案 B**：给 vscode 授权：在**系统偏好设置 > 安全与隐私** 中允许 vscode 的授权。
- **解决方案 C**：按以下步骤关闭保护机制：

  1.1 重启系统，**按住** command + r 键，**直到**系统进入保护模式。

  1.2 打开 terminal 输入 `csrutil disable` 禁用保护机制。

  1.3 重启，正常进入系统，此时就可以使用 vscode 的终端启动项目了。

  1.4 如需重新启动保护机制，只需要在第二步中执行 `csrutil enable` 。

Electron 在控制台中报错“xx is not defined”？

当运行项目时，Electron 在控制台中提示 `xx is not defined` ，其中 `xx` 指代 node 模块。例如：

```
Uncaught ReferenceError: require is not defined
```

在 Electron 的 `main.js` 文件中将 `nodeIntegration` 配置项改成 true：

```
let win = new BrowserWindow({
    width: 1366,
    height: 1024,
    webPreferences: {
        nodeIntegration: true,  // 请将此项设置为 true
    },
});
```

Mac 下，打包安装后，运行时白屏、崩溃问题

在 Mac OS 10.4 及以后版本，运行安装包时，如果获取不到 摄像头、麦克风、屏幕录制 权限，程序会因为没有这些硬件的访问权限，在进入 TRTC 房间后直接白屏或者崩溃。解决方法如下：

1. 添加 entitlements.mac.plist 文件，文件内容如下，相关配置的具体含义，请参见 苹果开发者网站 。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
  <dict>
 <key>com.apple.security.cs.allow-unsigned-executable-memory</key>
 <true></true>
 <key>com.apple.security.cs.allow-jit</key>
 <true></true>
 <key>com.apple.security.cs.allow-dyld-environment-variables</key>
 <true></true>
```

```
  <key>com.apple.security.device.audio-input</key>
  <true></true>
  <key>com.apple.security.device.camera</key>
  <true></true>
   </dict>
</plist>
```

2. 使用 electron-builder 打包时，需要将 entitlements.mac.plist 文件路径配置到 electron-builder 打包配置中。参考代码如下，注意 `"entitlements"` 和 `"entitlementsInherit"` 两个配置项， `"hardenedRuntime"` 需要配置为 true，配置项含义请参见 electron-builder官网 。

```
{
  "build": {
    "mac": {
      "extraFiles": [
        {
          "from": "node_modules/trtc-electron-sdk/build/Release/trtc_electron_sdk.node",
          "to": "./Resources"
        }
      ],
      "type": "distribution",
      "hardenedRuntime": true,
      "entitlements": "assets/entitlements.mac.plist",
      "entitlementsInherit": "assets/entitlements.mac.plist",
      "gatekeeperAssess": false,
      "target": [
        "dmg"
      ]
    },
  }
}
```

3. Mac OS 12.1 及以上版本，仅使用以上配置，已不足以申请 麦克风、摄像头权限，需要借助 Electron API 的 systemPreferences.askForMediaAccess() 接口，在主进程中，主动申请一次摄像头、麦克风权限。参考代码如下。systemPreferences.getMediaAccessStatus() 接口可以检测 麦克风、摄像头、屏幕录制权限，但 systemPreferences.askForMediaAccess() 接口只能申请 摄像头 和 麦克风权限。

```
async checkAndApplyDeviceAccessPrivilege() {
  const cameraPrivilege = systemPreferences.getMediaAccessStatus('camera');
  console.log(
    `checkAndApplyDeviceAccessPrivilege before apply cameraPrivilege: ${cameraPrivilege}`
  );
  if (cameraPrivilege !== 'granted') {
    await systemPreferences.askForMediaAccess('camera');
  }

     const micPrivilege = systemPreferences.getMediaAccessStatus('microphone');
  console.log(
    `checkAndApplyDeviceAccessPrivilege before apply micPrivilege: ${micPrivilege}`
  );
  if (micPrivilege !== 'granted') {
    await systemPreferences.askForMediaAccess('microphone');
```

```
  }

    const screenPrivilege = systemPreferences.getMediaAccessStatus('screen');
  console.log(
    `checkAndApplyDeviceAccessPrivilege before apply screenPrivilege: ${screenPrivilege}`
  );
}
```

4. 更多崩溃问题处理方法，请参阅 Electron 应用崩溃问题排查与解决方法 。

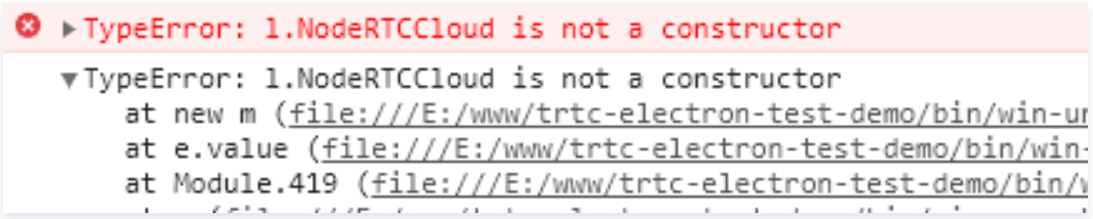
打包相关

.node 模块的加载问题?

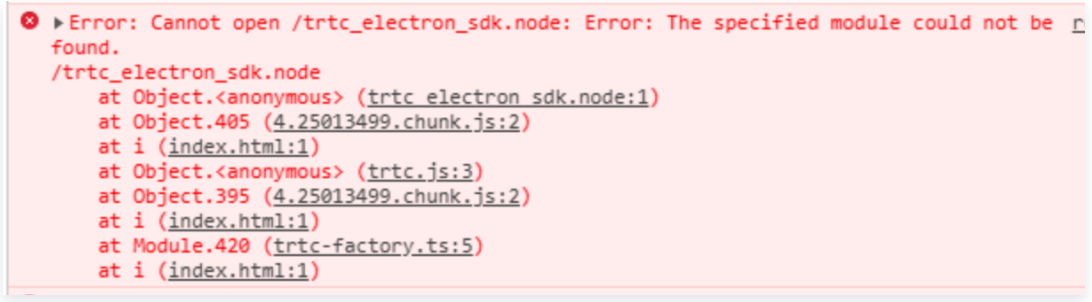
报错信息

打包编译出的程序在运行时，在控制台中看到类似的报错信息：

- `NodeRTCCloud is not a constructor`



- `Cannot open xxx/trtc_electron_sdk.node` 或者 `The specified module could not be found`



- `dlopen(xxx/trtc_electron_sdk.node, 1): image not found`

解决方法

出现类似上述的信息，说明 trtc_electron_sdk.node 模块没有被正确的打包到程序中，可按照以下步骤进行处理。

1. 安装 `native-ext-loader` 。

```
npm i native-ext-loader -D
```

2. 修改 webpack 配置。

2.1 使 `webpack.config.js` 在构建时可以接收名为 `--target_platform` 的命令行参数，以使代码构建过程按不同的目标平台特点正确打包，在 `module.exports` 之前添加以下代码：

```
const os = require('os');
// 如果不传 target_platform 参数，程序会默认按当前平台类型进行打包
const targetPlatform = (function(){
 let target = os.platform();
 for (let i=0; i<process.argv.length; i++) {
    if (process.argv[i].includes('--target_platform=')) {
            target = process.argv[i].replace('--target_platform=', '');
            break;
        }
    }
  // win32 统一表示 Windows 平台，包含 32 位和 64 位。darwin 表示 Mac 平台
  if (!['win32', 'darwin'].includes) target = os.platform();
  return target;
})();
```

2.2 添加以下 rules 配置：

```
module: {
 rules: [
 {
     test: /\.node$/,
     loader: 'native-ext-loader',
     options: {
         rewritePath: targetPlatform === 'win32' ? './resources' : '../Resources'
     }
 },
 ]
}
```

> **注意：**
> - 使用 `vue-cli` 创建的项目，webpack 配置存放在 `vue.config.js` 文件中的 `configureWebpack` 选项中。
> - 使用 `create-react-app` 创建的项目，webpack 配置文件为 `[项目目录]/node_modules/react-scripts/config/webpack.config.js` 。

3. 配置 packages.json 文件，添加打包配置和构建脚本。

3.1 添加 `electron-builder` 打包配置（注意大小写）：

```
"build": {
 "省略": "...",
 "win": {
 "extraFiles": [
     {
     "from": "node_modules/trtc-electron-sdk/build/Release/",
     "to": "./resources",
     "filter": ["**/*"]
     }
 ]
 },
 "mac": {
 "extraFiles": [
```

```
      {
      "from": "node_modules/trtc-electron-sdk/build/Release/trtc_electron_sdk.node",
      "to": "./Resources"
      }
  ]
  },
  "directories": {
  "output": "./bin"
  }
},
```

3.2 添加 scripts 构建、打包脚本 `create-react-app` 项目请参考以下配置：

```
"scripts": {
  "build:mac": "react-scripts build --target_platform=darwin",
  "build:win": "react-scripts build --target_platform=win32",
  "compile:mac": "node_modules/.bin/electron-builder --mac",
  "compile:win64": "node_modules/.bin/electron-builder --win --x64",
  "pack:mac": "npm run build:mac && npm run compile:mac",
  "pack:win64": "npm run build:win && npm run compile:win64"
}
```

3.3 `vue-cli` 项目请参考以下配置：

```
"scripts": {
  "build:mac": "vue-cli-service build --target_platform=darwin",
  "build:win": "vue-cli-service build --target_platform=win32",
  "compile:mac": "node_modules/.bin/electron-builder --mac",
  "compile:win64": "node_modules/.bin/electron-builder --win --x64",
  "pack:mac": "npm run build:mac && npm run compile:mac",
  "pack:win64": "npm run build:win && npm run compile:win64"
}
```

找不到入口文件?

使用 `create-react-app` 创建的项目，使用 `electron-builder` 打包时可能会遇到此问题：

```
$ node_modules\.bin\electron-builder.cmd
  • electron-builder  version=22.6.0 os=6.1.7601
  • loaded configuration  file=package.json ("build" field)
  • public/electron.js not found. Please see https://medium.com/@kitze/%EF%B8%8F-from-react-
to-an-electron-app-ready-for-production-a0468ecb1da3
  • loaded parent configuration  preset=react-cra
```

其中 `public/electron.js not found` 指的就是无法找到入口文件。

解决方案

1. 移动并重命名入口文件：

```
cd [项目目录]
```

```
mv main.electron.js ./public/electron.js
```

2. 修改 pacakge.json 文件：

```
{
 "main": "public/electron.js",
 "省略": "..."
}
```

在执行打包时，出现 fs-extra 模块的语法错误?

```
[项目目录]\node_modules\electron-builder\node_modules\fs-extra\lib\empty\index.js:33
    } catch {
            ^

SyntaxError: Unexpected token {
    at new Script (vm.js:51:7)
```

可以升级到最新的 node ，具体请参见 Node.js 官方网站 。

Mac 下如何构建 X64 和 ARM64 双架构包?

TRTC Electron SDK 从 10.6.403 版本开始，默认支持 Mac 下构建 X64 和 ARM64 双架构包，只需要修改一下 electron-builder 和构建工具 Webpack 或者 Vite 配置即可。详细配置请参见 TRTC Electron SDK: Mac 下构建双架构包 。
支持双架构打包后，如果不修改配置，则默认构建单架构应用包，架构类型与构建机器 CPU 类型相同。

> **说明：**
> 更多 Electron 相关问题请关注 Electron 常见问题收录 和 Electron 常见问题收录II ，我们将持续更新。

接入房间引擎（RoomEngine-SDK）
概述

最近更新时间：2025-05-22 16:39:43

RTC Room Engine SDK 是 多人会议（RoomKit）和 直播与语聊房（TUILiveKit）共同使用的软件开发工具包。它支持房间管理、屏幕分享、成员管理、麦位控制、基础美颜等丰富功能，同时确保720P和1080P高清画质，在70%丢包率的弱网环境下仍能保持高质量音视频传输。此外，SDK 采用48kHz高音质采样，结合腾讯天籁实验室的3A处理算法，消除回声和啸叫，实现全链路128kbps高音质立体声，为用户带来清晰沉浸的互动体验。

多人会议（RoomKit）

多人会议 (RoomKit) 是一款定位企业会议、网络研讨会、在线教育等多人音视频会话场景的含 UI 低代码组件，支持小程序、 iOS、Android、Web、uni-app、Flutter 等平台全球互通。提供房间管理、成员管理、屏幕分享等会控功能，支持标清、高清、超高清等多种画质。通过集成该组件，仅需3步1天内就可以为您的 App 添加多人音视频会话功能，快速上线业务。

计费详见：多人会议（含UI）开通服务

直播与语聊室（TUILiveKit）

直播与语聊室 (TUILiveKit) 是一款适用于社交娱乐、电商、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：

计费详细参见：直播与语聊房（含UI）开通服务

快速接入
Android

最近更新时间：2025-05-22 16:39:43

本文主要介绍如何快速将 RTC Room Engine SDK 集成到您的 Android 项目中。

开发环境要求

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

集成 SDK

在项目需要使用 RTC Room Engine SDK 的 module 工程的 build.gradle 文件（或 build.gradle.kts 文件）中，添加 RoomEngine 模块。

build.gradle.kts

```
api("io.trtc.uikit:rtc_room_engine:latest.release")
// rtc room engine 依赖 trtc sdk 和 im sdk
api("com.tencent.liteav:LiteAVSDK_Professional:latest.release")
api("com.tencent.imsdk:imsdk-plus:latest.release")
```

build.gradle

```
api "io.trtc.uikit:rtc_room_engine:latest.release"
// rtc room engine 依赖 trtc sdk 和 im sdk
api "com.tencent.liteav:LiteAVSDK_Professional:latest.release"
api "com.tencent.imsdk:imsdk-plus:latest.release"
```

版权所有：腾讯云计算（北京）有限责任公司

iOS

最近更新时间：2025-05-22 16:39:43

本文主要介绍如何快速将 RTC Room Engine SDK 集成到您的 iOS 项目中。

开发环境要求

Xcode 15 及以上。
iOS 13.0 及以上。
CocoaPods 环境安装，详细操作方法参见 CocoaPods 官方安装指引 。
项目已配置有效的开发者签名。

集成 SDK

方案一：使用 CocoaPods

安装 CocoaPods

在终端窗口中输入如下命令（需要提前在 Mac 中安装 Ruby 环境）：

```
sudo gem install cocoapods
```

创建 Podfile 文件

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

```
pod init
```

编辑 Podfile 文件

请您按照如下方式设置 Podfile 文件：

```
platform :ios, '13.0'

target 'App' do
  use_frameworks!
  # 添加 RoomEngine SDK
  pod 'RTCRoomEngine'
end
```

更新并安装 SDK

在终端窗口中输入如下命令以更新本地库文件，并安装 Chat SDK：

```
pod install
```

或使用以下命令更新本地库版本：

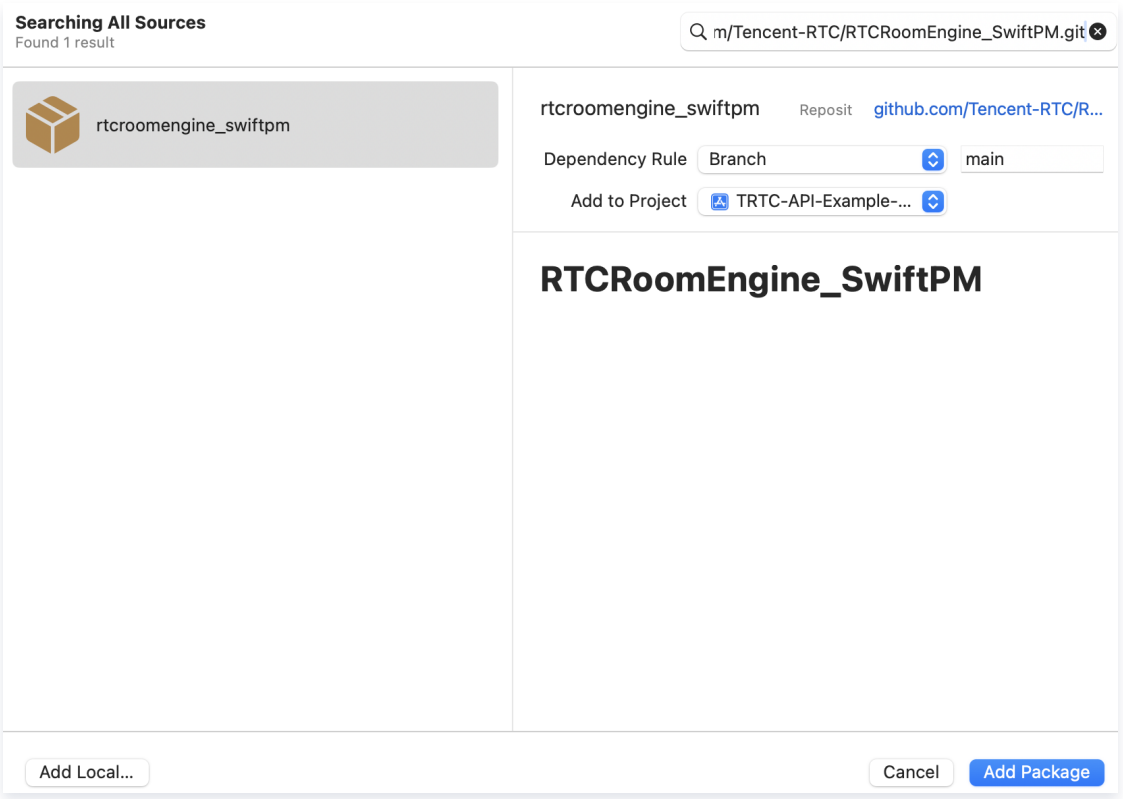
```
pod update
```

pod 命令执行完后，会生成集成了 SDK 的 .xcworkspace 后缀的工程文件，双击打开即可。若 pod 搜索失败，建议尝试更新 pod 的本地 repo 缓存。更新命令如下：

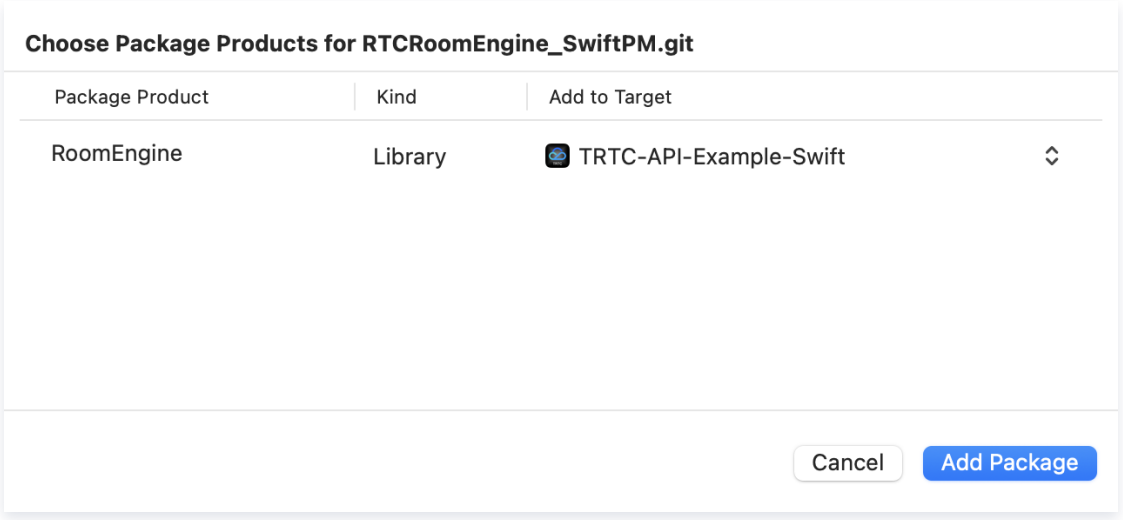
```
pod setup
pod repo update
rm ~/Library/Caches/CocoaPods/search_index.json
```

方案二：使用 Spm

1. 打开您的 Xcode 项目，在 Xcode 菜单依次单击 **File** > **Add Package Dependencies…**。
2. 在弹出的对话框右上角输入网址：`https://github.com/Tencent-RTC/RTCRoomEngine_SwiftPM.git` 。
3. 选择 **Dependency Rule** 为 **Branch**，分支名为 **main**，或者选择您需要的版本号，**Add to Project** 选择当前项目。



4. 单击 **Add Package** 进入设置 target 界面。



5. 单击 Add Package 完成集成。

引用 RoomEngine SDK

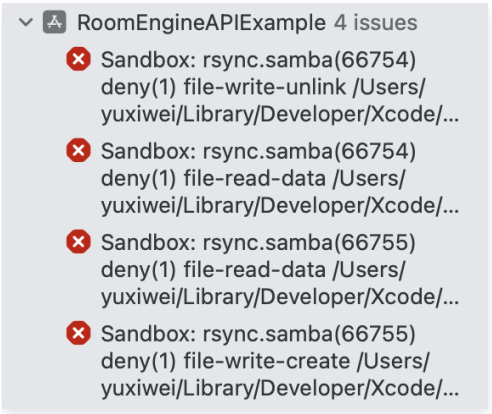
然后在项目需要使用 SDK API 的文件里，引入 RoomEngine 模块。

```
import RTCRoomEngine
```

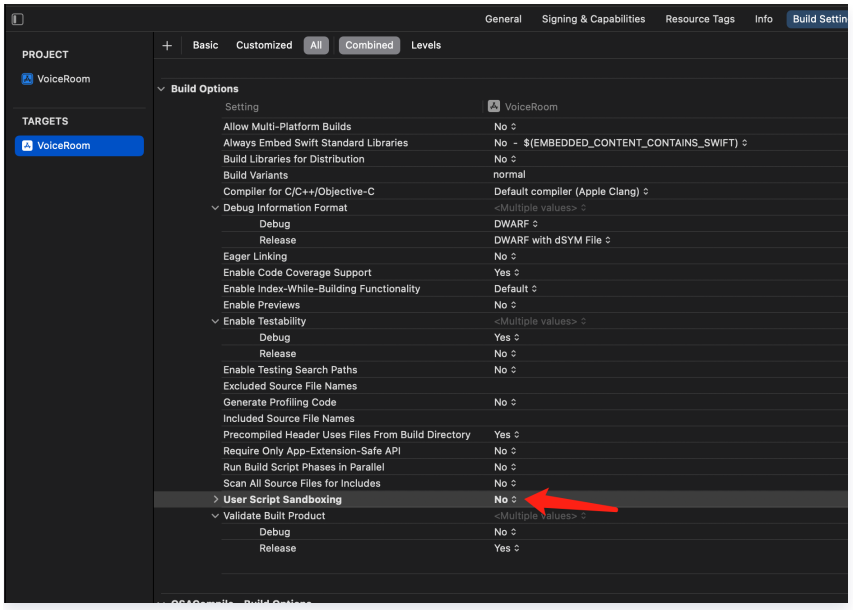
常见问题

Xcode 15 开发者沙盒选项问题

Sandbox: bash(xxx) deny(1) file-write-create



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



基础功能
登录

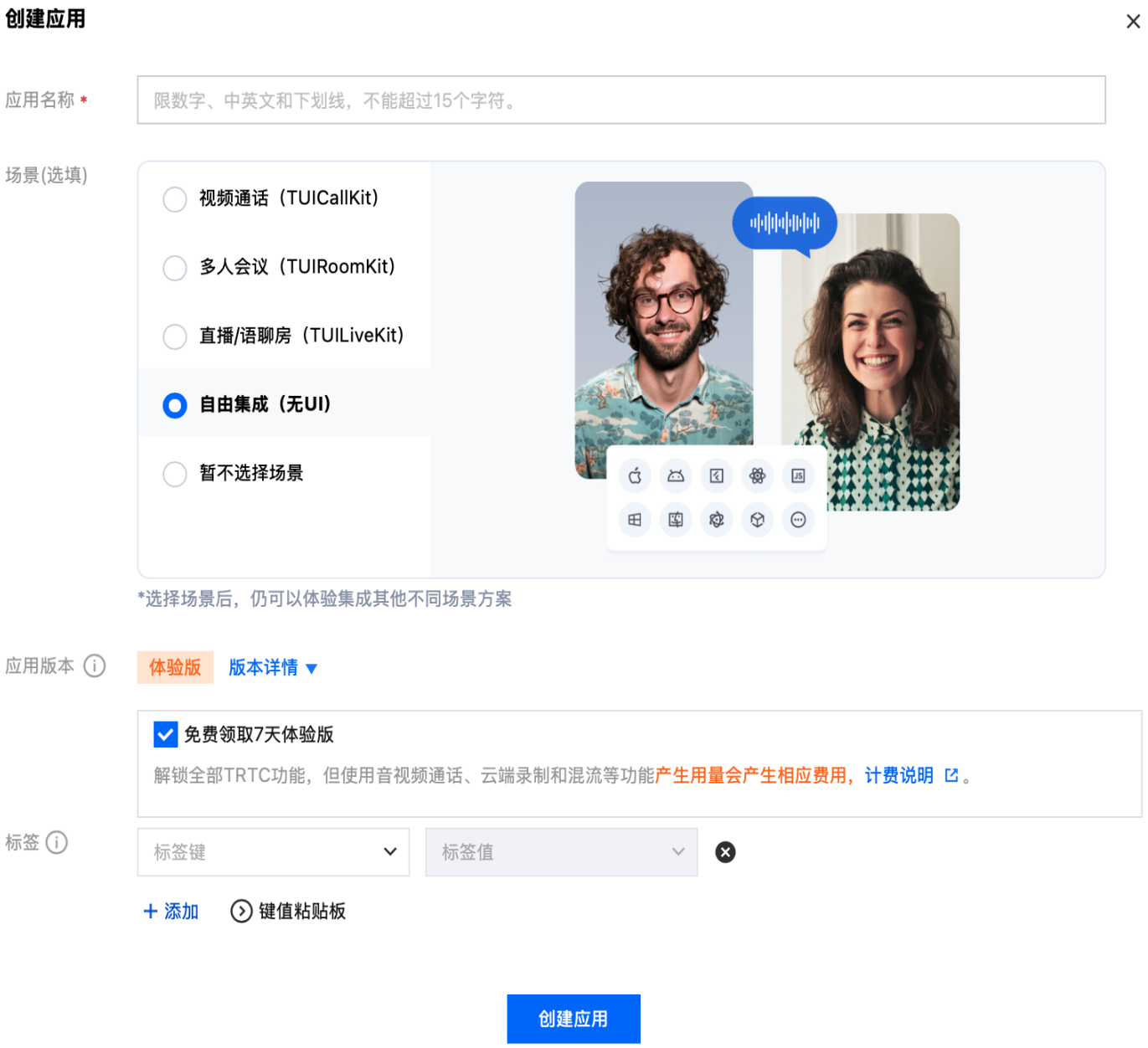
最近更新时间：2025-05-22 16:39:43

准备 SDKAppID

SDKAppID 是用于区分客户账号的唯一标识。我们建议每一个独立的 App 都申请一个新的 SDKAppID。不同 SDKAppID 之间的消息是天然隔离的，不能互通。您必须拥有正确的 SDKAppID，才能进行初始化。您可以在 实时音视频控制台 查看所有的 SDKAppID，单击**创建应用**按钮，创建新的 SDKAppID。

在弹窗中输入您的**应用名称**，选择**自由集成（无UI）**场景，勾选**免费领取7天体验版**，单击**创建应用**。

> **说明：**
> 领取体验版后，将解锁全部音视频通话功能。



功能描述

您需要调用 SDK 登录接口，验证账号身份，获得账号的功能使用权限。登录 SDK 成功后，才能进入房间，并进行一系列的操作。

登录

首次登录账号时，不需要先注册这个账号，直接登录即可，SDK 在登录过程中发现是未注册的账号，会自动注册。您可以调用 login(iOS & Mac / Android) 接口进行登录。
login 接口的关键参数如下：

参数	含义	说明
UserID	登录用户唯一标识	建议只包含大小写英文字母（a-z、A-Z）、数字（0-9）、下划线（_）和连词符（-）。长度不超过 32 字节。
UserSig	登录票据	由您的业务服务器进行计算以保证安全。计算方法请参考 用户鉴权 。

您需要在以下场景调用 login 接口：

- App 启动后首次使用 SDK 的功能。
- 在线时被踢下线：用户在线情况下被踢，SDK 会通过 `onKickedOffline` 回调通知给您，此时可以在 UI 提示用户，并调用 login 重新登录。

以下场景无需调用 login 接口：

- 用户的网络断开并重新连接后，不需要调用 login 函数，SDK 会自动上线。
- 当一个登录过程在进行时，不需要进行重复登录。

> **说明：**
> 1. 调用 SDK 接口成功登录后，将会开始计算 MAU，请根据业务场景合理调用登录接口，避免出现 MAU 过高的情况。
> 2. 在一个 App 中，SDK 不支持多个账号同时在线，如果同时登录多个账号，只有最后登录的账号在线。

示例代码如下：

iOS

```
let sdkAppId = 1400000000 // 请设置自己应用的 sdkAppID
let userId = "your user id"
let userSig = "userSig from your server"

TUIRoomEngine.login(sdkAppId: sdkAppId, userId: userId, userSig: userSig) {
    print("success")
} onError: { code, message in
    // 如果返回以下错误码，表示使用 UserSig 已过期，请您使用新签发的 UserSig 进行再次登录。
    // 1. ERR_USER_SIG_EXPIRED(6206)
    // 2. ERR_SVR_ACCOUNT_USERSIG_EXPIRED(70001)
    // 注意：其他的错误码，请不要在这里调用登录接口，避免 SDK 登录进入死循环。
    print("failure, code:\(code), message:\(message)")
}
```

Android

```
Context context = getApplicationContext();
int sdkAppId = 1400000000; // 请设置自己应用的 sdkAppID
String userId = "your user id";
String userSig = "userSig from your server";

TUIRoomEngine.login(context, sdkAppId, userId, userSig,
        new TUIRoomDefine().ActionCallback() {
        @Override
        public void onSuccess() {
            Log.i("sdk", "success");
        }

        @Override
        public void onError(TUICommonDefine.Error error, String message) {
            // 如果返回以下错误码，表示使用 UserSig 已过期，请您使用新签发的 UserSig 进行再次登录。
            // 1. ERR_USER_SIG_EXPIRED(6206)
            // 2. ERR_SVR_ACCOUNT_USERSIG_EXPIRED(70001)
            // 注意：其他的错误码，请不要在这里调用登录接口，避免 SDK 登录进入死循环。
```

```
            Log.i("sdk", "failure code:" + error + ",message:" + message);
        }
    });
```

获取登录用户的基本信息

在登录成功后，通过调用 getSelfInfo(iOS & Mac / Android) 获取登录用户信息的基本信息。如果登录失败，获取的登录用户 UserID 为空。
示例代码如下：

iOS

```
// 获取登录成功的用户基本信息
let selfUserInfo = TUIRoomEngine.getSelfInfo()
```

Android

```
TUIRoomDefine.LoginUserInfo loginUserInfo = TUIRoomEngine.getSelfInfo();
```

登出

普通情况下，如果您的应用生命周期跟 SDK 生命周期一致，退出应用前可以不登出，直接退出即可。但有些特殊场景，例如您只在进入特定界面后才使用 SDK，退出界面后不再使用，此时可以调用 logout (iOS & Mac / Android) 接口登出 SDK。
示例代码如下：

iOS

```
TUIRoomEngine.logout {
    print("success")
} onError: { code, message in
    print("failure, code:\(code), message:\(message)")
}
```

Android

```
TUIRoomEngine.logout(new TUIRoomDefine.ActionCallback() {
    @Override
    public void onSuccess() {
        Log.i("sdk","success");
    }

    @Override
    public void onError(TUICommonDefine.Error error, String message) {
        Log.i("sdk","error" + error + ",message" + message);
    }
});
```

账号切换

如果您希望在应用中实现账号切换的需求，只需要每次切换账号时调用 login 即可。
例如已经登录了 alice，现在要切换到 bob，只需要直接 login bob 即可。login bob 前无需显式调用 logout alice，SDK 内部会自动处理。

视频开播与观看

最近更新时间：2025-06-03 17:32:42

本文档主要介绍如何使用 `RTC Room Engine` SDK 实现 视频开播与观看功能。

前提条件

在使用 `RTC RoomEngine` SDK 前，您需要先调用 登录 SDK，以便后续功能正常使用。

视频开播

使用 `RTC Room Engine` 实现语聊开播核心步骤需要4步：创建并加入直播间、上麦推流、开启媒体设备、设置本地预览画面，下面将进行详细步骤介绍。

步骤1： 创建并加入直播间

iOS

在开播前您需要填写关键参数 `TUIRoomInfo`，接下来进行详细介绍：

参数：TUIRoomInfo

`TUIRoomInfo` 由很多的字段构成，但通常您只需要关心如下几个字段的填写：

参数名称	字段含义	补充说明	数据类型	填写示例
roomId	房间 ID	只允许包含大小写英文字母（a-z、A-Z）、数字（0-9）及下划线和连词符。 该字段为创建房间时**必填参数**，最大支持48个字节。	字符串	"live_100001"
name	房间名称	字符串类型的房间名称。(未填写时和 roomId 一致)	字符串	"denny`s room"
seatMode	上麦模式	该字段只有开启麦位控制后生效。 分为“自由上麦模式(freeToTake)”和“申请上麦模式(applyToTake)”。自由上麦模式下台下观众可以自由上麦，无需申请。上麦模式下台下观众上麦需要房主或同意后才能上麦。	枚举值	TUISeatMode.applyToTake
maxSeatCount	最大麦位数	数字类型的最大麦位数，这里指最多 maxSeatCount 个人同时在麦上。 最大麦位数和购买的套餐包连麦人数上限一致。	数字	10
isSeatEnabled	是否开启麦位控制	布尔类型的麦位控制使能标志符。	布尔值	true
roomType	房间类型	分为“会议(conference)”和“直播(live)”两种房间类型，语聊属于直播，因此这里使用 live。	枚举值	TUIRoomType.live

在准备好参数 `TUIRoomInfo` 后，就可以调用 `createRoom` 和 `enterRoom` 接口函数创建并加入直播间了。

Android

在开播前您需要填写关键参数 `RoomInfo`，接下来进行详细介绍：

参数：RoomInfo

`RoomInfo` 由很多的字段构成，但通常您只需要关心如下几个字段的填写：

参数名称	字段含义	补充说明	数据类型	填写示例
roomId	房间 ID	只允许包含大小写英文字母（a-z、A-Z）、数字（0-9）及下划线和连词符。 该字段为创建房间时**必填参数**，最大支持48个字节。	字符串	"live_100001"
name	房间名称	字符串类型的房间名称。(未填写时和roomId一致)	字符串	"denny`s room"
seatMode	上麦模式	该字段只有开启麦位控制后生效。 分为“自由上麦模式(FREE_TO_TAKE)”和“申请上麦模式(APPLY_TO_TAKE)”。自由上麦模式下台下观众可以自由上麦，无需申请。上麦模式下台下观众上麦需要房主或同意后才能上麦。	枚举值	SeatMode.APPLY_TO_TAKE
maxSeatCount	最大麦位数	数字类型的最大麦位数，这里指最多maxSeatCount 个人同时在麦上。 最大麦位数和购买的 套餐包 连麦人数上限一致。	数字	10
isSeatEnabled	是否开启麦位控制	布尔类型的麦位控制使能标志符。	布尔值	true
roomType	房间类型	分为“会议(CONFERENCE)”和“直播(LIVE)”两种房间类型，语聊属于直播，因此这里使用 live。	枚举值	RoomType.LIVE

在准备好参数 `TUIRoomInfo` 后，就可以调用 `createRoom` 和 `enterRoom` 接口函数创建并加入直播间了。

iOS

```
import RTCRoomEngine

let roomInfo = TUIRoomInfo()
roomInfo.roomId = "video_100001"  // 请替换成您自己的房间ID
roomInfo.name = "denny`s room"    // 请替换成您自己的房间名称
roomInfo.seatMode = .applyToTake  // 通常视频直播场景下连麦都采用申请上麦模式，若您的业务中观众无需申请才能上麦时，此处可改写为freeToTake
roomInfo.maxSeatCount = 10        // 请替换成您购买的Live套餐包的最大麦位数
roomInfo.isSeatEnabled = true     // 如果您不需要麦位控制，可以将isSeatEnabled置为false
roomInfo.roomType = .live         // 房间类型请确保是直播(live)类型
TUIRoomEngine.sharedInstance().createRoom(roomInfo) { [weak self] in
    guard let self = self else { return }
```

```
    TUIRoomEngine.sharedInstance.enterRoom(self.roomId, roomType: .live) { [weak self]
roomInfo in
        guard let self = self else { return }
        // 加入直播间成功
    } onError: { code, message in
        // 加入直播间失败
    }
} onError: { code, message in
    // 创建直播间失败
}
```

Android

```
TUIRoomDefine.RoomInfo roomInfo = new TUIRoomDefine.RoomInfo();
roomInfo.roomId = "video_100001";  // 请替换成您自己的房间ID
roomInfo.name = "denny`s room";     // 请替换成您自己的房间名称
roomInfo.seatMode = TUIRoomDefine.SeatMode.APPLY_TO_TAKE;  // 通常视频直播场景下连麦都采用申请上麦
模式，若您的业务中观众无需申请才能上麦时，此处可改写为freeToTake
roomInfo.maxSeatCount = 10;        // 请替换成您购买的Live套餐包的最大麦位数
roomInfo.isSeatEnabled = true;      // 如果您不需要麦位控制，可以将isSeatEnabled置为false
roomInfo.roomType = TUIRoomDefine.RoomType.LIVE; // 房间类型请确保是直播(live)类型

TUIRoomEngine.sharedInstance().createRoom(roomInfo, new TUIRoomDefine.ActionCallback() {
    @Override
    public void onSuccess() {
        TUIRoomEngine.sharedInstance().enterRoom(roomInfo.roomId,
TUIRoomDefine.RoomType.LIVE, new TUIRoomDefine.GetRoomInfoCallback() {
            @Override
            public void onSuccess(TUIRoomDefine.RoomInfo roomInfo) {
                // 加入直播间成功
            }
            @Override
            public void onError(TUICommonDefine.Error error, String message) {
                // 加入直播间失败
            }
        });
    }
    @Override
    public void onError(TUICommonDefine.Error error, String message) {
        // 创建直播间失败
    }
});
```

混流直播：

如果您想要开始一个混流直播间，您需要在创建房间前调用实验性接口来开启混流设置。

```
private func enableUnlimitedRoom() {
    var jsonObject = [String: Any]()
    jsonObject["api"] = "enableUnlimitedRoom"
    var params = [String: Any]()
    params["enable"] = true
```

```
    jsonObject["params"] = params
    if let jsonData = try? JSONSerialization.data(withJSONObject: jsonObject, options: []),
       let jsonString = String(data: jsonData, encoding: .utf8) {
        TUIRoomEngine.callExperimentalAPI(jsonStr: jsonString)
    }
}
```

步骤2：上麦推流

只有上麦后才能推流，因此在您成功创建房间后，您需要调用 `takeSeat` 接口上麦并开始推流。

iOS

```
import RTCRoomEngine

let index = -1    // 请将这里替换成您想要申请的麦位号，填写-1时表示无须关注麦位索引
let timeout = 30 // 请将这里替换成您申请上麦的超时时间，单位秒，如果设置为 0，SDK 不会做超时检测，也不会触发超时回调

TUIRoomEngine.sharedInstance().takeSeat(index, timeout: TimeInterval(timeout)) { requestId, userId in
    // 上麦成功
} onRejected: { requestId, userId, messagae in
    // 上麦请求被拒绝
} onCancelled: { requestId, userId in
    // 上麦请求被取消
} onTimeout: { requestId, userId in
    // 上麦请求已超时
} onError: { requestId, userId, code, message in
    // 上麦失败
}
```

Android

```
int index = -1;    // 请将这里替换成您想要申请的麦位号，填写-1时表示无须关注麦位索引
int timeout = 30; // 请将这里替换成您申请上麦的超时时间，单位秒，如果设置为 0，SDK 不会做超时检测，也不会触发超时回调

TUIRoomEngine.sharedInstance().takeSeat(index, timeout, new TUIRoomDefine.RequestCallback()
{
    @Override
    public void onAccepted(String requestId, String userId) {
        // 上麦成功
    }
    @Override
    public void onRejected(String requestId, String userId, String message) {
        // 上麦请求被拒绝
    }

    @Override
    public void onCancelled(String requestId, String userId) {
        // 上麦请求被取消
```

```
    }
    @Override
    public void onTimeout(String requestId, String userId) {
        // 上麦请求已超时
    }
    @Override
    public void onError(String requestId, String userId, TUICommonDefine.Error error, String
message) {
        // 上麦失败
    }
});
```

步骤3：开启媒体设备

iOS

在开播后，您还需要调用 `openLocalMicrophone` 和 `openLocalCamera` 接口开启麦克风和摄像头，确保观众可以看到您的画面和听到您的声音。
`openLocalMicrophone` 接口需要传入音频质量参数 `quality` 。
`quality` 是 `TUIAudioQuality` 类型的枚举。

枚举值类型	含义
speech	人声模式。单声道；音频裸码率：18kbps；适合语音通话为主的场景。
default	默认模式。单声道；音频裸码率：50kbps；SDK 默认的音频质量，如无特殊需求推荐选择之。
music	音乐模式。双声道 + 全频带；音频裸码率：128kbps；适合需要高保真传输音乐的场景，例如在线 K 歌、音乐直播等。

`openLocalCamera` 接口需要传入 选择前后置摄像头 `isFront` 和 视频质量 `quality` 两个参数。
`isFront` 是一个布尔值，true为打开前置摄像头，false为打开后置摄像头。
`quality` 是 `TUIVideoQuality` 类型的枚举。

枚举值类型	含义
quality360P	低清360P。
quality540P	标清540P。
quality720P	高清720P。
quality1080P	超清1080P。

下面以音频质量为 `default` ，打开前置摄像头并且视频质量为 `quality1080P` 为例，打开本地麦克风和摄像头。

Android

在开播后，您还需要调用 `openLocalMicrophone` 和 `openLocalCamera` 接口开启麦克风和摄像头，确保观众可以看到您的画面和听到您的声音。
`openLocalMicrophone` 接口需要传入音频质量参数 `quality` 。

`quality` 是 `AudioQuality` 类型的枚举。

枚举值类型	含义
SPEECH	人声模式。单声道；音频裸码率：18kbps；适合语音通话为主的场景。
DEFAULT	默认模式。单声道；音频裸码率：50kbps；SDK 默认的音频质量，如无特殊需求推荐选择之。
MUSIC	音乐模式。双声道 + 全频带；音频裸码率：128kbps；适合需要高保真传输音乐的场景，例如在线 K 歌、音乐直播等。

`openLocalCamera` 接口需要传入 选择前后置摄像头 `isFront` 和 视频质量 `quality` 两个参数。
`isFront` 是一个布尔值，true 为打开前置摄像头，false 为打开后置摄像头。
`quality` 是 `VideoQuality` 类型的枚举。

枚举值类型	含义
Q_360P	低清360P。
Q_540P	标清540P。
Q_720P	高清720P。
Q_1080P	超清1080P。

下面以音频质量为 `DEFAULT`，打开前置摄像头并且视频质量为 `Q_1080P` 为例，打开本地麦克风和摄像头。

iOS

```
import RTCRoomEngine

// 打开本地麦克风
let audioQuality: TUIAudioQuality = .default
TUIRoomEngine.sharedInstance().openLocalMicrophone(audioQuality) {
    // 打开麦克风成功
} onError: { code, message in
    // 打开麦克风失败
}

// 打开前置摄像头
let isFrontCamera = true
let videoQuality: TUIVideoQuality = .quality1080P
TUIRoomEngine.sharedInstance().openLocalCamera(isFront: isFrontCamera, quality:
videoQuality) {
    // 打开前置摄像头成功
} onError: { code, message in
    // 打开前置摄像头失败
}
```

Android

```
// 打开本地麦克风
TUIRoomDefine.AudioQuality audioQuality = TUIRoomDefine.AudioQuality.MUSIC;
```

```
TUIRoomEngine.sharedInstance().openLocalMicrophone(audioQuality, new
TUIRoomDefine.ActionCallback() {
    @Override
    public void onSuccess() {
        // 打开麦克风成功
    }
    @Override
    public void onError(TUICommonDefine.Error error, String message) {
        // 打开麦克风失败
    }
});


// 打开前置摄像头
boolean isFrontCamera = true;
TUIRoomDefine.VideoQuality videoQuality = TUIRoomDefine.VideoQuality.Q_1080P;
TUIRoomEngine.sharedInstance().openLocalCamera(isFrontCamera, videoQuality, new
TUIRoomDefine.ActionCallback() {
    @Override
    public void onSuccess() {
        // 打开前置摄像头成功
    }
    @Override
    public void onError(TUICommonDefine.Error error, String message) {
        // 打开前置摄像头失败
    }
});
```

步骤4：设置本地预览画面

若您需要查看本地的预览画面，可通过 调用 `setLocalVideo` 接口实现。

iOS

```
import RTCRoomEngine

let videoView = UIView()
// ...将 viewoView 添加到您的视图上并对其进行布局
TUIRoomEngine.sharedInstance().setLocalVideoView(view: videoView)
```

Android

```
TUIVideoView videoView = new TUIVideoView(context);
// ...将 viewoView 添加到您的视图上并对其进行布局
TUIRoomEngine.sharedInstance().setLocalVideoView(videoView);
```

新选项

视频观看

步骤1： 加入直播间

您仅需要调用 `enterRoom` 接口成功进房即可收听到视频主播的声音，若您还想观看主播的视频画面，请确保进房成功后执行了步骤2。
enterRoom 您需要传入两个参数：想要收听的主播所在房间的房间 ID 和房间类型。

> **说明：**
> 房间类型有两种：会议（conference）和直播（live），语聊房属于直播房间，因此您在调用 `enterRoom` 进房语聊房时，房间类型需要传入 `live` 。

iOS

```
import RTCRoomEngine

let roomId = "video_100001"
let roomType = .live

TUIRoomEngine.sharedInstance().enterRoom(roomId, roomType: roomType) { roomInfo in
    // 进房成功
} onError: { code, message in
    // 进房失败
}
```

Android

```
String roomId = "video_100001";
TUIRoomDefine.RoomType roomType = TUIRoomDefine.RoomType.LIVE;

TUIRoomEngine.sharedInstance().enterRoom(roomId, roomType, new
TUIRoomDefine.GetRoomInfoCallback() {
    @Override
    public void onSuccess(TUIRoomDefine.RoomInfo roomInfo) {
        // 进房成功
    }
    @Override
    public void onError(TUICommonDefine.Error error, String message) {
        // 进房失败
    }
});
```

新选项

步骤2：设置观看视图

iOS

您可调用 `setRemoteVideoView` 设置远端视频观看视图，传入三个参数： 观看的用户的用户 ID、播放的视频流类型和用来播放视频的视图对象。
若您期望观看的用户的房主，则用户 ID 为 步骤1 进房成功后的 roomInfo.ownerId。
其中视频流类型是一个 `TUIVideoStreamType` 类型的枚举值。

枚举值类型	含义
cameraStream	高清摄像头视频流。
screenStream	屏幕分享视频流。
cameraStreamLow	低清摄像头视频流。

Android

您可调用 `setRemoteVideoView` 设置远端视频观看视图，传入三个参数： 观看的用户的用户 ID、播放的视频流类型和用来播放视频的视图对象。
若您期望观看的用户的房主，则用户 ID 为 步骤1 进房成功后的 roomInfo.ownerId。
其中视频流类型是一个 `VideoStreamType` 类型的枚举值。

枚举值类型	含义
CAMERA_STREAM	高清摄像头视频流。
SCREEN_STREAM	屏幕分享视频流。
CAMERA_STREAM_LOW	低清摄像头视频流。

以观看房主的高清摄像头视频流为例：

iOS

```
import RTCRoomEngine

let videoView = UIView()
// ...将 viewoView 添加到您的视图上并对其进行布局

let ownerId = ""    // 请将其替换成您加入的直播间的房主用户Id
let streamType = TUIVideoStreamType.cameraStream
TUIRoomEngine.sharedInstance().setRemoteVideoView(userId: ownerId,
                                                  streamType: streamType,
                                                  view: videoView)
```

Android

```
TUIVideoView videoView = new TUIVideoView(context);
// ...将 viewoView 添加到您的视图上并对其进行布局
```

```
String ownerId = "";    // 请将其替换成您加入的直播间的房主用户Id
TUIRoomDefine.VideoStreamType streamType = TUIRoomDefine.VideoStreamType.CAMERA_STREAM;
TUIRoomEngine.sharedInstance().setRemoteVideoView(ownerId, streamType, videoView);
```

步骤3：观看主播视频画面

您可调用 `startPlayRemoteVideo` 接口观看远端用户的视频流，播放的画面将在 步骤2 设置的视图对象上展示。
`startPlayRemoteVideo` 需要传入两个视频参数： 观看的用户的用户Id和播放的视频流类型。
以播放房主的高清视频流为例：

iOS

```
import RTCRoomEngine
let ownerId = ""    // 请将其替换成您加入的直播间的房主用户Id
let streamType = TUIVideoStreamType.cameraStream
TUIRoomEngine.sharedInstance().startPlayRemoteVideo(userId: ownerId,
                                                    streamType: streamType) { userId in
    // 播放视频画面中
} onLoading: { userId in
    // 视频画面加载中
} onError: { userId, code, message in
    // 视频画面播放失败
}
```

Android

```
String ownerId = "";    // 请将其替换成您加入的直播间的房主用户Id
TUIRoomDefine.VideoStreamType streamType = TUIRoomDefine.VideoStreamType.CAMERA_STREAM;

TUIRoomEngine.sharedInstance().startPlayRemoteVideo(ownerId, streamType, new
TUIRoomDefine.PlayCallback() {
    @Override
    public void onPlaying(String userId) {
        // 播放视频画面中
    }
    @Override
    public void onLoading(String userId) {
        // 视频画面加载中
    }
    @Override
    public void onPlayError(String userId, TUICommonDefine.Error error, String message) {
        // 视频画面播放失败
    }
});
```

语聊开播与收听

最近更新时间：2025-05-22 16:39:43

本文档主要介绍如何使用 `RTC Room Engine` SDK 实现语聊开播与收听功能。

前提条件

在使用 `RTC Room Engine` SDK 前，您需要先调用 登录 SDK，以便后续功能正常使用。

音频开播

使用 `RTC Room Engine` 实现语聊开播核心步骤需要3步：创建并加入直播间、上麦推流、开启媒体设备，下面将对齐进行详细介绍。

步骤1： 创建并加入直播间

iOS

在开播前您需要填写关键参数 `TUIRoomInfo`，接下来进行详细介绍：

参数：TUIRoomInfo

`TUIRoomInfo` 由很多的字段构成，但通常您只需要关心如下几个字段的填写：

参数名称	字段含义	补充说明	数据类型	填写示例
roomId	房间ID	只允许包含大小写英文字母（a-z、A-Z）、数字（0-9）及下划线和连词符。 该字段为创建房间时**必填参数**，最大支持48个字节。	字符串	"live_100001"
name	房间名称	字符串类型的房间名称。(未填写时和roomId 一致)	字符串	"denny`s room"
seatMode	上麦模式	该字段只有开启麦位控制后生效。 分为“自由上麦模式(freeToTake)”和“申请上麦模式(applyToTake)”。自由上麦模式下台下观众可以自由上麦，无需申请。上麦模式下台下观众上麦需要房主或同意后才能上麦。	枚举值	TUISeatMode.applyToTake
maxSeatCount	最大麦位数	数字类型的最大麦位数，这里指最多maxSeatCount 个人同时在麦上。 最大麦位数和购买的 套餐包 连麦人数上限一致。	数字	10
isSeatEnabled	是否开启麦位控制	布尔类型的麦位控制使能标志符。	布尔值	true
roomType	房间类型	分为“会议(conference)”和“直播(live)”两种房间类型，语聊属于直播，因此这里使用live。	枚举值	TUIRoomType.live

在准备好参数 `TUIRoomInfo` 后，就可以调用 `createRoom` 和 `enterRoom` 接口函数创建并加入直播间了。

Android

在开播前您需要填写关键参数 `RoomInfo`，接下来进行详细介绍：

参数：RoomInfo

`RoomInfo` 由很多的字段构成，但通常您只需要关心如下几个字段的填写：

参数名称	字段含义	补充说明	数据类型	填写示例
roomId	房间ID	只允许包含大小写英文字母（a-z、A-Z）、数字（0-9）及下划线和连词符。 该字段为创建房间时**必填参数**，最大支持48个字节。	字符串	"live_100001"
name	房间名称	字符串类型的房间名称。(未填写时和roomId一致)	字符串	"denny`s room"
seatMode	上麦模式	该字段只有开启麦位控制后生效。 分为“自由上麦模式(FREE_TO_TAKE)”和“申请上麦模式(APPLY_TO_TAKE)”。自由上麦模式下台下观众可以自由上麦，无需申请。 上麦模式下台下观众上麦需要房主或同意后才能上麦。	枚举值	SeatMode.APPLY_TO_TAKE
maxSeatCount	最大麦位数	数字类型的最大麦位数，这里指最多maxSeatCount个人同时在麦上。 最大麦位数和购买的套餐包连麦人数上限一致。	数字	10
isSeatEnabled	是否开启麦位控制	布尔类型的麦位控制使能标志符。	布尔值	true
roomType	房间类型	分为“会议(CONFERENCE)”和“直播(LIVE)”两种房间类型，语聊属于直播，因此这里使用live。	枚举值	RoomType.Live

在准备好参数 `RoomInfo` 后，就可以调用 `createRoom` 和 `enterRoom` 接口函数创建并加入直播间了。

iOS

```
import RTCRoomEngine

let roomInfo = TUIRoomInfo()
roomInfo.roomId = "voice_100001"  // 请替换成您自己的房间ID
roomInfo.name = "denny`s room"    // 请替换成您自己的房间名称
roomInfo.seatMode = .applyToTake  // 也可根据您的需求替换为.freeToTake
roomInfo.maxSeatCount = 10        // 请替换成您购买的Live套餐包的最大麦位数
roomInfo.isSeatEnabled = true     // 如果您不需要麦位控制，可以将isSeatEnabled置为false
roomInfo.roomType = .live         // 房间类型请确保是直播(live)类型

TUIRoomEngine.sharedInstance().createRoom(roomInfo) { [weak self] in
    guard let self = self else { return }
```

```
    TUIRoomEngine.sharedInstance.enterRoom(self.roomId, roomType: .live) { [weak self]
roomInfo in
        guard let self = self else { return }
        // 加入直播间成功
    } onError: { code, message in
        // 加入直播间失败
    }
} onError: { code, message in
    // 创建直播间失败
}
```

Android

```
TUIRoomDefine.RoomInfo roomInfo = new TUIRoomDefine.RoomInfo();
roomInfo.roomId = "voice_100001";  // 请替换成您自己的房间ID
roomInfo.name = "denny`s room";    // 请替换成您自己的房间名称
roomInfo.seatMode = TUIRoomDefine.SeatMode.APPLY_TO_TAKE;  // 也可根据您的需求替换为.freeToTake
roomInfo.maxSeatCount = 10;        // 请替换成您购买的Live套餐包的最大麦位数
roomInfo.isSeatEnabled = true;     // 如果您不需要麦位控制，可以将isSeatEnabled置为false
roomInfo.roomType = TUIRoomDefine.RoomType.LIVE;          // 房间类型请确保是直播(live)类型

TUIRoomEngine.sharedInstance().createRoom(roomInfo, new TUIRoomDefine.ActionCallback() {
    @Override
    public void onSuccess() {
        TUIRoomEngine.sharedInstance().enterRoom(roomInfo.roomId,
TUIRoomDefine.RoomType.LIVE, new TUIRoomDefine.GetRoomInfoCallback() {
            @Override
            public void onSuccess(TUIRoomDefine.RoomInfo roomInfo) {
                // 加入直播间成功
            }
            @Override
            public void onError(TUICommonDefine.Error error, String message) {
                // 加入直播间失败
            }
        });
    }
    @Override
    public void onError(TUICommonDefine.Error error, String message) {
        // 创建直播间失败
    }
});
```

步骤2：上麦推流

只有上麦后才能推流，因此在您成功创建房间后，您需要调用 `takeSeat` 接口上麦并开始推流。

iOS

```
import RTCRoomEngine

let index = 0    // 请将这里替换成您想要申请的麦位号
```

```
let timeout = 30 // 请将这里替换成您申请上麦的超时时间，单位秒，如果设置为 0，SDK 不会做超时检测，也不会触发超时回调

TUIRoomEngine.sharedInstance().takeSeat(index, timeout: TimeInterval(timeout)) { requestId, userId in
    // 上麦成功
} onRejected: { requestId, userId, messagae in
    // 上麦请求被拒绝
} onCancelled: { requestId, userId in
    // 上麦请求被取消
} onTimeout: { requestId, userId in
    // 上麦请求已超时
} onError: { requestId, userId, code, message in
    // 上麦失败
}
```

Android

```
int index = 0;     // 请将这里替换成您想要申请的麦位号
int timeout = 30; // 请将这里替换成您申请上麦的超时时间，单位秒，如果设置为 0，SDK 不会做超时检测，也不会触发超时回调

TUIRoomEngine.sharedInstance().takeSeat(index, timeout, new TUIRoomDefine.RequestCallback()
{
    @Override
    public void onAccepted(String requestId, String userId) {
        // 上麦成功
    }
    @Override
    public void onRejected(String requestId, String userId, String message) {
        // 上麦请求被拒绝
    }

    @Override
    public void onCancelled(String requestId, String userId) {
        // 上麦请求被取消
    }
    @Override
    public void onTimeout(String requestId, String userId) {
        // 上麦请求已超时
    }
    @Override
    public void onError(String requestId, String userId, TUICommonDefine.Error error, String message) {
        // 上麦失败
    }
});
```

步骤3：开启媒体设备

iOS

在开播后，您还需要调用 `openLocalMicrophone` 连个接口开启麦克风，传入一个 `TUIAudioQuality` 类型的参数 `quality` ，确保观众可以听到您的声音。
`TUIAudioQuality` 是一个枚举。

参数名称	字段含义
speech	人声模式。单声道；音频裸码率：18kbps；适合语音通话为主的场景。
default	默认模式。单声道；音频裸码率：50kbps；SDK 默认的音频质量，如无特殊需求推荐选择之。
music	音乐模式。双声道 + 全频带；音频裸码率：128kbps；适合需要高保真传输音乐的场景，例如在线K歌、音乐直播等。

Android

在开播后，您还需要调用 `openLocalMicrophone` 连个接口开启麦克风，传入一个 `AudioQuality` 类型的参数 `quality` ，确保观众可以听到您的声音。
`AudioQuality` 是一个枚举。

参数名称	字段含义
SPEECH	人声模式。单声道；音频裸码率：18kbps；适合语音通话为主的场景。
DEFAULT	默认模式。单声道；音频裸码率：50kbps；SDK 默认的音频质量，如无特殊需求推荐选择之。
MUSIC	音乐模式。双声道 + 全频带；音频裸码率：128kbps；适合需要高保真传输音乐的场景，例如在线K歌、音乐直播等。

下面以音乐模式为例，打开本地麦克风。

iOS

```
import RTCRoomEngine

let audioQuality: TUIAudioQuality = .music  // 请根据您对音质的场景需求选择对应的模式
TUIRoomEngine.sharedInstance().openLocalMicrophone(audioQuality) {
    // 打开麦克风成功
} onError: { code, message in
    // 打开麦克风失败
}
```

Android

```
TUIRoomDefine.AudioQuality audioQuality  = TUIRoomDefine.AudioQuality.MUSIC;  // 请根据您对音质
的场景需求选择对应的模式
TUIRoomEngine.sharedInstance().openLocalMicrophone(audioQuality, new
TUIRoomDefine.ActionCallback() {
    @Override
    public void onSuccess() {
```

```
        // 打开麦克风成功
    }
    @Override
    public void onError(TUICommonDefine.Error error, String message) {
        // 打开麦克风失败
    }
});
```

音频收听

您仅需要调用 `enterRoom` 接口成功进房即可收听到语聊房主播的声音。
enterRoom 您需要传入两个参数：想要收听的主播所在房间的房间 ID 和房间类型。

> **说明：**
> 房间类型有两种，分别为：会议（conference）和直播（live），语聊房属于直播房间，因此您在调用 `enterRoom` 进房语聊房时，房间类型需要传入 `live` 。

iOS

```
import RTCRoomEngine

let roomId = "voice_100001" // 收听的房间Id
let roomType = .live        // 此处需设置为.live

TUIRoomEngine.sharedInstance().enterRoom(roomId, roomType: roomType) { roomInfo in
    // 进房成功
} onError: { code, message in
    // 进房失败
}
```

Android

```
String roomId = "voice_100001"; // 收听的房间Id
TUIRoomDefine.RoomType roomType = TUIRoomDefine.RoomType.LIVE; // 此处需设置为.live

TUIRoomEngine.sharedInstance().enterRoom(roomId, roomType, new
TUIRoomDefine.GetRoomInfoCallback() {
    @Override
    public void onSuccess(TUIRoomDefine.RoomInfo roomInfo) {
        // 进房成功
    }
    @Override
    public void onError(TUICommonDefine.Error error, String message) {
        // 进房失败
    }
});
```

观众连麦

最近更新时间：2025-05-22 16:39:43

本文档主要介绍如何使用 `RTC Room Engine` SDK 实现 观众连麦功能。
`RTC Room Engine` 支持如下麦位管理能力：

上麦响应配置	获取麦位列表	获取上麦请求列表
申请上麦	主动下麦	移动麦位
踢人下麦	邀请上麦	锁定麦位

前提条件

在使用 `RTC RoomEngine` SDK 前，您需要先调用 登录 SDK，以便后续功能正常使用。

使用指引

> **说明：**
> 使用麦位管理时，您需要确保您已经开播或进入到了直播间。

上麦响应配置

iOS

当您时房主时，您可以调用 `updateRoomSeatModeByAdmin` 接口实现，传入参数麦位模式 `seatMode` 即可。
seatMode 是一个 `TUISeatMode` 类型的枚举。

枚举值类型	含义
freeToTake	申请上麦无需等待房主同意，可直接上麦。
applyToTake	申请上麦需要等待房主同意才可上麦。

Android

当您时房主时，您可以调用 `updateRoomSeatModeByAdmin` 接口实现，传入参数麦位模式 `seatMode` 即可。
seatMode 是一个 `SeatMode` 类型的枚举。

枚举值类型	含义
FREE_TO_TAKE	申请上麦无需等待房主同意，可直接上麦。
APPLY_TO_TAKE	申请上麦需要等待房主同意才可上麦。

以更新上麦相应配置为申请上麦模式为例：

iOS

```
import RTCRoomEngine

let seatMode: TUISeatMode = .applyToTake    // 这里选择申请上麦模式，若您需要选择自由上麦模式，可更改为.freeToTake
TUIRoomEngine.sharedInstance().updateRoomSeatModeByAdmin(seatMode) {
    // 设置上麦响应配置成功
} onError: { code, message in
    // 设置上麦响应配置失败
}
```

Android

```
TUIRoomDefine.SeatMode seatMode = TUIRoomDefine.SeatMode.APPLY_TO_TAKE;    // 这里选择申请上麦模式，若您需要选择自由上麦模式，可更改为.freeToTake
TUIRoomEngine.sharedInstance().updateRoomSeatModeByAdmin(seatMode, new TUIRoomDefine.ActionCallback() {
    @Override
    public void onSuccess() {
        // 设置上麦响应配置成功
    }
    @Override
    public void onError(TUICommonDefine.Error error, String message) {
        // 设置上麦响应配置失败
    }
});
```

获取麦位列表

您可以调用 `getSeatList` 接口获取当前的麦位列表信息。

iOS

```
import RTCRoomEngine

TUIRoomEngine.sharedInstance().getSeatList { seatList in
    // 获取麦位列表成功
} onError: { code, messagea in
    // 获取麦位列表失败
}
```

Android

```
TUIRoomEngine.sharedInstance().getSeatList(new TUIRoomDefine.GetSeatListCallback() {
    @Override
    public void onSuccess(List<TUIRoomDefine.SeatInfo> list) {
        // 获取麦位列表成功
    }
    @Override
    public void onError(TUICommonDefine.Error error, String message) {
        // 获取麦位列表失败
```

```
    }
});
```

获取上麦请求列表

您可以调用 `getSeatApplicationList` 接口获取当前的上麦请求列表信息。

iOS

```
import RTCRoomEngine

TUIRoomEngine.sharedInstance().getSeatApplicationList { applications in
    // 获取上麦请求列表成功
} onError: { code, message in
    // 获取上麦请求列表失败
}
```

Android

```
TUIRoomEngine.sharedInstance().getSeatApplicationList(new
TUIRoomDefine.RequestListCallback() {
    @Override
    public void onSuccess(List<TUIRoomDefine.Request> list) {
        // 获取上麦请求列表成功
    }
    @Override
    public void onError(TUICommonDefine.Error error, String message) {
        // 获取上麦请求列表失败
    }
});
```

申请上麦

当您未上麦时，您可通过调用 `takeSeat` 接口实现申请上麦，传入两个参数：想要申请的麦位索引和超时时长。
以申请上1号麦为例：

iOS

```
import RTCRoomEngine

let index = 1     // 请将这里替换成您想要申请的麦位号
let timeout = 30 // 请将这里替换成您申请上麦的超时时间，单位秒，如果设置为 0，SDK 不会做超时检测，也不会触
发超时回调
TUIRoomEngine.sharedInstance().takeSeat(index, timeout: TimeInterval(timeout)) { requestId,
userId in
    // 上麦申请被接受
} onRejected: { requestId, userId, message in
    // 上麦申请被拒绝
} onCancelled: { requestId, userId in
    // 上麦申请被取消
} onTimeout: { requestId, userId in
```

```
    // 上麦申请已超时
} onError: { requestId, userId, code, message in
    // 上麦异常
}
```

Android

```
int index = 1;     // 请将这里替换成您想要申请的麦位号
int timeout = 30; // 请将这里替换成您申请上麦的超时时间,单位秒，如果设置为 0，SDK 不会做超时检测，也不会触发超时回调
TUIRoomEngine.sharedInstance().takeSeat(index, timeout, new TUIRoomDefine.RequestCallback()
{
    @Override
    public void onAccepted(String requestId, String userId) {
        // 上麦申请被接受
    }
    @Override
    public void onRejected(String requestId, String userId, String message) {
        // 上麦申请被拒绝
    }
    @Override
    public void onCancelled(String requestId, String userId) {
        // 上麦申请被取消
    }
    @Override
    public void onTimeout(String requestId, String userId) {
        // 上麦申请已超时
    }

    @Override
    public void onError(String requestId, String userId, TUICommonDefine.Error error, String message) {
        // 上麦异常
    }
});
```

当您是房主时，若您通过 `addObserver` 接口成为了 `RTC Room Engine` SDK的观察者，则当有人申请上麦时，您会收到 `onRequestReceived` 回调，您可通过 `responseRemoteRequest` 接收或拒绝该请求。

iOS

```
func onRequestReceived(request: TUIRequest) {
    if request.requestAction == .takeSeat {
        let agreeToTakeSeat = true    // 若您想拒绝该上麦请求，此处可置为false
        TUIRoomEngine.sharedInstance().responseRemoteRequest(request.requestId, agree: agreeToTakeSeat) {
            // 处理上麦请求成功
        } onError: { code, message in
            // 处理上麦请求失败
        }
    }
```

```
}
```

Android

```
public void onRequestReceived(TUIRoomDefine.Request request) {
    if (TUIRoomDefine.RequestAction.REQUEST_TO_TAKE_SEAT == request.requestAction) {
        boolean agreeToTakeSeat = true;    // 若您想拒绝该上麦请求，此处可置为false
        TUIRoomEngine.sharedInstance().responseRemoteRequest(request.requestId,
agreeToTakeSeat, new TUIRoomDefine.ActionCallback() {
            @Override
            public void onSuccess() {
                // 处理上麦请求成功
            }
            @Override
            public void onError(TUICommonDefine.Error error, String message) {
                // 处理上麦请求失败
            }
        });
    }
}
```

主动下麦

当您已经在麦上时，您可通过调用 `leaveSeat` 接口实现主动下麦。

iOS

```
import RTCRoomEngine

TUIRoomEngine.sharedInstance().leaveSeat {
    // 主动下麦成功
} onError: { code, message in
    // 主动下麦失败
}
```

Android

```
TUIRoomEngine.sharedInstance().leaveSeat(new TUIRoomDefine.ActionCallback() {
    @Override
    public void onSuccess() {
        // 主动下麦成功
    }
    @Override
    public void onError(TUICommonDefine.Error error, String message) {
        // 主动下麦失败
    }
});
```

移动麦位

当您已经在麦上时，您可通过调用 `moveToSeat` 接口实现移动麦位功能，传入参数：想要移动到的麦位索引。
以移动到2号麦位为例：

iOS

```
import RTCRoomEngine

let targetIndex = 2
TUIRoomEngine.sharedInstance().moveToSeat(targetSeatIndex: targetIndex) {
    // 移动麦位成功
} onError: { code, message in
    // 移动麦位失败
}
```

Android

```
int targetIndex = 2;
TUIRoomEngine.sharedInstance().moveToSeat(targetIndex, new TUIRoomDefine.ActionCallback() {
    @Override
    public void onSuccess() {
        // 移动麦位成功
    }
    @Override
    public void onError(TUICommonDefine.Error error, String message) {
        // 移动麦位失败
    }
});
```

踢人下麦

当您是房主时，您可通过调用 `kickUserOffSeatByAdmin` 接口实现该功能，传入参数：想要踢下麦的用户的用户Id。
以将麦位用户 Id 为100001的用户踢下麦为例：

iOS

```
import RTCRoomEngine

let targetIndex = -1      // 请将改值设为-1，其他值无任何意义
let userId = "100001"     // 请将其替换成您需要踢下麦的用户
TUIRoomEngine.sharedInstance().kickUserOffSeatByAdmin(targetIndex, userId: userId) {
    // 踢人下麦成功
} onError: { code, messagae in
    // 踢人下麦失败
}
```

Android

```
int targetIndex = -1;      // 请将改值设为-1，其他值无任何意义
String userId = "100001";     // 请将其替换成您需要踢下麦的用户
```

```
TUIRoomEngine.sharedInstance().kickUserOffSeatByAdmin(targetIndex, userId, new
TUIRoomDefine.ActionCallback() {
    @Override
    public void onSuccess() {
        // 踢人下麦成功
    }
    @Override
    public void onError(TUICommonDefine.Error error, String message) {
        // 踢人下麦失败
    }
});
```

邀请上麦

当您是房主时，您可通过调用 `takeUserOnSeatByAdmin` 接口，传入三个参数：想要操作的麦位索引、想要邀请的用户的用户Id和超时时长。
以邀请用户 Id 为 100002 的用户上4号麦为例：

iOS

```
import RTCRoomEngine

let targetIndex = 4
let timeout = 30  // 请将这里替换成您申请上麦的超时时间，单位秒，如果设置为 0，SDK 不会做超时检测，也不会触发超时回调
let targetUserId = "100002"  // 请替换成您想要踢下麦的主播的用户Id
TUIRoomEngine.sharedInstance().takeUserOnSeatByAdmin(tartgetIndex,
                                                     userId: targetUserId,
                                                     timeout: TimeInterval(timeout)) {
requestId, userId in
    // 上麦邀请被接受
} onRejected: { requestId, userId, message in
    // 上麦邀请被拒绝
} onCancelled: { requestId, userId in
    // 上麦邀请超时
} onTimeout: { requestId, userId in
    // 上麦邀请被取消
} onError: { requestId, userId, code, message in
    // 上麦邀请异常
}
```

Android

```
int targetIndex = 4;
int timeout = 30;  // 请将这里替换成您申请上麦的超时时间，单位秒，如果设置为 0，SDK 不会做超时检测，也不会触发超时回调
String targetUserId = "100002";  // 请替换成您想要踢下麦的主播的用户Id
TUIRoomEngine.sharedInstance().takeUserOnSeatByAdmin(targetIndex, targetUserId, timeout, new
TUIRoomDefine.RequestCallback() {
    @Override
    public void onAccepted(String requestId, String userId) {
```

```
        // 上麦邀请被接受
    }
    @Override
    public void onRejected(String requestId, String userId, String message) {
        // 上麦邀请被拒绝
    }
    @Override
    public void onCancelled(String requestId, String userId) {
        // 上麦邀请超时
    }
    @Override
    public void onTimeout(String requestId, String userId) {
        // 上麦邀请被取消
    }
    @Override
    public void onError(String requestId, String userId, TUICommonDefine.Error error, String
message) {
        // 上麦邀请异常
    }
});
```

若您通过 `addObserver` 接口成为了 `RTC Room Engine` SDK的观察者，则当有人邀请您上麦时， 您会收到 `onRequestReceived` 回调， 您可通过调用 `responseRemoteRequest` 同意/拒绝对方的上麦邀请。

iOS

```
func onRequestReceived(request: TUIRequest) {
    if request.requestAction == .remoteUserOnSeat {
        let agreeToTakeSeat = true    // 若您想拒绝该上麦邀请，此处可置为false
        TUIRoomEngine.sharedInstance().responseRemoteRequest(request.requestId,
                                                             agree: agreeToTakeSeat) {
            // 处理上麦邀请成功
        } onError: { code, message in
            // 处理上麦邀请失败
        }
    }
}
```

Android

```
public void onRequestReceived(TUIRoomDefine.Request request) {
    if (TUIRoomDefine.RequestAction.REQUEST_REMOTE_USER_ON_SEAT == request.requestAction) {
        boolean agreeToTakeSeat = true;    // 若您想拒绝该上麦邀请，此处可置为false
        TUIRoomEngine.sharedInstance().responseRemoteRequest(request.requestId,
agreeToTakeSeat, new TUIRoomDefine.ActionCallback() {
            @Override
            public void onSuccess() {
                // 处理上麦邀请成功
            }
            @Override
            public void onError(TUICommonDefine.Error error, String message) {
```

```
                // 处理上麦邀请失败
            }
        });
    }
}
```

锁定麦位

iOS

当您是房主，且想将5号空麦位锁定不让其他人上麦，或想将6号麦位上的主播静音且将7号麦位上的主播禁画时，可调用 `lockSeat` 接口，传入两个参数：将要锁定的麦位索引和锁定模式。
锁定模式(`TUISeatLockParams`)的结构如下：

属性名称	字段含义	补充说明	数据类型	填写示例
lockSeat	锁定麦位	锁定对应麦位则该麦位不允许申请上麦。	布尔值	当锁定麦位时为 true
lockVideo	锁定麦位摄像头	锁定对应麦位摄像头则该麦位不再发布视频流。	布尔值	false
lockAudio	锁定麦位麦克风	锁定对应麦位摄像头则该麦位不再发布音频流。	布尔值	当锁定麦位麦克风时为 true

Android

当您是房主，且想将5号空麦位锁定不让其他人上麦，或想将6号麦位上的主播静音且将7号麦位上的主播禁画时，可调用 `lockSeat` 接口，传入两个参数：将要锁定的麦位索引和锁定模式。
锁定模式(`SeatLockParams`)的结构如下：

属性名称	字段含义	补充说明	数据类型	填写示例
lockSeat	锁定麦位	锁定对应麦位则该麦位不允许申请上麦。	布尔值	当锁定麦位时为 true
lockVideo	锁定麦位摄像头	锁定对应麦位摄像头则该麦位不再发布视频流。	布尔值	false
lockAudio	锁定麦位麦克风	锁定对应麦位摄像头则该麦位不再发布音频流。	布尔值	当锁定麦位麦克风时为 true

您可通过调用 `lockSeatByAdmin` 接口实现上述功能：

iOS

```
import RTCRoomEngine

// 锁定麦位
let lockSeatIndex = 5
let lockSeatParam = TUISeatLockParams()
```

```
lockSeatParam.lockSeat = true
TUIRoomEngine.sharedInstance().lockSeatByAdmin(lockSeatIndex,
                                               lockMode: lockSeatParam) {
    // 锁定麦位成功
} onError: { code, message in
    // 锁定麦位失败
}

// 锁定麦位麦克风
let lockSeatAudioIndex = 6
let lockSeatAudioParam = TUISeatLockParams()
lockSeatAudioParam.lockAudio = true
TUIRoomEngine.sharedInstance().lockSeatByAdmin(lockSeatAudioIndex,
                                               lockMode: lockSeatAudioParam) {
    // 锁定麦位麦克风成功
} onError: { code, message in
    // 锁定麦位麦克风失败
}

// 锁定麦位摄像头
let lockSeatVideoIndex = 7
let lockSeatVideoParam = TUISeatLockParams()
lockSeatAudioParam.lockVideo = true
TUIRoomEngine.sharedInstance().lockSeatByAdmin(lockSeatVideoIndex,
                                               lockMode: lockSeatAudioParam) {
    // 锁定麦位摄像头成功
} onError: { code, message in
    // 锁定麦位摄像头失败
}
```

Android

```
// 锁定麦位
int lockSeatIndex = 5;
TUIRoomDefine.SeatLockParams lockSeatParam = new TUIRoomDefine.SeatLockParams();
lockSeatParam.lockSeat = true;
TUIRoomEngine.sharedInstance().lockSeatByAdmin(lockSeatIndex, lockSeatParam, new
TUIRoomDefine.ActionCallback() {
    @Override
    public void onSuccess() {
        // 锁定麦位成功
    }
    @Override
    public void onError(TUICommonDefine.Error error, String message) {
        // 锁定麦位失败
    }
});

// 锁定麦位麦克风
int lockSeatAudioIndex = 6;
TUIRoomDefine.SeatLockParams lockSeatAudioParam = new TUIRoomDefine.SeatLockParams();
lockSeatAudioParam.lockAudio = true;
```

```
TUIRoomEngine.sharedInstance().lockSeatByAdmin(lockSeatAudioIndex, lockSeatAudioParam, new
TUIRoomDefine.ActionCallback() {
    @Override
    public void onSuccess() {
        // 锁定麦位麦克风成功
    }
    @Override
    public void onError(TUICommonDefine.Error error, String message) {
        // 锁定麦位麦克风失败
    }
});

// 锁定麦位摄像头
int lockSeatVideoIndex = 7;
TUIRoomDefine.SeatLockParams lockSeatVideoParam = new TUIRoomDefine.SeatLockParams();
lockSeatAudioParam.lockVideo = true;
TUIRoomEngine.sharedInstance().lockSeatByAdmin(lockSeatVideoIndex, lockSeatAudioParam, new
TUIRoomDefine.ActionCallback() {
    @Override
    public void onSuccess() {
        // 锁定麦位摄像头成功
    }
    @Override
    public void onError(TUICommonDefine.Error error, String message) {
        // 锁定麦位摄像头失败
    }
});
```

监听回调

iOS

您可通过调用 `addObserver` 成为 `RTC Room Engine` SDK的观察者，监听麦位相关的回调。
以 `CoGuestController` 成为观察者为例：

```
import RTCRoomEngine

class CoGuestController: NSObject, TUIRoomObserver {  // 请将其替换为您的业务类，这里只做示例
    override init() {
        super.init()
        TUIRoomEngine.sharedInstance().addObserver(self)
    }

    deinit {
        TUIRoomEngine.sharedInstance().removeObserver(self)
    }

    // 房间上麦模式发生变化时触发
    func onRoomSeatModeChanged(roomId: String, seatMode: TUISeatMode) {
        // 房间麦位模式改变了，当前模式：seatMode
    }
    // 当麦位列表改变时触发
```

```
    func onSeatListChanged(seatList: [TUISeatInfo], seated seatedList: [TUISeatInfo],
left leftList: [TUISeatInfo]) {
        // 麦位列表改变了，麦上最新的用户列表:seatList，新上麦的用户列表:seatedList，新下麦的用户列
表:leftList
    }
    // 当收到其他用户的请求时触发
    func onRequestReceived(request: TUIRequest) {
        switch request.requestAction {
        case .takeSeat:
            // 收到来自request.userName的上麦请求
        case .remoteUserOnSeat:
            // 收到来自request.userName的上麦邀请
        default:
            break
        }
    }
    // 当其他用户取消请求时触发
    func onRequestCancelled(request: TUIRequest, operateUser: TUIUserInfo) {
        switch request.requestAction {
        case .takeSeat:
            // 来自request.userName的上麦请求被取消了
        case .remoteUserOnSeat:
           // 来自request.userName的上麦邀请被取消了
        default:
            break
        }
    }

    // 收到请求被其他 管理员/房主 处理时触发
    func onRequestProcessed(request: TUIRequest, operateUser: TUIUserInfo) {
        switch request.requestAction {
        case .takeSeat:
            // 来自request.userName的上麦请求被operateUser.userName处理了
        case .remoteUserOnSeat:
            // 来自request.userName的上麦邀请被operateUser.userName处理了
        default:
            break
        }
    }
}
```

Android

您可通过调用 `addObserver` 成为 `RTC Room Engine` SDK的观察者，监听麦位相关的回调。
以 `CoGuestObserver` 成为观察者为例：

```
class CoGuestObserver extends TUIRoomObserver {  // 请将其替换为您的业务类,这里只做示例

    CoGuestObserver() {
        TUIRoomEngine.sharedInstance().addObserver(this);
    }
```

```
    // 房间上麦模式发生变化时触发
    @Override
    public void onRoomSeatModeChanged(String roomId, TUIRoomDefine.SeatMode seatMode) {
        // 房间麦位模式改变了，当前模式：seatMode
    }
    // 当麦位列表改变时触发
    @Override
    public void onSeatListChanged(List<TUIRoomDefine.SeatInfo> seatList,
List<TUIRoomDefine.SeatInfo> seatedList,
                                  List<TUIRoomDefine.SeatInfo> leftList) {
        // 麦位列表改变了，麦上最新的用户列表:seatList，新上麦的用户列表:seatedList，新下麦的用户列
表:leftList
    }
    // 当收到其他用户的请求时触发
    @Override
    public void onRequestReceived(TUIRoomDefine.Request request) {
        switch (request.requestAction) {
            case REQUEST_TO_TAKE_SEAT:
            // 收到来自request.userName的上麦请求
            case REQUEST_REMOTE_USER_ON_SEAT:
            // 收到来自request.userName的上麦邀请
            default:
                break;
        }
    }
    // 当其他用户取消请求时触发
    @Override
    public void onRequestCancelled(TUIRoomDefine.Request request, TUIRoomDefine.UserInfo
operateUser) {
        switch (request.requestAction) {
            case REQUEST_TO_TAKE_SEAT:
            // 来自request.userName的上麦请求被取消了
            case REQUEST_REMOTE_USER_ON_SEAT:
            // 来自request.userName的上麦邀请被取消了
            default:
                break;
        }
    }

    // 收到请求被其他 管理员/房主 处理时触发
    @Override
    public void onRequestProcessed(TUIRoomDefine.Request request, TUIRoomDefine.UserInfo
operateUser) {
        switch (request.requestAction) {
            case REQUEST_TO_TAKE_SEAT:
                // 来自request.userName的上麦请求被operateUser.userName处理了
            case REQUEST_REMOTE_USER_ON_SEAT:
                // 来自request.userName的上麦邀请被operateUser.userName处理了
            default:
                break;
        }
    }
```

```
}
```

主播连线

最近更新时间：2025-05-22 16:39:43

本文档主要介绍如何使用 `RTC Room Engine` SDK 实现主播连线功能。

前提条件

在使用 `RTC RoomEngine` SDK前，您需要先调用 登录 SDK，以便后续功能正常使用。

使用指引

> **说明：**
> 使用连线功能时，请确保您已开播。

发起连线请求

您首先需要通过 `getLiveConnectionManager` 接口获取 `TUILiveConnectionManager` 插件。
然后再使用 `TUILiveConnectionManager` 插件的 `requestConnection` 接口实现该功能，传入三个参数：邀请连线的主播所在房间的房间Id、超时时长和扩展信息。
以邀请直播间Id为 live_100002 的主播连线为例：

iOS

```
import RTCRoomEngine

let roomIdList = ["live_100002"]     //请将其替换成您想要邀请连线的主播所在房间的房间ID
let timeout = 30  // 请将这里替换成您申请上麦的超时时间，单位秒，如果设置为 0，SDK 不会做超时检测，也不会触发超时回调
let extensionInfo = ""
let connectionManager = TUIRoomEngine.sharedInstance().getLiveConnectionManager()
connectionManager.requestConnection(roomIdList: roomIdList,
                                    timeout: TimeInterval(timeout),
                                    extensionInfo: extensionInfo) { connectionResults in
    // 发起连线请求成功
} onError: { code, message in
    // 发起连线请求失败
}
```

Android

```
List<String> roomIdList = Collections.singletonList("live_100002");    //请将其替换成您想要邀请连线的主播所在房间的房间ID
int timeout = 30;  // 请将这里替换成您申请上麦的超时时间，单位秒，如果设置为 0，SDK 不会做超时检测，也不会触发超时回调
String  extensionInfo = "";
TUILiveConnectionManager connectionManager =
TUIRoomEngine.sharedInstance().getLiveConnectionManager();
connectionManager.requestConnection(roomIdList, timeout, extensionInfo, new
TUILiveConnectionManager.ConnectionRequestCallback() {
    @Override
```

```
    public void onSuccess(Map<String, TUILiveConnectionManager.ConnectionCode> resultMap) {
        // 发起连线请求成功
    }
    @Override
    public void onError(TUICommonDefine.Error error, String message) {
        // 发起连线请求失败
    }
});
```

若您通过 `addObserver` 接口成为了 `TUILiveConnectionManager` SDK的观察者，则当有人申请和您连线时，您会收到 `onConnectionRequestReceived` 回调，您可通过 `acceptConnection` / `rejectConnection` 接口接受或拒绝该请求。

iOS

```
func onConnectionRequestReceived(inviter: TUIConnectionUser,
                                 inviteeList: [TUIConnectionUser],
                                 extensionInfo: String) {
    // 同意请求
    connectionManager.acceptConnection(inviter.roomId) {
        // 接受连线请求成功
    } onError: { code, message in
        // 接受连线请求失败
    }

    // 拒绝请求
    connectionManager.rejectConnection(inviter.roomId) {
        // 拒绝连线请求成功
    } onError: { code, message in
        // 拒绝连线请求失败
    }
}
```

Android

```
public void onConnectionRequestReceived(TUILiveConnectionManager.ConnectionUser inviter,
                                        List<TUILiveConnectionManager.ConnectionUser>
inviteeList,
                                        String extensionInfo) {
    // 同意请求
    connectionManager.acceptConnection(inviter.roomId, new TUIRoomDefine.ActionCallback() {
        @Override
        public void onSuccess() {
            // 接受连线请求成功
        }
        @Override
        public void onError(TUICommonDefine.Error error, String message) {
            // 接受连线请求失败
        }
    });
```

```
    // 拒绝请求
    connectionManager.rejectConnection(inviter.roomId, new TUIRoomDefine.ActionCallback() {
        @Override
        public void onSuccess() {
            // 拒绝连线请求成功
        }
        @Override
        public void onError(TUICommonDefine.Error error, String message) {
            // 拒绝连线请求失败
        }
    });
}
```

放弃连线请求

您首先需要通过 `getLiveConnectionManager` 接口获取 `TUILiveConnectionManager` 插件。
然后再使用 `TUILiveConnectionManager` 插件的 `cancelConnectionRequest` 接口实现该功能，传入参数：放弃连线邀请的主播所在房间的房间Id。
以放弃和直播间Id为 live_100002 的主播连线请求为例：

iOS

```
let roomIdList = ["live_100002"]     //请将其替换成您想要放弃连线邀请的主播所在房间的房间ID
let connectionManager = TUIRoomEngine.sharedInstance().getLiveConnectionManager()
connectionManager.cancelConnectionRequest(roomIdList: roomIdList) {
    // 放弃连线邀请成功
} onError: { code, message in
    // 放弃连线邀请失败
}
```

Android

```
List<String> roomIdList = Collections.singletonList("live_100002");  //请将其替换成您想要放弃连线邀请的主播所在房间的房间ID

TUILiveConnectionManager connectionManager =
TUIRoomEngine.sharedInstance().getLiveConnectionManager();
connectionManager.cancelConnectionRequest(roomIdList, new TUIRoomDefine.ActionCallback() {
    @Override
    public void onSuccess() {
        // 放弃连线邀请成功
    }
    @Override
    public void onError(TUICommonDefine.Error error, String message) {
        // 放弃连线邀请失败
    }
});
```

中断连线

您首先需要通过 `getLiveConnectionManager` 接口获取 `TUILiveConnectionManager` 插件。

然后再使用 `TUILiveConnectionManager` 插件的 `disconnect` 接口中断当前正在进行的连线。

iOS

```
import RTCRoomEngine

let connectionManager = TUIRoomEngine.sharedInstance().getLiveConnectionManager()
connectionManager.disconnect {
    // 中断连线成功
} onError: { code, message in
    // 中断连线失败
}
```

Android

```
TUILiveConnectionManager connectionManager =
TUIRoomEngine.sharedInstance().getLiveConnectionManager();
connectionManager.disconnect(new TUIRoomDefine.ActionCallback() {
    @Override
    public void onSuccess() {
        // 中断连线成功
    }

    @Override
    public void onError(TUICommonDefine.Error error, String message) {
        // 中断连线失败
    }
});
```

监听回调

iOS

您可通过调用 `addObserver` 成为 `TUILiveConnectionManager` 的观察者，监听连线相关的回调。
以 `CoHostController` 成为观察者为例：

```
import RTCRoomEngine

class CoHostController: NSObject, TUILiveConnectionObserver {
    override init() {
        super.init()
        TUIRoomEngine.sharedInstance().getLiveConnectionManager().addObserver(self)
    }

    deinit {
        TUIRoomEngine.sharedInstance().getLiveConnectionManager().removeObserver(self)
    }

    // 连线用户列表发生变化时触发
    func onConnectionUserListChanged(connectedList: [TUIConnectionUser],
```

```
                                    joinedList: [TUIConnectionUser],
                                    leavedList: [TUIConnectionUser]) {
        // 连线列表发生变化了，最新的连线列表:connectedList，新连线的用户列表:joinedList，新中断连
线的用户列表:leavedList
    }
    // 收到连线邀请时触发
    func onConnectionRequestReceived(inviter: TUIConnectionUser,
                                    inviteeList: [TUIConnectionUser],
                                    extensionInfo: String) {
        // 收到了来自inviter.userName的连线邀请
    }
    // 收到连线邀请被取消时触发
    func onConnectionRequestCancelled(inviter: TUIConnectionUser) {
        // 来自inviter.userName的连线邀请被取消了
    }
    // 收到连线邀请被接受时触发
    func onConnectionRequestAccept(invitee: TUIConnectionUser) {
        // 您对invitee.userName的连线邀请被接受了
    }
    // 收到连线邀请被拒绝时触发
    func onConnectionRequestReject(invitee: TUIConnectionUser) {
        // 您对invitee.userName的连线邀请被拒绝了
    }
    // 收到连线邀请被超时触发
    func onConnectionRequestTimeout(inviter: TUIConnectionUser, invitee:
TUIConnectionUser) {
        // 来自inviter.userName的连线邀请超时了
    }
}
```

Android

您可通过调用 `addObserver` 成为 `TUILiveConnectionManager` 的观察者，监听连线相关的回调。
以 `CoHostObserver` 成为观察者为例：

```
class CoHostObserver extends TUILiveConnectionManager.Observer {
    CoHostObserver() {
        TUIRoomEngine.sharedInstance().getLiveConnectionManager().addObserver(this);
    }


    // 连线用户列表发生变化时触发
    @Override
    public void
onConnectionUserListChanged(List<TUILiveConnectionManager.ConnectionUser> connectedList,

List<TUILiveConnectionManager.ConnectionUser> joinedList,

List<TUILiveConnectionManager.ConnectionUser> leavedList) {
        // 连线列表发生变化了，最新的连线列表:connectedList，新连线的用户列表:joinedList，新中断连
线的用户列表:leavedList
```

```
    }
    // 收到连线邀请时触发
    @Override
    public void onConnectionRequestReceived(TUILiveConnectionManager.ConnectionUser
inviter,

List<TUILiveConnectionManager.ConnectionUser> inviteeList,
                                            String extensionInfo) {
        // 收到了来自inviter.userName的连线邀请
    }
    // 收到连线邀请被取消时触发
    @Override
    public void onConnectionRequestCancelled(TUILiveConnectionManager.ConnectionUser
inviter) {
        // 来自inviter.userName的连线邀请被取消了
    }
    // 收到连线邀请被接受时触发
    @Override
    public void onConnectionRequestAccept(TUILiveConnectionManager.ConnectionUser
invitee) {
        // 您对invitee.userName的连线邀请被接受了
    }
    // 收到连线邀请被拒绝时触发
    @Override
    public void onConnectionRequestReject(TUILiveConnectionManager.ConnectionUser
invitee) {
        // 您对invitee.userName的连线邀请被拒绝了
    }
    // 收到连线邀请被超时触发
    @Override
    public void onConnectionRequestTimeout(TUILiveConnectionManager.ConnectionUser
inviter,
                                           TUILiveConnectionManager.ConnectionUser
invitee) {
        // 来自inviter.userName的连线邀请超时了
    }
}
```

主播 PK

最近更新时间：2025-05-22 16:39:43

本文档主要介绍如何使用 `RTC Room Engine` SDK 实现 主播 PK 功能。

前提条件

在使用 `RTC RoomEngine` SDK前，您需要先调用 登录 SDK，以便后续功能正常使用。

使用指引

> **说明：**
> 使用 PK 功能时，请确保您已开播。

发起 PK 请求

您首先需要通过 `getLiveBattleManager` 接口获取 `TUILiveBattleManager` 插件。
然后再使用 `TUILiveBattleManager` 插件的 `requestBattle` 接口实现该功能，传入三个参数：PK 配置信息、邀请 PK 的主播的用户 ID 和超时时长。
PK 配置信息是一个 TUIBattleConfig 的结构体，在配置时，您一般只需设置其中的 PK 持续时间 `duration` 即可。
以邀请用户 ID 为 100001 的主播 PK 为例：

iOS

```
import RTCRoomEngine

let battleManager = TUIRoomEngine.sharedInstance().getLiveBattleManager()
let config = TUIBattleConfig()
config.duration = 30         // 请将其替换为您的PK持续时间，单位为秒
let userIds = ["100001"]     // 请将其替换为您想与之PK的主播的用户Id
let timeout = 30     // 请将这里替换成您申请上麦的超时时间,单位秒，如果设置为 0，SDK 不会做超时检测，也不会触发超时回调

battleManager.requestBattle(config: config,
                            userIdList: userIds,
                            timeout: TimeInterval(timeout)) { battleInfo, battleResults in
    // 发起 PK 请求成功, battleId:battleInfo.battleId
} onError: { code, message in
    // 发起 PK 请求失败
}
```

Android

```
TUILiveBattleManager battleManager = TUIRoomEngine.sharedInstance().getLiveBattleManager();
TUILiveBattleManager.BattleConfig config = new TUILiveBattleManager.BattleConfig();
config.duration = 30;         // 请将其替换为您的PK持续时间，单位为秒
List<String> userIds = Collections.singletonList("100001");    // 请将其替换为您想与之PK的主播的用户Id
int timeout = 30;     // 请将这里替换成您申请上麦的超时时间,单位秒，如果设置为 0，SDK 不会做超时检测，也不会触发超时回调
```

```
battleManager.requestBattle(config, userIds, timeout, new
TUILiveBattleManager.BattleRequestCallback() {
    @Override
    public void onSuccess(TUILiveBattleManager.BattleInfo battleInfo, Map<String,
TUILiveBattleManager.BattleCode> resultMap) {
        // 发起 PK 请求成功, battleId:battleInfo.battleId
    }
    @Override
    public void onError(TUICommonDefine.Error error, String message) {
        // 发起 PK 请求失败
    }
});
```

若您通过 `addObserver` 接口成为了 `TUILiveBattleManager` 插件的观察者，则当有人申请和您连线时，您会收到 `onBattleRequestReceived` 回调，您可通过 `acceptBattle` / `rejectBattle` 接口接受或拒绝该请求。

iOS

```
func onBattleRequestReceived(battleInfo: TUIBattleInfo,
                             inviter: TUIBattleUser,
                             invitee: TUIBattleUser) {
    // 接受 PK 请求
    let battleManager = TUIRoomEngine.sharedInstance().getLiveBattleManager()
    battleManager.acceptBattle(battleId: battleInfo.battleId) {
        // 接受 PK 请求成功
    } onError: { code, message in
        // 接受 PK 请求失败
    }

    // 拒绝 PK 请求
    battleManager.rejectBattle(battleId: battleInfo.battleId) {
        // 拒绝 PK 请求成功
    } onError: { code, message in
        // 拒绝 PK 请求失败
    }
}
```

Android

```
public void onBattleRequestReceived(TUILiveBattleManager.BattleInfo battleInfo,
TUILiveBattleManager.BattleUser inviter, TUILiveBattleManager.BattleUser invitee) {
    // 接受 PK 请求
    TUILiveBattleManager battleManager =
TUIRoomEngine.sharedInstance().getLiveBattleManager();
    battleManager.acceptBattle(battleInfo.battleId, new TUIRoomDefine.ActionCallback() {
        @Override
        public void onSuccess() {
            // 接受 PK 请求成功
        }
```

```
        @Override
        public void onError(TUICommonDefine.Error error, String message) {
            // 接受 PK 请求失败
        }
    });

    // 拒绝 PK 请求
    battleManager.rejectBattle(battleInfo.battleId, new TUIRoomDefine.ActionCallback() {
        @Override
        public void onSuccess() {
            // 拒绝 PK 请求成功
        }

        @Override
        public void onError(TUICommonDefine.Error error, String message) {
            // 拒绝 PK 请求失败
        }
    });
}
```

放弃 PK 请求

您首先需要通过 `getLiveBattleManager` 接口获取 `TUILiveBattleManager` 插件。
然后再使用 `TUILiveBattleManager` 插件的 `cancelBattleRequest` 接口实现该功能，传入两个参数：发起 PK 请求成功后 battleInfo 的 battleId 和 放弃 PK 邀请的主播用户 ID。
以放弃和用户Id为 100001 的主播 PK 请求为例：

iOS

```
let battleManager = TUIRoomEngine.sharedInstance().getLiveBattleManager()
let battleId = ""          // 请将其替换为您调用requestBattle请求PK成功后battleInfo中的battleId值
let userIds = ["100001"]   // 请将其替换为您邀请PK的主播的用户Id
battleManager.cancelBattleRequest(battleId: battleId, userIdList: userIds) {
    // 放弃 PK 请求成功
} onError: { code, message in
    // 放弃 PK 请求失败
}
```

Android

```
TUILiveBattleManager battleManager = TUIRoomEngine.sharedInstance().getLiveBattleManager();
String battleId = "";          // 请将其替换为您调用requestBattle请求PK成功后battleInfo中的battleId
值
List<String> userIds = Collections.singletonList("100001"); // 请将其替换为您邀请PK的主播的用户Id
battleManager.cancelBattleRequest(battleId, userIds, new TUIRoomDefine.ActionCallback() {
    @Override
    public void onSuccess() {
        // 放弃 PK 请求成功
    }
    @Override
    public void onError(TUICommonDefine.Error error, String message) {
```

```
        // 放弃 PK 请求失败
    }
});
```

退出 PK

您首先需要通过 `getLiveBattleManager` 接口获取 `TUILiveBattleManager` 插件。
然后再使用 `TUILiveBattleManager` 插件的 `exitBattle` 接口，传入参数：发起 PK 请求成功后 battleInfo 的 battleId，实现退出当前正在进行的 PK。

iOS

```
import RTCRoomEngine

let battleManager = TUIRoomEngine.sharedInstance().getLiveBattleManager()
let battleId = ""          // 请将其替换为您调用requestBattle请求PK成功后battleInfo中的battleId值
battleManager.exitBattle(battleId: battleId) {
    // 退出 PK 成功
} onError: { code, message in
    // 退出 PK 失败
}
```

Android

```
TUILiveBattleManager battleManager = TUIRoomEngine.sharedInstance().getLiveBattleManager();
String battleId = "";          // 请将其替换为您调用requestBattle请求PK成功后battleInfo中的battleId
值
battleManager.exitBattle(battleId, new TUIRoomDefine.ActionCallback() {
    @Override
    public void onSuccess() {
        // 退出 PK 成功
    }
    @Override
    public void onError(TUICommonDefine.Error error, String message) {
        // 退出 PK 失败
    }
});
```

监听回调

iOS

您可通过调用 `addObserver` 成为 `TUILiveBattleManager` 的观察者，监听 PK 相关的回调。
以 `AnchorBattleController` 成为观察者为例：

```
import RTCRoomEngine

class AnchorBattleController: NSObject, TUILiveBattleObserver {
    override init() {
        super.init()
```

```
        TUIRoomEngine.sharedInstance().getLiveBattleManager().addObserver(self)
    }

    deinit {
        TUIRoomEngine.sharedInstance().getLiveBattleManager().removeObserver(self)
    }

    // PK 开始时触发
    func onBattleStarted(battleInfo: TUIBattleInfo) {
        // PK 开始了, battleId:battleInfo.battleId
    }
    // PK 结束时触发
    func onBattleEnded(battleInfo: TUIBattleInfo, reason: TUIBattleStoppedReason) {
        // PK 结束了, battleId:battleInfo.battleId
    }
    // 有新主播加入 PK 时触发
    func onUserJoinBattle(battleId: String, battleUser: TUIBattleUser) {
        // battleUser.userName加入了 PK
    }
    // 正在 PK 的主播离开了 PK 时触发
    func onUserExitBattle(battleId: String, battleUser: TUIBattleUser) {
        // battleUser.userName离开了 PK
    }
    // PK 得分变化时触发
    func onBattleScoreChanged(battleId: String, battleUserList: [TUIBattleUser]) {
        // PK 得分改变了，第一个用户的比分为: battleUserList.first?.score
    }
    // 收到 PK 邀请时触发
    func onBattleRequestReceived(battleInfo: TUIBattleInfo, inviter: TUIBattleUser,
invitee: TUIBattleUser) {
        // 收到了来自inviter.userName的 PK 邀请
    }
    // 收到 PK 邀请被取消时触发
    func onBattleRequestCancelled(battleInfo: TUIBattleInfo, inviter: TUIBattleUser,
invitee: TUIBattleUser) {
        // 来自inviter.userName的 PK 邀请被取消了
    }
    // 收到 PK 邀请超时触发
    func onBattleRequestTimeout(battleInfo: TUIBattleInfo, inviter: TUIBattleUser,
invitee: TUIBattleUser) {
        // 来自inviter.userName的 PK 邀请超时了
    }
    // 发出的 PK 邀请被接受时触发
    func onBattleRequestAccept(battleInfo: TUIBattleInfo, inviter: TUIBattleUser,
invitee: TUIBattleUser) {
        // 对invitee.userName的 PK 邀请被接受了
    }
    // 发出的 PK 邀请被拒绝时触发
    func onBattleRequestReject(battleInfo: TUIBattleInfo, inviter: TUIBattleUser,
invitee: TUIBattleUser) {
        // 对invitee.userName的 PK 邀请被拒绝了
    }
}
```

Android

您可通过调用 `addObserver` 成为 `TUILiveBattleManager` 的观察者，监听 PK 相关的回调。
以 `AnchorBattleObserver` 成为观察者为例：

```
class AnchorBattleObserver extends TUILiveBattleManager.Observer {
    AnchorBattleObserver() {

        TUIRoomEngine.sharedInstance().getLiveBattleManager().addObserver(this);
    }

    // PK 开始时触发
    @Override
    public void onBattleStarted(TUILiveBattleManager.BattleInfo battleInfo)  {
        // PK 开始了, battleId:battleInfo.battleId
    }
    // PK 结束时触发
    @Override
    public void onBattleEnded(TUILiveBattleManager.BattleInfo battleInfo,
TUILiveBattleManager.BattleStoppedReason reason) {
        // PK 结束了, battleId:battleInfo.battleId
    }
    // 有新主播加入 PK 时触发
    @Override
    public void onUserJoinBattle(String battleId, TUILiveBattleManager.BattleUser
battleUser)  {
        // battleUser.userName加入了 PK
    }
    // 正在 PK 的主播离开了 PK 时触发
    @Override
    public void onUserExitBattle(String battleId, TUILiveBattleManager.BattleUser
battleUser) {
        // battleUser.userName离开了 PK
    }
    // PK 得分变化时触发
    @Override
    public void onBattleScoreChanged(String battleId,
List<TUILiveBattleManager.BattleUser> battleUserList) {
        // PK 得分改变了，第一个用户的比分为: battleUserList.first?.score
    }
    // 收到 PK 邀请时触发
    @Override
    public void onBattleRequestReceived(TUILiveBattleManager.BattleInfo battleInfo,
TUILiveBattleManager.BattleUser inviter, TUILiveBattleManager.BattleUser invitee) {
        // 收到了来自inviter.userName的 PK 邀请
    }
    // 收到 PK 邀请被取消时触发
    @Override
    public void onBattleRequestCancelled(TUILiveBattleManager.BattleInfo battleInfo,
TUILiveBattleManager.BattleUser inviter, TUILiveBattleManager.BattleUser invitee) {
        // 来自inviter.userName的 PK 邀请被取消了
```

```
    }
    // 收到 PK 邀请超时触发
    @Override
    public void onBattleRequestTimeout(TUILiveBattleManager.BattleInfo battleInfo,
TUILiveBattleManager.BattleUser inviter, TUILiveBattleManager.BattleUser invitee) {
        // 来自inviter.userName的 PK 邀请超时了
    }
    // 发出的 PK 邀请被接受时触发
    @Override
    public void onBattleRequestAccept(TUILiveBattleManager.BattleInfo battleInfo,
TUILiveBattleManager.BattleUser inviter, TUILiveBattleManager.BattleUser invitee) {
        // 对invitee.userName的 PK 邀请被接受了
    }
    // 发出的 PK 邀请被拒绝时触发
    @Override
    public void onBattleRequestReject(TUILiveBattleManager.BattleInfo battleInfo,
TUILiveBattleManager.BattleUser inviter, TUILiveBattleManager.BattleUser invitee) {
        // 对invitee.userName的 PK 邀请被拒绝了
    }
}
```

房间列表

最近更新时间：2025-05-22 16:39:43

本文档主要介绍如何使用 `RTC Room Engine` SDK 实现房间列表功能。
您可利用 `RTC Room Engine` SDK提供 `TUILiveListManager` 插件实现房间列表功能。
在使用 TUILiveListManager 插件时，您只需要关注**如何让自己的直播间在房间列表中可见**和**如何获取直播间列表即可**。

前提条件

在使用 `RTC RoomEngine` SDK 前，您需要先调用 登录 SDK，以便后续功能正常使用。

使用指引

如何让自己的直播间在房间列表可见

您首先需要通过 `getExtension` 接口获取 `TUILiveListManager` 插件。
然后再使用 `TUILiveListManager` 插件的 `setLiveInfo` 接口实现该功能，传入两个参数：直播间信息和修改标识。

iOS

```
import RTCRoomEngine

let liveListManager = TUIRoomEngine.sharedInstance().getExtension(extensionType:
.liveListManager) as! TUILiveListManager

let liveInfo = TUILiveInfo()
liveInfo.roomInfo.roomId = "live_100001"  //请将其替换成您自己的直播间房间Id
liveInfo.isPublicVisible = true          //让直播间可在直播列表中展示
liveListManager.setLiveInfo(liveInfo, modifyFlag: [.publish]) {
    // 设置直播间信息成功
} onError: { code, message in
    // 设置直播间信息失败
}
```

Android

```
TUILiveListManager liveListManager = (TUILiveListManager)
TUIRoomEngine.sharedInstance().getExtension(TUICommonDefine.ExtensionType.LIVE_LIST_MANAGER)
;

TUILiveListManager.LiveInfo liveInfo = new TUILiveListManager.LiveInfo();
liveInfo.roomInfo.roomId = "live_100001";  //请将其替换成您自己的直播间房间Id
liveInfo.isPublicVisible = true;
List<TUILiveListManager.LiveModifyFlag> flagList = new ArrayList<>();
flagList.add(TUILiveListManager.LiveModifyFlag.PUBLISH);
liveListManager.setLiveInfo(liveInfo, flagList, new TUIRoomDefine.ActionCallback() {
    @Override
    public void onSuccess() {
        // 设置直播间信息成功
    }
    @Override
```

```
    public void onError(TUICommonDefine.Error error, String message) {
        // 设置直播间信息失败
    }
});
```

> **说明：**
> 在设置直播间可见前请确保您的直播间已处于**开播**状态。

如何获取直播间列表

您首先需要通过 `getExtension` 接口获取 `TUILiveListManager` 插件。
然后再使用 `TUILiveListManager` 插件的 `fetchLiveList` 接口实现该功能，传入两个参数：字符串类型的列表下标和单次拉取直播间个数。

iOS

```
import RTCRoomEngine

let liveListManager = TUIRoomEngine.sharedInstance().getExtension(extensionType:
.liveListManager) as! TUILiveListManager

let cursor = ""                // 首次拉取时下标填空字符串即可，后续拉取根据接口返回的cursor设值
let singleFetchRoomLimit = 50  // 请将其替换成您单次拉取直播间个数，个数上限为50
liveListManager.fetchLiveList(cursor: "", count: singleFetchRoomLimit) { cursor, liveList in
    // 获取直播间列表成功
} onError: { code, message in
    // 获取直播间列表失败
}
```

Android

```
TUILiveListManager liveListManager = (TUILiveListManager)
TUIRoomEngine.sharedInstance().getExtension(TUICommonDefine.ExtensionType.LIVE_LIST_MANAGER)
;

String cursor = ""; // 首次拉取时下标填空字符串即可，后续拉取根据接口返回的cursor设值
int singleFetchRoomLimit = 50;  // 请将其替换成您单次拉取直播间个数，个数上限为50
liveListManager.fetchLiveList("", singleFetchRoomLimit, new
TUILiveListManager.LiveInfoListCallback() {
    @Override
    public void onSuccess(TUILiveListManager.LiveInfoListResult result) {
        // 获取直播间列表成功
    }
    @Override
    public void onError(TUICommonDefine.Error error, String message) {
        // 获取直播间列表失败
    }
});
```

高级功能
自定义房间信息

最近更新时间：2025-05-22 16:39:43

本文档主要介绍如何使用 `RTC Room Engine` SDK 实现自定义房间信息。

前提条件

在使用 `RTC RoomEngine` SDK 前，您需要先调用 登录 SDK，以便后续功能正常使用。
只有当您是主播时，设置的直播间信息才会生效。您可以参见 语聊开播与收听 或 视频开播 来完成直播间的创建。

使用指引

> **说明：**
> 使用自定义房间信息功能时，请确保您是房主或者管理员。房间类型为 live。

设置房间信息：

iOS

您需要准备好需要设置的参数 `TUILiveInfo` ，接下来进行详细介绍：
参数：TUILiveInfo
`TUILiveInfo` 由很多的字段构成，但通常您只需要关心如下几个字段的填写：

参数名称	类型	描述
activityStatus	Int	直播间活跃状态：用户自定义标记。
backgroundUrl	String	直播间背景，最大支持 200 个字节。
categoryList	List<Int>	直播间分类标签，单个房间最大支持3个标记。
coverUrl	String	直播间封面，最大支持 200 个字节。
isPublicVisible	Bool	直播间是否公开，设置为 true 后即可在 房间列表 中被显示。

在准备好 TUILiveInfo 之后，便可以通过调用 `setLiveInfo` 接口设置直播间信息。

Android

您需要准备好需要设置的参数 `LiveInfo` ，接下来进行详细介绍：
参数：LiveInfo
`LiveInfo` 由很多的字段构成，但通常您只需要关心如下几个字段的填写：

参数名称	类型	描述
activityStatus	Int	直播间活跃状态: 用户自定义标记。
backgroundUrl	String	直播间背景，最大支持 200 个字节。
categoryList	List<Int>	直播间分类标签，单个房间最大支持3个标记。

coverUrl	String	直播间封面，最大支持 200 个字节。
isPublicVisible	Bool	直播间是否公开，设置为 true 后即可在 房间列表 中被显示。

在准备好 TUILiveInfo 之后，便可以通过调用 `setLiveInfo` 接口设置直播间信息。

示例：

iOS

```
import RTCRoomEngine

let roomEngine = TUIRoomEngine.sharedInstance()
let liveListManager = roomEngine.getExtension(extensionType: .liveListManager) as?
TUILiveListManager

let liveInfo = TUILiveInfo()
liveInfo.backgroundUrl = "backgroundUrl"    // 替换为您需要的直播间背景图片
liveInfo.coverUrl = "coverUrl"              // 替换为您需要的直播间封面图片
liveInfo.isPublicVisible = true             // 直播间公开
liveInfo.categoryList = [1, 2]              // 可替换为您业务中的直播间分类

let modifyFlag: TUILiveModifyFlag = [.backgroundUrl, .coverUrl, .publish, .category]  // 这里
为您所修改的类别

liveListManager?.setLiveInfo(liveInfo, modifyFlag: modifyFlag) {
    // 设置直播间信息成功
} onError: { code, message in
    // 设置直播间信息失败
}
```

Android

```
TUIRoomEngine roomEngine = TUIRoomEngine.sharedInstance();
TUILiveListManager liveListManager = (TUILiveListManager)
roomEngine.getExtension(TUICommonDefine.ExtensionType.LIVE_LIST_MANAGER);

TUILiveListManager.LiveInfo liveInfo = new TUILiveListManager.LiveInfo();
liveInfo.backgroundUrl = "backgroundUrl";                       // 替换为您需要的直播间背景图片
liveInfo.coverUrl = "coverUrl";                                 // 替换为您需要的直播间封面图片
liveInfo.isPublicVisible = true;                                // 直播间公开
liveInfo.categoryList =new ArrayList<>(Arrays.asList(1, 2));    // 可替换为您业务中的直播间分类


List<TUILiveListManager.LiveModifyFlag> modifyFlag  =  new ArrayList<>();
// 下面为您所修改的类别
modifyFlag.add(TUILiveListManager.LiveModifyFlag.BACKGROUND_URL);
modifyFlag.add(TUILiveListManager.LiveModifyFlag.COVER_URL);
modifyFlag.add(TUILiveListManager.LiveModifyFlag.PUBLISH);
modifyFlag.add(TUILiveListManager.LiveModifyFlag.CATEGORY);
liveListManager.setLiveInfo(liveInfo, modifyFlag, new TUIRoomDefine.ActionCallback() {
```

版权所有：腾讯云计算（北京）有限责任公司

```
    @Override
    public void onSuccess() {
        // 设置直播间信息成功
    }
    @Override
    public void onError(TUICommonDefine.Error error, String message) {
        // 设置直播间信息失败
    }
});
```

修改房间名称：

使用 `updateRoomNameByAdmin` API修改房间名称。SDK会通过 `onRoomNameChanged` 回调通知房间内用户。
此函数仅有房主或管理员可以调用。
示例：

iOS

```
import RTCRoomEngine

let roomEngine = TUIRoomEngine.sharedInstance()
let name = "New Name" //替换为您的真实用户名
roomEngine.updateRoomNameByAdmin(roomName: newName) {
            //修改房间名成功
            } onError: { code, message in
            //修改房间名失败
            }
```

Android

```
TUIRoomEngine roomEngine = TUIRoomEngine.sharedInstance();
String newName = ""; //替换为你需要的newName
roomEngine.updateRoomNameByAdmin(newName ,new TUIRoomDefine.ActionCallback() {
    @Override
    public void onSuccess() {
        //修改房间名成功
    }

    @Override
    public void onError(TUICommonDefine.Error error, String message) {
        //修改房间名成功
    }
});
```

自定义房间信息：

使用 `setRoomMetadataByAdmin` API设置房间自定义信息，已有该属性则更新其 value 值，没有则添加该属性。
此函数仅有房主或管理员可以调用，且此函数仅支持直播房间类型为 `Live`
示例：

iOS

```
import RTCRoomEngine

let roomEngine = TUIRoomEngine.sharedInstance()
roomEngine.setRoomMetadataByAdmin([key:value], onSuccess: {
      //设置自定义数据成功
      }, onError: {code ,message in
      //设置自定义数据失败
      })
```

Android

```
TUIRoomEngine roomEngine = TUIRoomEngine.sharedInstance();

// 准备要设置或更新的元数据键值对
HashMap<String, String> metadataToUpdate = new HashMap<>(); 请填入您想要的键值对
metadataToUpdate.put("is_recording", "true");       // 设置是否正在录制，此处仅为举例

roomEngine.setRoomMetadataByAdmin(metadataToUpdate, new TUIRoomDefine.ActionCallback() {
    @Override
    public void onSuccess() {
        // 设置成功的回调
    }

    @Override
    public void onError(int errorCode, String errorMessage) {
        // 设置失败的回调
    }
});
```

获取房间自定义信息：

当设置自定义信息完成之后，您可以调用 `getRoomMetadata` 接口来获取当前直播间的全部或指定的信息。
此函数仅有房主或管理员可以调用，且此函数仅支持直播房间类型为 `Live`
示例：

iOS

```
import RTCRoomEngine

let roomEngine = TUIRoomEngine.sharedInstance()
let keys: [String] = [""]//参数传入房间自定义信息 key 列表，如果 keys 传空则获取所有自定义信息

roomEngine.getRoomMetadata(array) { metadata in
        //获取自定义信息成功
        } onError: { code, message in
        //获取自定义信息失败
        }
```

Android

```
TUIRoomEngine roomEngine = TUIRoomEngine.sharedInstance();

List<String> keysToQuery = new ArrayList<>();//替换为您需要查询的列表
roomEngine.getRoomMetadata(keysToQuery, new TUIRoomDefine.GetRoomMetadataCallback() {
    @Override
    public void onSuccess(Map<String, String> metadata) {
        // 成功回调
    }

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

视频设置

最近更新时间：2025-05-22 16:39:43

本文档主要介绍如何使用 `RTC Room Engine` SDK 实现视频设置相关功能。

前提条件

在使用 `RTC Room Engine` SDK 提供的视频设置相关功能之前，您需要先 登录 SDK。

使用指引

开启/关闭本地摄像头

iOS

您可以分别通过调用 `openLocalCamera` 和 `closeLocalCamera` 两个接口，来开启或关闭您本地的摄像头。`openLocalCamera` 接口需要传入 选择前后置摄像头 `isFront` 和 视频质量 `quality` 两个参数。`isFront` 为布尔值，true 为打开前置摄像头，false 为打开后置摄像头。`quality` 是 `TUIVideoQuality` 类型的枚举。

枚举值类型	含义
quality360P	低清360P。
quality540P	标清540P。
quality720P	高清720P。
quality1080P	超清1080P。

下面以打开前置摄像头并且视频质量为 `quality1080P` 为例，提供了开启本地麦克风和关闭本地摄像头的示例代码。

Android

您可以分别通过调用 `openLocalCamera` 和 `closeLocalCamera` 两个接口，来开启或关闭您本地的摄像头。`openLocalCamera` 接口需要传入 选择前后置摄像头 `isFront` 和 视频质量 `quality` 两个参数。`isFront` 为布尔值，true 为打开前置摄像头，false 为打开后置摄像头。`quality` 是 `VideoQuality` 类型的枚举。

枚举值类型	含义
Q_360P	低清360P。
Q_540P	标清540P。
Q_720P	高清720P。
Q_1080P	超清1080P。

下面以打开前置摄像头并且视频质量为 `Q_1080P` 为例，提供了开启本地麦克风和关闭本地摄像头的示例代码。

iOS

```
import RTCRoomEngine

let roomEngine = TUIRoomEngine.sharedInstance()

// 开启本地摄像头
let isFrontCamera = true
let videoQuality: TUIVideoQuality = .quality1080P
roomEngine.openLocalCamera(isFront: isFrontCamera, quality: videoQuality) {
    // 开启成功
} onError: { code, message in
    // 开启失败
}

// 关闭本地摄像头
roomEngine.closeLocalCamera()
```

Android

```
TUIRoomEngine roomEngine = TUIRoomEngine.sharedInstance();

// 开启本地摄像头
boolean isFrontCamera = true;
TUIRoomDefine.VideoQuality videoQuality = TUIRoomDefine.VideoQuality.Q_1080P;
roomEngine.openLocalCamera(isFrontCamera, videoQuality, new TUIRoomDefine.ActionCallback() {
    @Override
    public void onSuccess() {
        // 开启成功
    }
    @Override
    public void onError(TUICommonDefine.Error error, String message) {
        // 开启失败
    }
});

// 关闭本地摄像头
roomEngine.closeLocalCamera();
```

设置本地视频画面镜像

以下是一个开启本地视频画面镜像的示例，您可以通过以下代码开启/关闭本地视频画面的镜像。

iOS

```
import RTCRoomEngine
import TXLiteAVSDK_Professional

let trtcCloud = TUIRoomEngine.sharedInstance().getTRTCCloud()

let params = TRTCRenderParams()
params.mirrorType = .enable            // 关闭镜像时，将此处参数设置为 .disable 即可
trtcCloud.setLocalRenderParams(params)
```

```
trtcCloud.setVideoEncoderMirror(true)
```

Android

```
TRTCCloud trtcCloud = TUIRoomEngine.sharedInstance().getTRTCCloud();
TRTCCloudDef.TRTCRenderParams params = new TRTCCloudDef.TRTCRenderParams();
params.mirrorType = TRTC_VIDEO_MIRROR_TYPE_ENABLE; // 关闭镜像时，将此处参数设置为
TRTC_VIDEO_MIRROR_TYPE_DISABLE 即可
trtcCloud.setLocalRenderParams(params);
trtcCloud.setVideoEncoderMirror(true);
```

切换摄像头

调用 `switchCamera` 接口切换摄像头时，需要传入一个 Bool 参数 `frontCamera`，传入 true 时切换为前置摄像头，传入 false 时切换为后置摄像头。以下是切换为前置摄像头的示例代码：

iOS

```
import RTCRoomEngine

TUIRoomEngine.sharedInstance().getMediaDeviceManager().switchCamera(true)
```

Android

```
TUIRoomEngine.sharedInstance().getMediaDeviceManager().switchCamera(true);
```

更新本地视频编码质量

iOS

更新本地视频编码质量时，需要传入的参数类型 `TUIVideoQuality` 与上文中提到的相同。下面以默认模式为例，调用 `updateVideoQuality` 接口更新本地视频的编码质量：

```
import RTCRoomEngine

let videoQuality: TUIVideoQuality = .quality1080P
TUIRoomEngine.sharedInstance().updateVideoQuality(videoQuality)
```

Android

更新本地视频编码质量时，需要传入的参数类型 `VideoQuality` 与上文中提到的相同。下面以默认模式为例，调用 `updateVideoQuality` 接口更新本地视频的编码质量：

```
TUIRoomDefine.VideoQuality videoQuality = TUIRoomDefine.VideoQuality.Q_1080P;
```

```
TUIRoomEngine.sharedInstance().updateVideoQuality(videoQuality);
```

设置视频编码器的编码参数

iOS

设置视频编码器的编码参数时，需要传入的参数类型 `TUIRoomVideoEncoderParams` 。下面以默认模式为例，调用 `updateVideoQualityEx` 接口更新本地视频的编码质量：
`TUIRoomVideoEncoderParams` 参数介绍：

参数项	参数名称
分辨率	videoResolution
帧率	fps
分辨率模式	resolutionMode
最高码率	bitrate

```
import RTCRoomEngine

let roomEngine = TUIRoomEngine.sharedInstance()

let params = TUIRoomVideoEncoderParams()
params.fps = 10 //此处换为您真实需要的值
params.resolutionMode = .portrait //竖屏分辨率
params.bitrate = .1600 //此处换为您真实需要的值
params.videoResolution = .quality720P //此处换位您需要的值

roomEngine.updateVideoQualityEx(streamType: .screenStream, params: params)
```

Android

设置视频编码器的编码参数时，需要传入的参数类型 `TUIRoomDefine.RoomVideoEncoderParams` 。下面以默认模式为例，调用 `updateVideoQualityEx` 接口更新本地视频的编码质量：
`TUIRoomDefine.RoomVideoEncoderParams` 参数介绍：

参数项	参数名称
分辨率	videoResolution
帧率	fps
分辨率模式	resolutionMode
最高码率	bitrate

```
TUIRoomEngine roomEngine = TUIRoomEngine.sharedInstance();

TUIRoomDefine.RoomVideoEncoderParams params = new
TUIRoomDefine.RoomVideoEncoderParams();
params.videoResolution = TUIRoomDefine.VideoQuality.Q_720P; //此处换为您真实需要的值
params.bitrate = 1600; //此处换为您真实需要的值
params.fps = 10; //此处换为您真实需要的值
params.resolutionMode = TUIRoomDefine.ResolutionMode.PORTRAIT; //竖屏分辨率

roomEngine.updateVideoQualityEx(TUIRoomDefine.RoomVideoEncoderParams params)
```

开始/停止推送本地视频

当您在直播间内时，您可能需要开始/停止推送您的本地视频，您可以通过调用以下接口来实现：

iOS

```
import RTCRoomEngine

let roomEngine = TUIRoomEngine.sharedInstance()

// 开始推送本地视频（默认开启）
roomEngine.startPushLocalVideo()

// 停止推送本地视频
roomEngine.stopPushLocalVideo()
```

Android

```
TUIRoomEngine roomEngine = TUIRoomEngine.sharedInstance();

// 开始推送本地视频（默认开启）
roomEngine.startPushLocalVideo();

// 停止推送本地视频
roomEngine.stopPushLocalVideo();
```

说明：
在房间内，若您已打开您的摄像头，调用如上接口开始/停止推送本地视频后，SDK 会通过 `TUIRoomObserver` 中的 onUserVideoStateChanged 回调通知房间内用户。

音频设置

最近更新时间：2025-05-22 16:39:43

本文档主要介绍如何使用 `RTC Room Engine` SDK 实现音频设置相关功能。

前提条件

在使用 `RTC Room Engine` SDK 提供的音频设置相关功能之前，您需要先完成 登录 SDK，并确保自己已在一个直播间当中。

使用指引

开启/关闭本地麦克风

iOS

您可以分别通过调用 `openLocalMicrophone` 和 `closeLocalMicrophone` 两个接口，来开启或关闭您本地的麦克风。
在调用 `openLocalMicrophone` 开启麦克风时，需要传入一个 `TUIAudioQuality` 类型的参数 `quality` 以设置音频编码质量，`TUIAudioQuality` 包含以下类型，您可以根据您的业务需求进行选择：

枚举值类型	含义
speech	人声模式。单声道；音频裸码率：18kbps；适合语音通话为主的场景。
default	默认模式。单声道；音频裸码率：50kbps；SDK 默认的音频质量，若无特殊需求推荐选择默认模式。
music	音乐模式。双声道 + 全频带；音频裸码率：128kbps；适合需要高保真传输音乐的场景，例如在线 K 歌、音乐直播等。

Android

您可以分别通过调用 `openLocalMicrophone` 和 `closeLocalMicrophone` 两个接口，来开启或关闭您本地的麦克风。
在调用 `openLocalMicrophone` 开启麦克风时，需要传入一个 `AudioQuality` 类型的参数 `quality` 以设置音频编码质量，`AudioQuality` 包含以下类型，您可以根据您的业务需求进行选择：

枚举值类型	含义
SPEECH	人声模式。单声道；音频裸码率：18kbps；适合语音通话为主的场景。
DEFAULT	默认模式。单声道；音频裸码率：50kbps；SDK 默认的音频质量，若无特殊需求推荐选择默认模式。
MUSIC	音乐模式。双声道 + 全频带；音频裸码率：128kbps；适合需要高保真传输音乐的场景，例如在线 K 歌、音乐直播等。

下面以默认模式为例，提供了开启本地麦克风和关闭本地麦克风的示例代码。

iOS

```
import RTCRoomEngine
```

```
let roomEngine = TUIRoomEngine.sharedInstance()

// 开启本地麦克风
roomEngine.openLocalMicrophone(.default) {
    // 开启麦克风成功
} onError: { code, message in
    // 开启麦克风失败
}

// 关闭本地麦克风
roomEngine.closeLocalMicrophone()
```

Android

```
TUIRoomEngine roomEngine = TUIRoomEngine.sharedInstance();
// 开启本地麦克风
roomEngine.openLocalMicrophone(TUIRoomDefine.AudioQuality.DEFAULT, new
TUIRoomDefine.ActionCallback() {
    @Override
    public void onSuccess() {
        // 开启麦克风成功
    }
    @Override
    public void onError(TUICommonDefine.Error error, String message) {
        // 开启麦克风失败
    }
});
// 关闭本地麦克风
roomEngine.closeLocalMicrophone();
```

更新本地音频编码质量

iOS

更新本地音频编码质量时，需要传入的参数类型 `TUIAudioQuality` 与上文中提到的相同。下面以默认模式为例，调用 `updateAudioQuality` 接口更新本地音频的编码质量：

```
import RTCRoomEngine

let audioQuality: TUIAudioQuality = .default
TUIRoomEngine.sharedInstance().updateAudioQuality(audioQuality)
```

Android

更新本地音频编码质量时，需要传入的参数类型 `AudioQuality` 与上文中提到的相同。下面以默认模式为例，调用 `updateAudioQuality` 接口更新本地音频的编码质量：

```
TUIRoomDefine.AudioQuality audioQuality = TUIRoomDefine.AudioQuality.DEFAULT;
```

```
TUIRoomEngine.sharedInstance().updateAudioQuality(audioQuality);
```

暂停/恢复发布本地音频流

当您在直播间内时，您可能需要暂停/恢复发布您的本地音频流，您可以通过调用以下接口来实现：

iOS

```
import RTCRoomEngine

let roomEngine = TUIRoomEngine.sharedInstance()

// 暂停发布本地音频流
roomEngine.muteLocalAudio()

// 恢复发布本地音频流
roomEngine.unmuteLocalAudio() {
    // 恢复发布成功
} onError: { code, message in
    // 恢复发布失败
}
```

Android

```
TUIRoomEngine roomEngine = TUIRoomEngine.sharedInstance();

// 暂停发布本地音频流
roomEngine.muteLocalAudio();

// 恢复发布本地音频流
roomEngine.unmuteLocalAudio(new TUIRoomDefine.ActionCallback() {
    @Override
    public void onSuccess() {
        // 恢复发布成功
    }

    @Override
    public void onError(TUICommonDefine.Error error, String message) {
        // 恢复发布失败
    }
});
```

说明：
在房间内，若您已打开您的麦克风，调用如上接口暂停/恢复发布本地的音频流后，SDK 会通过 `TUIRoomObserver` 中的 onUserAudioStateChanged 回调通知房间内用户。

背景音乐

最近更新时间：2025-05-22 16:39:43

本文档主要介绍如何使用 `RTC Room Engine` SDK 实现背景音乐相关功能。

前提条件

在使用 `RTC Room Engine` SDK 提供的背景音乐设置相关功能之前，您需要先 登录 SDK。

使用指引

开始/停止播放背景音乐

iOS

您可以分别通过调用 `startPlayMusic` 和 `stopPlayMusic` 两个接口，来开始或停止播放背景音乐。
在调用 `startPlayMusic` 开始播放背景音乐时，需要传入一个 `TXAudioMusicParam` 类型的参数来设置播放控制信息。
`TXAudioMusicParam` 包含以下信息，您可以分别对其进行设置：

枚举类型	描述
id	**字段含义**：音乐 ID。 **特殊说明**：SDK 允许播放多路音乐，因此需要使用 ID 进行标记，用于控制音乐的开始、停止、音量等。
endTimeMS	**字段含义**：音乐结束播放时间点，单位毫秒，0表示播放至文件结尾。
isShortFile	**字段含义**：播放的是否为短音乐文件。 **推荐取值**： • YES：需要重复播放的短音乐文件。 • NO：正常的音乐文件。默认值：NO。
loopCount	**字段含义**：音乐循环播放的次数。 **推荐取值**：取值范围为0 - 任意正整数，默认值：0。0 表示播放音乐一次；1 表示播放音乐两次；以此类推。
path	**字段含义**：音效文件的完整路径或 URL 地址。支持的音频格式包括 MP3、AAC、M4A、WAV。
publish	**字段含义**：是否将音乐传到远端。 **推荐取值**： • YES：音乐在本地播放的同时，远端用户也能听到该音乐。 • NO：主播只能在本地听到该音乐，远端观众听不到。默认值：NO。
startTimeMS	**字段含义**：音乐开始播放时间点，单位：毫秒。

Android

您可以分别通过调用 `startPlayMusic` 和 `stopPlayMusic` 两个接口，来开始或停止播放背景音乐。
在调用 `startPlayMusic` 开始播放背景音乐时，需要传入一个 `AudioMusicParam` 类型的参数来设置播放控制信息。
`AudioMusicParam` 包含以下信息，您可以分别对其进行设置：

枚举类型	描述

id	**字段含义**：音乐 ID。 **特殊说明**：SDK 允许播放多路音乐，因此需要使用 ID 进行标记，用于控制音乐的开始、停止、音量等。
endTimeMS	**字段含义**：音乐结束播放时间点，单位毫秒，0表示播放至文件结尾。
isShortFile	**字段含义**：播放的是否为短音乐文件。 **推荐取值**： • YES：需要重复播放的短音乐文件。 • NO：正常的音乐文件。默认值：NO。
loopCount	**字段含义**：音乐循环播放的次数。 **推荐取值**：取值范围为0 – 任意正整数，默认值：0。0 表示播放音乐一次；1 表示播放音乐两次；以此类推。
path	**字段含义**：音效文件的完整路径或 URL 地址。支持的音频格式包括 MP3、AAC、M4A、WAV。
publish	**字段含义**：是否将音乐传到远端。 **推荐取值**： • YES：音乐在本地播放的同时，远端用户也能听到该音乐。 • NO：主播只能在本地听到该音乐，远端观众听不到。默认值：NO。
startTimeMS	**字段含义**：音乐开始播放时间点，单位：毫秒。

以下是一个简单开始/停止播放背景音乐的示例：

iOS

```
import RTCRoomEngine
import TXLiteAVSDK_Professional

let audioEffectManager =
TUIRoomEngine.sharedInstance().getTRTCCloud().getAudioEffectManager()

// 开始播放背景音乐
let musicParam = TXAudioMusicParam()
musicParam.id = 0                    // 替换为您自己的音乐ID
musicParam.path = "path"             // 替换为音乐文件的完整路径或 URL 地址
musicParam.publish = true            // 将音乐传到远端
musicParam.loopCount = 0             // 替换为您需要循环播放的次数
audioEffectManager.startPlayMusic(musicParam) { code in
    if code == 0 {
      // 开始播放成功
    } else {
      // 开始播放失败
    }
} onProgress: { progress, duration in
    // 播放进度回调
} onComplete: { _ in
    // 播放结束回调
}

// 停止播放背景音乐
audioEffectManager.stopPlayMusic(musicId)   // 替换为您需要停止播放的音乐ID
```

Android

```
TXAudioEffectManager audioEffectManager =
TUIRoomEngine.sharedInstance().getTRTCCloud().getAudioEffectManager();

int id = 0;  // 替换为您自己的音乐ID
String path = "path"; // 替换为音乐文件的完整路径或 URL 地址
// 开始播放背景音乐
TXAudioEffectManager.AudioMusicParam musicParam = new
TXAudioEffectManager.AudioMusicParam(id, path);
musicParam.publish = true;            // 将音乐传到远端
musicParam.loopCount = 0;             // 替换为您需要循环播放的次数
audioEffectManager.startPlayMusic(musicParam);

// 停止播放背景音乐
audioEffectManager.stopPlayMusic(id);  // 替换为您需要停止播放的音乐ID
```

设置背景音乐音量

您可以通过调用 `setAllMusicVolume` 接口并传入一个Int值，来设置背景音乐的音量。
传入的Int值代表音量大小，取值范围为0 – 100。以下是一个调用 `setAllMusicVolume` 设置背景音乐音量的示例：

iOS

```
import RTCRoomEngine
import TXLiteAVSDK_Professional

let audioEffectManager =
TUIRoomEngine.sharedInstance().getTRTCCloud().getAudioEffectManager()

let volume = 60    // 替换为您需要设置的音量大小
audioEffectManager.setAllMusicVolume(volume)
```

Android

```
TXAudioEffectManager audioEffectManager =
TUIRoomEngine.sharedInstance().getTRTCCloud().getAudioEffectManager();

int volume = 60;    // 替换为您需要设置的音量大小
audioEffectManager.setAllMusicVolume(volume);
```

美颜相关

最近更新时间：2025-05-22 16:39:43

本文档主要介绍如何使用 `RTC Room Engine` SDK 实现美颜相关功能。

前提条件

在使用 `RTC Room Engine` SDK 提供的美颜设置相关功能之前，您需要先 登录 SDK。

使用指引

设置美颜级别

您可以通过调用 `setBeautyLevel` 接口，传入一个 Int 值以设置美颜级别。
传入的 Int 值为美颜级别，取值范围 0 – 9； 0 表示关闭，9 表示效果最明显。

iOS

```
import RTCRoomEngine
import TXLiteAVSDK_Professional

let beautyManager = TUIRoomEngine.sharedInstance().getTRTCCloud().getBeautyManager()

let beautyLevel = 6
beautyManager.setBeautyLevel(beautyLevel)  // 替换为您需要设置级别的Int值
```

Android

```
TXBeautyManager beautyManager =
TUIRoomEngine.sharedInstance().getTRTCCloud().getBeautyManager();

int beautyLevel = 6;
beautyManager.setBeautyLevel(beautyLevel);  // 替换为您需要设置级别的Int值
```

设置美白级别

您可以通过调用 `setWhitenessLevel` 接口，传入一个 Int 值以设置美白级别。
传入的 Int 值为美白级别，取值范围 0 – 9；0 表示关闭，9 表示效果最明显。

iOS

```
import RTCRoomEngine
import TXLiteAVSDK_Professional

let beautyManager = TUIRoomEngine.sharedInstance().getTRTCCloud().getBeautyManager()

let whitenessLevel = 6
beautyManager.setWhitenessLevel(whitenessLevel)  // 替换为您需要设置级别的Int值
```

Android

```
TXBeautyManager beautyManager =
TUIRoomEngine.sharedInstance().getTRTCCloud().getBeautyManager();

int whitenessLevel = 6;
beautyManager.setWhitenessLevel(whitenessLevel);  // 替换为您需要设置级别的Int值
```

设置红润级别

您可以通过调用 `setRuddyLevel` 接口，传入一个 Int 值以设置红润级别。
传入的 Int 值为红润级别，取值范围 0 – 9；0 表示关闭，9 表示效果最明显。

iOS

```
import RTCRoomEngine
import TXLiteAVSDK_Professional

let beautyManager = TUIRoomEngine.sharedInstance().getTRTCCloud().getBeautyManager()

let ruddyLevel = 6
beautyManager.setRuddyLevel(ruddyLevel)  // 替换为您需要设置级别的Int值
```

Android

```
TXBeautyManager beautyManager =
TUIRoomEngine.sharedInstance().getTRTCCloud().getBeautyManager();

int ruddyLevel = 6;
beautyManager.setRuddyLevel(ruddyLevel);  // 替换为您需要设置级别的Int值
```

说明：
如您需要使用更多美颜的相关功能，详细请参见 TXBeatyManager 。

音效设置

最近更新时间：2025-05-22 16:39:43

本文档主要介绍如何使用 `RTC Room Engine` SDK 实现音效设置的相关功能。

前提条件

在使用 `RTC Room Engine` SDK 提供的背景音乐设置相关功能之前，您需要先 登录 。

使用指引

设置背景音乐音量

您可以通过调用 `setAllMusicVolume` 接口并传入一个 Int 值，来设置背景音乐的音量。
传入的 Int 值代表音量大小，取值范围为0 – 100。以下是一个调用 `setAllMusicVolume` 设置背景音乐音量的示例：

iOS

```
import RTCRoomEngine
import TXLiteAVSDK_Professional

let audioEffectManager =
TUIRoomEngine.sharedInstance().getTRTCCloud().getAudioEffectManager()

let volume = 60
audioEffectManager.setAllMusicVolume(volume)
```

Android

```
TXAudioEffectManager audioEffectManager =
TUIRoomEngine.sharedInstance().getTRTCCloud().getAudioEffectManager();

int volume = 60;
audioEffectManager.setAllMusicVolume(volume);
```

设置语音音量

您可以通过调用 `setVoiceVolume` 接口并传入一个 Int 值，来设置背景音乐的音量。
传入的 Int 值代表音量大小，取值范围为0 – 100。以下是一个调用 `setVoiceVolume` 设置语音音量为60的示例：

iOS

```
import RTCRoomEngine
import TXLiteAVSDK_Professional

let audioEffectManager =
TUIRoomEngine.sharedInstance().getTRTCCloud().getAudioEffectManager()

let volume = 60
audioEffectManager.setVoiceVolume(volume)
```

Android

```
TXAudioEffectManager audioEffectManager =
TUIRoomEngine.sharedInstance().getTRTCCloud().getAudioEffectManager();

int volume = 60;
audioEffectManager.setVoiceCaptureVolume(volume);
```

开启/关闭耳返

您可以通过调用 `enableVoiceEarMonitor` 接口并传入一个 Bool 值，来设置是否开启耳返。
传入的 Bool 值代表是否开启耳返，true 代表开启耳返，false 代表关闭耳返。以下是一个调用 `enableVoiceEarMonitor` 接口开启耳返的示例：

iOS

```
import RTCRoomEngine
import TXLiteAVSDK_Professional

let audioEffectManager =
TUIRoomEngine.sharedInstance().getTRTCCloud().getAudioEffectManager()

let enable = true
audioEffectManager.enableVoiceEarMonitor(enable)
```

Android

```
TXAudioEffectManager audioEffectManager =
TUIRoomEngine.sharedInstance().getTRTCCloud().getAudioEffectManager();

boolean enable = true;
audioEffectManager.enableVoiceEarMonitor(enable);
```

设置耳返音量

您可以通过调用 `setVoiceEarMonitorVolume` 接口并传入一个 Int 值，来设置耳返的音量。
传入的 Int 值代表音量大小，取值范围为0 – 100。以下是一个调用 `setVoiceEarMonitorVolume` 设置耳返音量为60的示例：

iOS

```
import RTCRoomEngine
import TXLiteAVSDK_Professional

let audioEffectManager =
TUIRoomEngine.sharedInstance().getTRTCCloud().getAudioEffectManager()

let volume = 60
audioEffectManager.setVoiceEarMonitorVolume(volume)
```

版权所有：腾讯云计算（北京）有限责任公司

Android

```
TXAudioEffectManager audioEffectManager =
TUIRoomEngine.sharedInstance().getTRTCCloud().getAudioEffectManager();

int volume = 60;
audioEffectManager.setVoiceEarMonitorVolume(volume);
```

设置人声变声效果

您可以通过调用 `setVoiceChangerType` 接口并传入一个 `TXVoiceChangeType` 值，来设置人声的变声效果。
`TXVoiceChangeType` 为一个枚举，包含以下类型：

枚举	取值	描述
TXVoiceChangeType_0	0	关闭
TXVoiceChangeType_1	1	熊孩子
TXVoiceChangeType_2	2	萝莉
TXVoiceChangeType_3	3	大叔
TXVoiceChangeType_4	4	重金属
TXVoiceChangeType_5	5	感冒
TXVoiceChangeType_6	6	外语腔
TXVoiceChangeType_7	7	困兽
TXVoiceChangeType_8	8	肥宅
TXVoiceChangeType_9	9	强电流
TXVoiceChangeType_10	10	重机械
TXVoiceChangeType_11	11	空灵

以下是一个调用 `setVoiceChangerType` 接口设置人声变声效果为 熊孩子 的示例：

iOS

```
import RTCRoomEngine
import TXLiteAVSDK_Professional

let audioEffectManager =
TUIRoomEngine.sharedInstance().getTRTCCloud().getAudioEffectManager()

let type: TXVoiceChangeType = ._1
audioEffectManager.setVoiceChangerType(type)
```

Android

```
TXAudioEffectManager audioEffectManager =
TUIRoomEngine.sharedInstance().getTRTCCloud().getAudioEffectManager();

TXAudioEffectManager.TXVoiceChangerType type =
TXAudioEffectManager.TXVoiceChangerType.TXLiveVoiceChangerType_1;
audioEffectManager.setVoiceChangerType(type);
```

设置混响效果

您可以通过调用 `setVoiceReverbType` 接口并传入一个 `TXVoiceReverbType` 值，来设置混响效果。
`TXVoiceReverbType` 为一个枚举，包含以下类型：

枚举	取值	描述
TXVoiceReverbType_0	0	关闭特效
TXVoiceReverbType_1	1	KTV
TXVoiceReverbType_2	2	小房间
TXVoiceReverbType_3	3	大会堂
TXVoiceReverbType_4	4	低沉
TXVoiceReverbType_5	5	洪亮
TXVoiceReverbType_6	6	金属声
TXVoiceReverbType_7	7	磁性
TXVoiceReverbType_8	8	空灵
TXVoiceReverbType_9	9	录音棚
TXVoiceReverbType_10	10	悠扬
TXVoiceReverbType_11	11	录音棚2

以下是一个调用 `setVoiceReverbType` 接口设置混响效果为 KTV 的示例：

iOS

```
import RTCRoomEngine
import TXLiteAVSDK_Professional

let audioEffectManager =
TUIRoomEngine.sharedInstance().getTRTCCloud().getAudioEffectManager()

let type: TXVoiceReverbType = ._1
audioEffectManager.setVoiceReverbType(type)
```

Android

```
TXAudioEffectManager audioEffectManager =
TUIRoomEngine.sharedInstance().getTRTCCloud().getAudioEffectManager();

TXAudioEffectManager.TXVoiceReverbType type =
TXAudioEffectManager.TXVoiceReverbType.TXLiveVoiceReverbType_1;
audioEffectManager.setVoiceReverbType(type);
```

说明：
如您需要使用更多音效设置的相关功能，详细请参见 TXAudioEffectManager 。

屏幕共享
iOS

最近更新时间：2025-05-22 16:39:43

在 iOS 平台下支持两种不同的屏幕分享方案：

- **应用内分享**

  即只能分享当前 App 的画面，该特性需要 iOS 13 及以上版本的操作系统才能支持。由于无法分享当前 App 之外的屏幕内容，因此适用于对隐私保护要求高的场景。
- **跨应用分享**

  基于苹果的 Replaykit 方案，能够分享整个系统的屏幕内容，但需要当前 App 额外提供一个 Extension 扩展组件，因此对接步骤也相对应用内分享要多一点。

支持的平台

iOS	Android	macOS	Windows	Electron	Chrome 浏览器
✓	✓	✓	✓	✓	✓

应用内分享

应用内分享的方案非常简单，只需要调用 RoomEngine 提供的接口 `startScreenCapture` 并传入参数 `appgroup` 即可。
同时您可以调用 `updateVideoQualityEx` 修改编码参数，我们推荐的用于 iOS 屏幕分享的编码参数是：

参数项	参数名称	常规推荐值	文字教学场景
分辨率	videoResolution	1280 × 720	1920 × 1080
帧率	fps	10 FPS	8 FPS
最高码率	bitrate	1600 kbps	2000 kbps

- 由于屏幕分享的内容一般不会剧烈变动，所以设置较高的 FPS 并不经济，推荐10 FPS即可。
- 如果您需要更高的画质，请调高FPS。
- 如果您要分享的屏幕内容包含大量文字，可以适当提高分辨率和码率设置。
- 最高码率（videoBitrate）是指画面在剧烈变化时的最高输出码率，如果屏幕内容变化较少，实际编码码率会比较低。

示例：

```
import RTCRoomEngine

let roomEngine = TUIRoomEngine.sharedInstance()

roomEngine.startScreenCapture(appGroup: "")

let params = TUIRoomVideoEncoderParams()
params.fps = 10 //替换为您真实需要的值
params.resolutionMode = .portrait //竖屏分辨率
params.bitrate = .1600 //此处换为您真实需要的值
params.videoResolution = .quality720P //此处换位您需要的值
```

```
roomEngine.updateVideoQualityEx(streamType: .screenStream, params: params)
```

跨应用分享

iOS 系统上的跨应用屏幕分享，需要增加 **Broadcast Upload Extension** 录屏进程以配合主 App 进程进行推流。Extension 录屏进程由系统在需要录屏的时候创建，并负责接收系统采集到屏幕图像。因此需要：

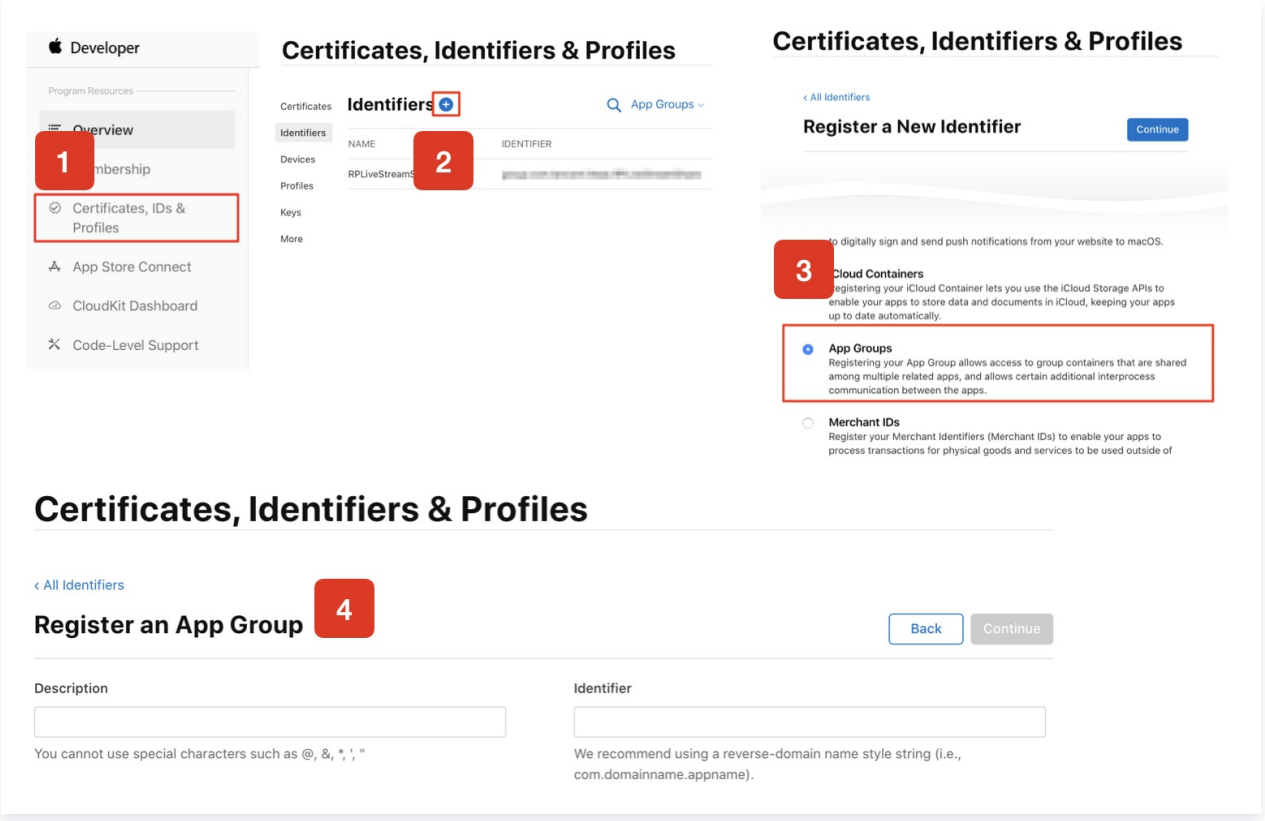
1. 创建 **App Group**，并在 XCode 中进行配置（可选）。这一步的目的是让 Extension 录屏进程可以同主 App 进程进行跨进程通信。
2. 在您的工程中，新建一个 **Broadcast Upload Extension** 的 Target，并在其中集成 SDK 压缩包中专门为扩展模块定制的 `TXLiteAVSDK_ReplayKitExt.framework` 。
3. 对接主 App 端的接收逻辑，让主 App 等待来自 **Broadcast Upload Extension** 的录屏数据。

> **注意：**
> 如果跳过第1步，也就是不配置 App Group（接口传 nil），屏幕分享依然可以运行，但稳定性要打折扣，故虽然步骤较多，但请尽量配置正确的 App Group 以保障屏幕分享功能的稳定性。

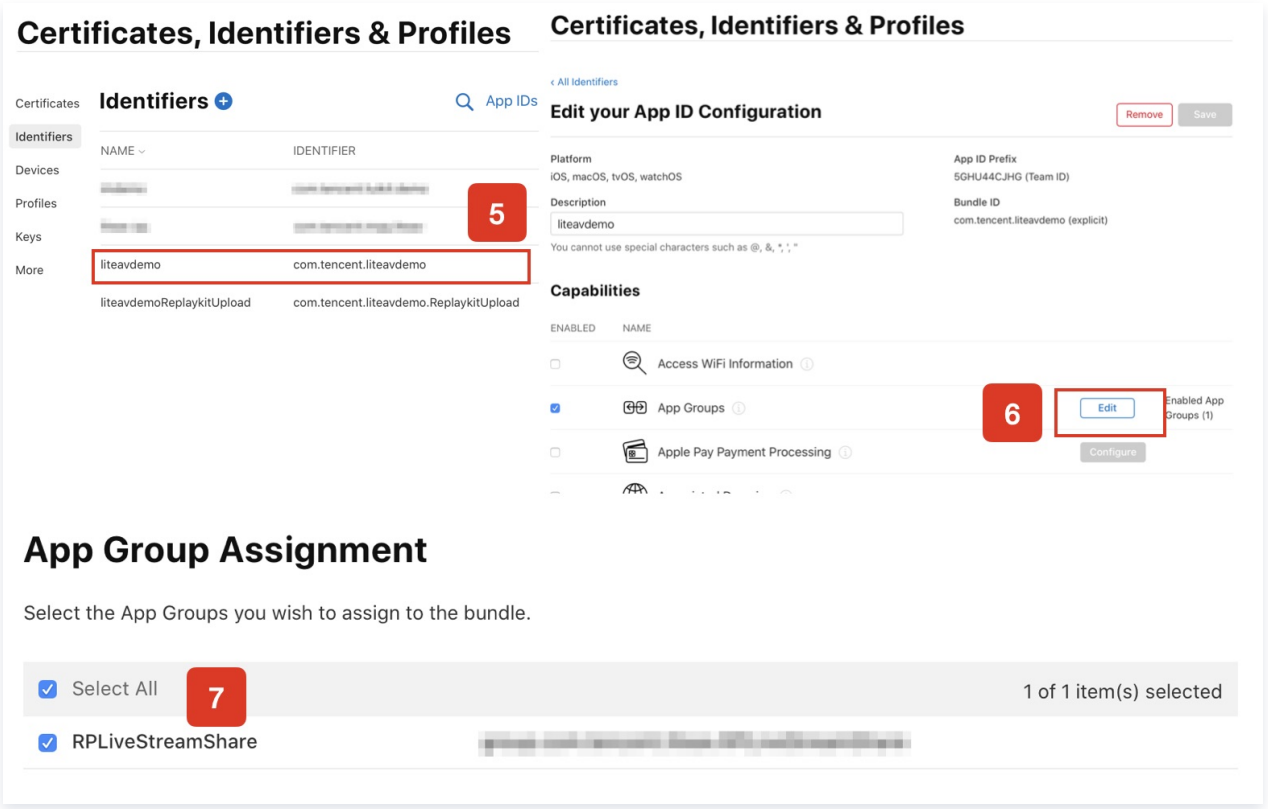
步骤1：创建 App Group

使用您的账号登录 Apple 开发者官方网站 ，进行以下操作，**注意完成后需要重新下载对应的 Provisioning Profile**。

1. 单击 **Certificates, IDs & Profiles**。
2. 在右侧的界面中单击加号。
3. 选择 **App Groups**，单击 **Continue**。
4. 在弹出的表单中填写 Description 和 Identifier，其中 Identifier 需要传入接口中的对应的 AppGroup 参数。完成后单击 **Continue**。



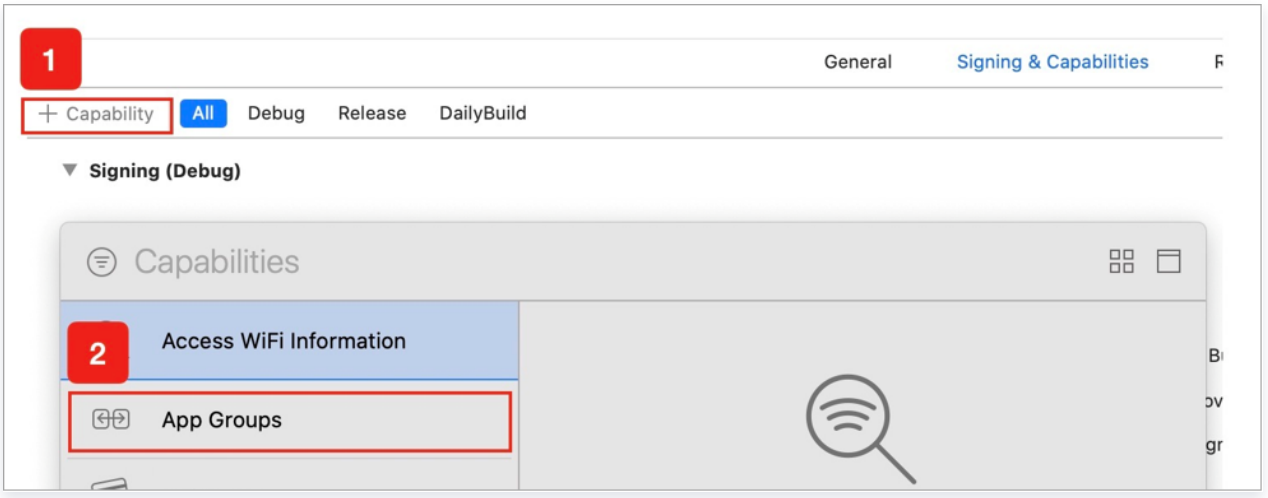
5. 回到 Identifier 页面，左上边的菜单中选择 **App IDs**，然后单击您的 App ID（主 App 与 Extension 的 AppID 需要进行同样的配置）。
6. 选中 **App Groups** 并单击 **Edit**。
7. 在弹出的表单中选择您之前创建的 App Group，单击 **Continue** 返回编辑页，单击 **Save** 保存。



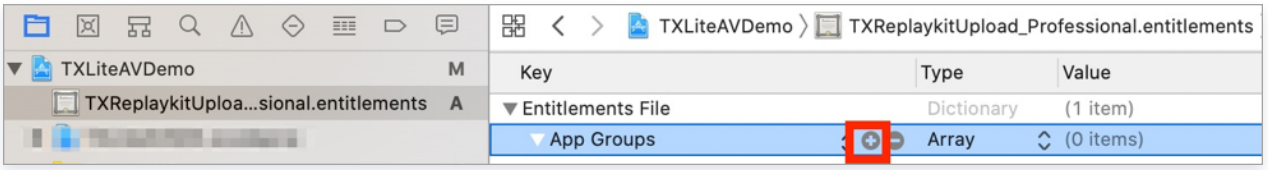
8. 重新下载 Provisioning Profile 并配置到 XCode 中。

步骤2：创建 Broadcast Upload Extension

1. 在 Xcode 菜单依次单击 **File**、**New** 、**Target...**，选择 **Broadcast Upload Extension**。
2. 在弹出的对话框中填写相关信息，不用勾选 **Include UI Extension**，单击 **Finish** 完成创建。
3. 将下载到的 SDK 压缩包中的 TXLiteAVSDK_ReplayKitExt.framework 拖动到工程中，勾选刚创建的 Target。
4. 选中新增加的 Target，依次单击 **+ Capability**，双击 **App Groups**，如下图：



操作完成后，会在文件列表中生成一个名为 `Target名.entitlements` 的文件，如下图所示，选中该文件并单击 + 号填写上述步骤中的 App Group 即可。



5. 选中主 App 的 Target，**并按照上述步骤对主 App 的 Target 做同样的处理。**
6. 在新创建的 Target 中，Xcode 会自动创建一个名为 "SampleHandler.h" 的文件，用如下代码进行替换。**需将代码中的 APPGROUP 改为上文中的创建的 App Group Identifier。**

```
import ReplayKit
import TXLiteAVSDK_ReplayKitExt

let APPGROUP = ""

class SampleHandler: RPBroadcastSampleHandler, TXReplayKitExtDelegate {

    let recordScreenKey = Notification.Name.init("TRTCRecordScreenKey")

    override func broadcastStarted(withSetupInfo setupInfo: [String : NSObject]?) {
        if let setupInfo = setupInfo {
        }
        TXReplayKitExt.sharedInstance().setup(withAppGroup: APPGROUP, delegate: self)
    }

    override func broadcastPaused() {
        // 可以添加暂停时的资源释放或状态保存逻辑
    }

    override func broadcastResumed() {
        // 可以添加恢复时的资源重新初始化逻辑
    }

    override func broadcastFinished(){
        // User has requested to finish the broadcast.
        TXReplayKitExt.sharedInstance().broadcastFinished()
    }

    func broadcastFinished(_ broadcast: TXReplayKitExt, reason: TXReplayKitExtReason) {
        var tip = ""
        switch reason {
            case TXReplayKitExtReason.requestedByMain:
                tip = "屏幕共享已结束"
                break
            case TXReplayKitExtReason.disconnected:
                tip = "应用断开"
                break
            case TXReplayKitExtReason.versionMismatch:
                tip = "集成错误（SDK 版本号不相符合）"
                break
            default:
                break
        }
```

```
        let error = NSError(domain: NSStringFromClass(self.classForCoder), code: 0,
userInfo: [NSLocalizedFailureReasonErrorKey:tip])
        finishBroadcastWithError(error)
    }

    override func processSampleBuffer(_ sampleBuffer: CMSampleBuffer, with
sampleBufferType: RPSampleBufferType) {
        switch sampleBufferType {
            case RPSampleBufferType.video:
                TXReplayKitExt
                    .sharedInstance()
                    .send(sampleBuffer, with: .video)
                break
            case RPSampleBufferType.audioApp:
                // 可以在此处处理应用音频
                break
            case RPSampleBufferType.audioMic:
                // 可以在此处处理麦克风音频
                break
            @unknown default:
                let error = "未知的采样缓冲区类型：\(sampleBufferType)"
                finishBroadcastWithError(NSError(domain: error, code: -1, userInfo: nil))
        }
    }
}
```

步骤3：对接主 App 端的接收逻辑

按照如下步骤，对接主 App 端的接收逻辑。也就是在用户触发屏幕分享之前，要让主 App 处于“等待”状态，以便随时接收来自 Broadcast Upload Extension 进程的录屏数据。

1. 调用 `startScreenCapture` 方法，并传入 步骤1 中设置的 AppGroup，让 SDK 进入“等待”状态。
2. 等待用户触发屏幕分享。如果不实现 步骤4 中的“触发按钮”，屏幕分享就需要用户在 iOS 系统的控制中心，通过长按录屏按钮来触发，这一操作步骤如下图所示：
3. 通过调用 `stopScreenCapture` 接口可以随时中止屏幕分享。

示例：

```
import RTCRoomEngine

TUIRoomEngine.sharedInstance().startScreenCapture(appGroup: "")
TUIRoomEngine.sharedInstance().stopScreenCapture()
```

步骤4：增加屏幕分享的触发按钮（可选）

截止到 步骤3 ，我们的屏幕分享还必须要用户从控制中心中长按录屏按钮来手动启动。您可通过下述方法实现类似腾讯会议的单击按钮即可触发的效果：

1. 在 `TRTCBroadcastExtensionLauncher` 文件实现了唤起屏幕分享，将其加入到您的工程中。
2. 在您的界面上放置一个按钮，并在按钮的响应函数中调用 `TRTCBroadcastExtensionLauncher` 中的 `launch` 函数，就可以唤起屏幕分享功能了。

示例：

```
@objc private func buttonTapped() {
    TRTCBroadcastExtensionLauncher.sharedInstance.launch()
}
```

```
import UIKit
import TXLiteAVSDK_ReplayKitExt

class TRTCBroadcastExtensionLauncher: NSObject {

    var systemBroacastExtensionPicker = RPSystemBroadcastPickerView()
    var prevLaunchEventTime : CFTimeInterval = 0

    static let sharedInstance =  TRTCBroadcastExtensionLauncher()

    override init() {
        super.init()
        let picker = RPSystemBroadcastPickerView(frame: CGRect(x: 0, y: 0, width: 44,
height: 44))
        picker.showsMicrophoneButton = false
        picker.autoresizingMask = [.flexibleTopMargin, .flexibleRightMargin]
        systemBroacastExtensionPicker = picker

        if let pluginPath = Bundle.main.builtInPlugInsPath,
           let contents = try? FileManager.default.contentsOfDirectory(atPath: pluginPath) {

            for content in contents where content.hasSuffix(".appex") {
                guard let bundle = Bundle(path: URL(fileURLWithPath:
pluginPath).appendingPathComponent(content).path),
                      let identifier : String = (bundle.infoDictionary?["NSExtension"] as?
[String:Any])? ["NSExtensionPointIdentifier"] as? String
                else {
                    continue
                }
                if identifier == "com.apple.broadcast-services-upload" {
                    picker.preferredExtension = bundle.bundleIdentifier
                    break
                }
            }
        }
    }

    static func launch() {
        TRTCBroadcastExtensionLauncher.sharedInstance.launch()
    }

    func launch() {
        // The pop-up on iOS 12 is slow and will crash if you click quickly.
        let now = CFAbsoluteTimeGetCurrent()
        if now - prevLaunchEventTime < 1.0 {
            return
        }
        prevLaunchEventTime = now
```

```
        for view in systemBroacastExtensionPicker.subviews {
            if let button = view as? UIButton {
                button.sendActions(for: .allTouchEvents)
                break
            }
        }
    }
}
```

> ⚠ **注意：**
>
> - 苹果在 iOS 12.0 中增加了 `RPSystemBroadcastPickerView` 可以从应用中弹出启动器供用户确认启动屏幕分享，到目前为止，`RPSystemBroadcastPickerView` 尚不支持自定义界面，也没有官方的唤起方法。
> - TRTCBroadcastExtensionLauncher 的原理就是遍历 `RPSystemBroadcastPickerView` 的子 View 寻找 UIButton 并触发了其点击事件。
> - 但该方案不被苹果官方推荐，并可能在新一轮的系统更新中失效，因此 步骤4 只是一个可选方案，您需要自行承担风险来选用此方案。

步骤5：混流模式下屏幕共享

当您完成了上述逻辑之后，您需要调用 setLiveStreamLayoutInfo API 修改画面布局。

```
import RTCRoomEngine

let roomEngine = TUIRoomEngine.sharedInstance()
let layoutManager = roomEngine.getExtension(extensionType: .liveLayoutManager) as?
TUILiveLayoutManager
roomEngine.startScreenCapture(appGroup: "")//替换为真实的appGroup

layoutManager.setLiveStreamLayoutInfo(
    roomID: roomInfo.roomId,
    layoutInfo: "" //详见下文
    onSuccess: {
    //修改成功
    },
    onError: {code ,message in
    }
)

//layoutInfo格式需求：
{
    "RoomId": "live_adams", //替换为您的RoomID
    //当layouttype是1000时候，修改VideoEncode时候需要同步修改LayoutInfo，九宫格模式下会默认自动根据分辨
率修改更新layoutInfo
    "VideoEncode": {
        "Width": 1080,
        "Height": 1920,
    },
    "LayoutMode": 1000, // 0~9 内置布局模板， 1000自定义布局， 只有1000时候才能修改LayoutInfo，0为九
宫格
```

```
    "LayoutInfo":{
        "LayoutList": [
            {
                "LocationX": 0,          // 具体数值
                "LocationY": 0,          // 具体数值
                "ImageWidth": 1080,      // 具体数值
                "ImageHeight": 960,      // 具体数值
                "ZOrder": 0,             // 层级
                "StreamType": 0,         // 0为摄像头， 1为屏幕共享
                "Member_Account": "admin001",  //替换为您的userId
                "BackgroundImageUrl": "ImageUrl",
                "RoomId":"live_adams", //替换为您的RoomId
                "BackgroundColor":"0x1F212C"
            },
            {
                "LocationX": 0,          // 具体数值
                "LocationY": 960,        // 具体数值
                "ImageWidth": 1080,      // 具体数值
                "ImageHeight": 960,      // 具体数值
                "ZOrder": 0,             // 层级
                "StreamType": 0,         // 0为摄像头， 1为屏幕共享
                "Member_Account": "admin001",    //替换为您的UserId
                "BackgroundImageUrl": "ImageUrl",
                "RoomId":"live_adams", //替换为您的RoomId
                "BackgroundColor":"0x1F212C"
            }
        ],
        "MaxUserLayout": {
                "LocationX": 0, // 具体数值
                "LocationY": 0, // 具体数值
                "ImageWidth": 1080, // 具体数值
                "ImageHeight": 1920, // 具体数值
                "ZOrder": 0,         //层级
                "StreamType": 0,     // 0为摄像头， 1为屏幕共享
                "Member_Account": "admin001",  //替换为您的userId
                "BackgroundImageUrl": "ImageUrl",
                "RoomId":"live_adams", //替换为您的roomId
                "BackgroundColor":"0x1F212C"
        }
}
```

步骤6：混流模式下停止分享

当您想要停止屏幕共享并且将画面布局修改回来时，您需要调用 stopScreenCapture 接口关闭并且调用 setLiveStreamLayoutInfo 接口恢复画面布局。

```
import RTCRoomEngine

let roomEngine = TUIRoomEngine.sharedInstance()
let layoutManager = roomEngine.getExtension(extensionType: .liveLayoutManager) as?
TUILiveLayoutManager
roomEngine.stopScreenCapture()
```

```
layoutManager.setLiveStreamLayoutInfo(
    roomID: roomInfo.roomId,
    layoutInfo: "",//详见下文
    onSuccess: {},
    onError: { code ,message in
    }
 )
//当您想要恢复默认的宫格布局时，您需要传入的layoutInfo参数如下：
{
    "RoomId": "live_12121", //替换为真实房间号
    "VideoEncode": {
        "Width": 1080,
        "Height": 1920
    },
    "LayoutMode": 0
}
```

观看屏幕分享

当有用户播放屏幕共享时您可以通过监听 TUIRoomObserver 中的 onUserVideoStateChanged 来感知到屏幕共享流。
示例：

```
import RTCRoomEngine

extension ScreenSharedController: TUIRoomObserver { //替换为您具体的业务类，此处仅为举例
      func onUserVideoStateChanged(userId: String, streamType: TUIVideoStreamType, hasVideo:
Bool, reason: TUIChangeReason) {
      }
 }
```

错误码表

最近更新时间：2025-05-22 16:39:43

TUIError

TUIError

错误码枚举定义

枚举	取值	描述
SUCCESS	0	操作成功。
FAILED	-1	暂未归类的通用错误。
FREQ_LIMIT	-2	请求被限频，请稍后重试。
REPEAT_OPERATION	-3	重复操作。
SDKAPPID_NOT_FOUND	-1000	未找到 SDKAppID，请在腾讯云视立方 SDK 控制台 确认应用信息。
INVALID_PARAMETER	-1001	调用 API 时，传入的参数不合法，检查入参是否合法。
SDK_NOT_INITIALIZED	-1002	未登录,请调用 Login 接口。
PERMISSION_DENIED	-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。Room场景下请使用以下错误码来处理: 摄像头没有系统授权: ERR_CAMERA_NOT_AUTHORIZED。 麦克风没有系统授权: ERR_MICROPHONE_NOT_AUTHORIZED。
REQUIRE_PAYMENT	-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。
CAMERA_START_FAIL	-1100	系统问题，打开摄像头失败。检查摄像头设备是否正常。
CAMERA_NOT_AUTHORIZED	-1101	摄像头没有系统授权, 检查系统授权。
CAMERA_OCCUPIED	-1102	摄像头被占用，检查是否有其他进程使用摄像头。
CAMERA_DEVICE_EMPTY	-1103	当前无摄像头设备，请插入摄像头设备解决该问题。
MICROPHONE_START_FAIL	-1104	系统问题，打开麦克风失败。检查麦克风设备是否正常。
MICROPHONE_NOT_AUTHORIZED	-1105	麦克风没有系统授权，检查系统授权。
MICROPHONE_OCCUPIED	-1106	麦克风被占用。
MICROPHONE_DEVICE_EMPTY	-1107	当前无麦克风设备。

GET_SCREEN_SHARING_TARGET_FAILED	-1108	获取屏幕分享源（屏幕和窗口）失败，检查屏幕录制权限。
START_SCREEN_SHARING_FAILED	-1109	开启屏幕分享失败，检查房间内是否有人正在屏幕分享。
ROOM_ID_NOT_EXIST	-2100	进房时房间不存在，或许已被解散。
OPERATION_INVALID_BEFORE_ENTER_ROOM	-2101	需要进房后才可使用此功能。
EXIT_NOT_SUPPORTED_FOR_ROOM_OWNER	-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。
OPERATION_NOT_SUPPORTED_IN_CURRENT_ROOM_TYPE	-2103	当前房间类型下不支持该操作。
ROOM_ID_INVALID	-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。
ROOM_ID_OCCUPIED	-2106	房间ID 已被使用，请选择别的房间ID。
ROOM_NAME_INVALID	-2107	房间名称非法，名称最长30字节，如果包含中文，字符编码必须是 UTF-8。
ALREADY_IN_OTHER_ROOM	-2108	当前用户已在别的房间内，需要先退房才能加入新的房间: 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。
NEED_PASSWORD	-2109	当前房间需要密码才能进入。
WRONG_PASSWORD	-2110	进房密码错误。
ROOM_USER_FULL	-2111	房间成员已满。
ROOM_METADATA_EXCEED_KEY_COUNT_LIMIT	-2112	房间自定义信息 key 数量超过上限
ROOM_METADATA_EXCEED_VALUE_SIZE_LIMIT	-2113	房间自定义信息 value 字节大小超过上限
USER_NOT_EXIST	-2200	用户不存在。
USER_NOT_ENTERED	-2201	用户不在当前房间内。
NEED_OWNER_PERMISSION	-2300	需要房主权限才能操作。
NEED_ADMIN_PERMISSION	-2301	需要房主或者管理员权限才能操作。
REQUEST_NO_PERMISSION	-2310	信令请求无权限，例如取消非自己发起的邀请。
REQUEST_ID_INVALID	-2311	信令请求 ID 无效或已经被处理过。
REQUEST_ID_REPEAT	-2312	信令请求重复。
REQUEST_ID_CONFLICT	-2313	信令请求冲突。

版权所有：腾讯云计算（北京）有限责任公司

MAX_SEAT_COUNT_LIMIT	-2340	最大麦位超出套餐包数量限制。
ALREADY_IN_SEAT	-2341	当前用户已经在麦位上。
SEAT_OCCUPIED	-2342	当前麦位已经有人了。
SEAT_LOCKED	-2343	当前麦位被锁。
SEAT_INDEX_NOT_EXIST	-2344	麦位编号不存在。
USER_NOT_IN_SEAT	-2345	当前用户没有在麦上。
ALL_SEAT_OCCUPIED	-2346	上麦人数已满。
SEAT_NOT_SUPPORT_LINK_MIC	-2347	不支持连麦。
OPEN_MICROPHONE_NEED_SEAT_UNLOCK	-2360	当前麦位音频被锁。
OPEN_MICROPHONE_NEED_PERMISSION_FROM_ADMIN	-2361	需要向房主或管理员申请后打开麦克风。
OPEN_CAMERA_NEED_SEAT_UNLOCK	-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。
OPEN_CAMERA_NEED_PERMISSION_FROM_ADMIN	-2371	需要向房主或管理员申请后打开摄像头。
OPEN_SCREEN_SHARE_NEED_SEAT_UNLOCK	-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。
OPEN_SCREEN_SHARE_NEED_PERMISSION_FROM_ADMIN	-2373	需要向房主或管理员申请后打开屏幕分享。
SEND_MESSAGE_DISABLED_FOR_ALL	-2380	当前房间已开启全员禁言。
SEND_MESSAGE_DISABLED_FOR_CURRENT	-2381	当前房间内，您已被已禁言。
ROOM_ALREADY_CONNECTED	-3001	当前房间已连线。
ROOM_CONNECTED_IN_OTHER	-3002	当前房间与其他房间连线中。
MAX_CONNECTED_COUNT_LIMIT	-3003	当前房间连线超出最大数量限制。
ROOM_NOT_SUPPORT_PRELOADING	-4001	当前房间不支持预加载。

版权所有：腾讯云计算（北京）有限责任公司