# 实时音视频

# AI 实时对话



腾讯云

版权所有：腾讯云计算（北京）有限责任公司

【版权声明】

©2013-2026 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】



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

AI 实时对话
AI 实时对话 2.0
功能介绍
开通服务
快速开始
发起 AI 实时语音对话（三段式）
发起 AI 实时语音对话（S2S）
基础功能
实时字幕（客户端）
打断指令（客户端）
接收 AI 机器人状态（客户端）
ASR 配置
LLM 配置
TTS 配置
S2S 配置
主动送文本给 LLM 或者 TTS
获取 AI 机器人状态以及上下文
高级功能
视觉理解能力（截图问答）
AI 对话 2.0 服务端回调
AI 实时对话 1.0
功能介绍
开通服务
准备工作
跑通 Demo
跑通 Demo（Android）
跑通 Demo（iOS）
跑通 Demo（Web）
全功能集成
全功能集成（Android）
全功能集成（iOS）
原子组件集成
原子组件集成（Android）
原子组件集成（iOS）
更多接入方案

版权所有：腾讯云计算（北京）有限责任公司

TRTC SDK 接入
高级设置
大模型配置
文字转语音配置
智能打断逻辑
如何实现上下文管理
AI 对话 SDK 端回调
AI 对话状态回调
AI 对话字幕回调
AI 对话指标回调
AI 对话错误回调
AI 对话服务端回调
自定义 TTS 协议
使用语义断句
更多功能
大模型透传自定义消息
AI 对话声纹能力接入
IM 信令方案
行业实践教程
情感陪伴
智能客服

版权所有：腾讯云计算（北京）有限责任公司

# AI 实时对话
# AI 实时对话 2.0
# 功能介绍

最近更新时间：2026-10-29 21:43:32

## 产品概述

TRTC AI 实时对话 2.0 是腾讯云实时音视频（TRTC）面向 AI 语音对话场景提供的服务端能力产品。AI 服务端作为机器人加入 TRTC 房间，订阅真实用户音视频流，可通过接入 SDK，调用服务端接口，完成实时语音对话链路，将合成/生成的音频回传到房间，实现自然、可打断的多轮语音对话。
基于腾讯 RTC 全球低延迟传输网络，语音对话延迟低，对话效果自然拟真，接入便捷，开箱即用。支持灵活接入多家 AI 大模型服务，打造符合业务场景的 AI 实时对话能力。

## 产品架构图




## 核心能力

- **链路能力。**

- **Cascade (三段式) 模式**：实时 ASR、LLM 流式推理、流式 TTS、端到端打断。
- **S2S（端到端) 模式**：音频直入直出的端到端语音大模型。
- **接入能力**：TRTC 自研 ASR / TTS 即开即用，支持自定义 ASR / LLM / TTS（OpenAI 兼容协议）。
- **消息通道**：SDK 与后台走自定义数据通道，承载字幕、状态、打断指令，无需引入额外信令通道。
- **视觉理解（VLM）**：截图对话能力，支持视频截图对话、定帧提问、本地相册提问。
- **对话控制**：欢迎语、静音追问、强制打断、运行时更新 LLM 配置。
- **可观测能力**：ASR / LLM / TTS / S2S 全过程事件回调、字幕回调、AI 状态流转回调。
- **生命周期管理**：创建、查询、修改、控制、删除五类标准 RESTful 操作。

## 应用场景

| 场景 | 说明 |
| --- | --- |
| **智能助手 / 智能硬件** | 手机、智能硬件的 AI 实时语音助手，边通话边看屏 + 定帧问答（推荐 Cascade + VLM）。 |
| **社交娱乐** | AI 陪伴、在线剧本杀、狼人杀等互动游戏中 AI 扮演主持或 NPC 角色，与玩家动态对话并推动情节发展。 |
| **智能客服 / 数字人** | 在线客服、AI 销售顾问、智能外呼、数字人直播，全天候实时语音问答，降低运营成本。 |
| **高效办公** | 语音命令和控制应用程序，拓展办公助手使用场景。 |
| **在线教育** | 虚拟教学助手，课内随时提问补充讲解，课外个性化辅导。 |
| **医疗辅助** | 远程诊断、医疗咨询，对话式交互更接近真实问诊体验。 |

## 功能优势

| 优势 | 说明 |
| --- | --- |
| **超低延迟的 AI 实时对话** | 全球范围内音视频传输端到端延迟低于 300ms，对话延迟保持在 1000ms 以下，媲美人类对话反应速度。 |
| **自然拟真的对话** | 支持中文普通话、方言、中英混说及 20+ 小语种；自研 ASR 字准 96% 以上、首包约 400ms；结合语义 VAD 实现智能语义打断，适应人类对话节奏。 |
| **服务模型集成灵活性** | ASR / LLM / TTS 各段均可独立配置厂商；LLM 按 OpenAI 兼容协议对接，仅需配置服务凭证即可无缝集成第三方模型。 |
| **接入简单，高效上线** | 服务端 REST API 五类标准操作管理任务生命周期，集成可在 1–2 天内完成；LLM 配置支持运行时热更新。 |

版权所有：腾讯云计算（北京）有限责任公司

| | |
|---|---|
| **视觉理解（VLM）** | 支持截图对话：屏幕/摄像头画面按需截取，连同用户语音一同送多模态 LLM 理解，覆盖“边打电话边看屏”场景。 |
| **领先的音频处理** | 支持服务端和客户端 AI 降噪、声纹降噪、3A 回声消除，提高语音识别精度和 AI 通话质量。 |

版权所有：腾讯云计算（北京）有限责任公司

# 开通服务

最近更新时间：2026-11-02 18:14:04

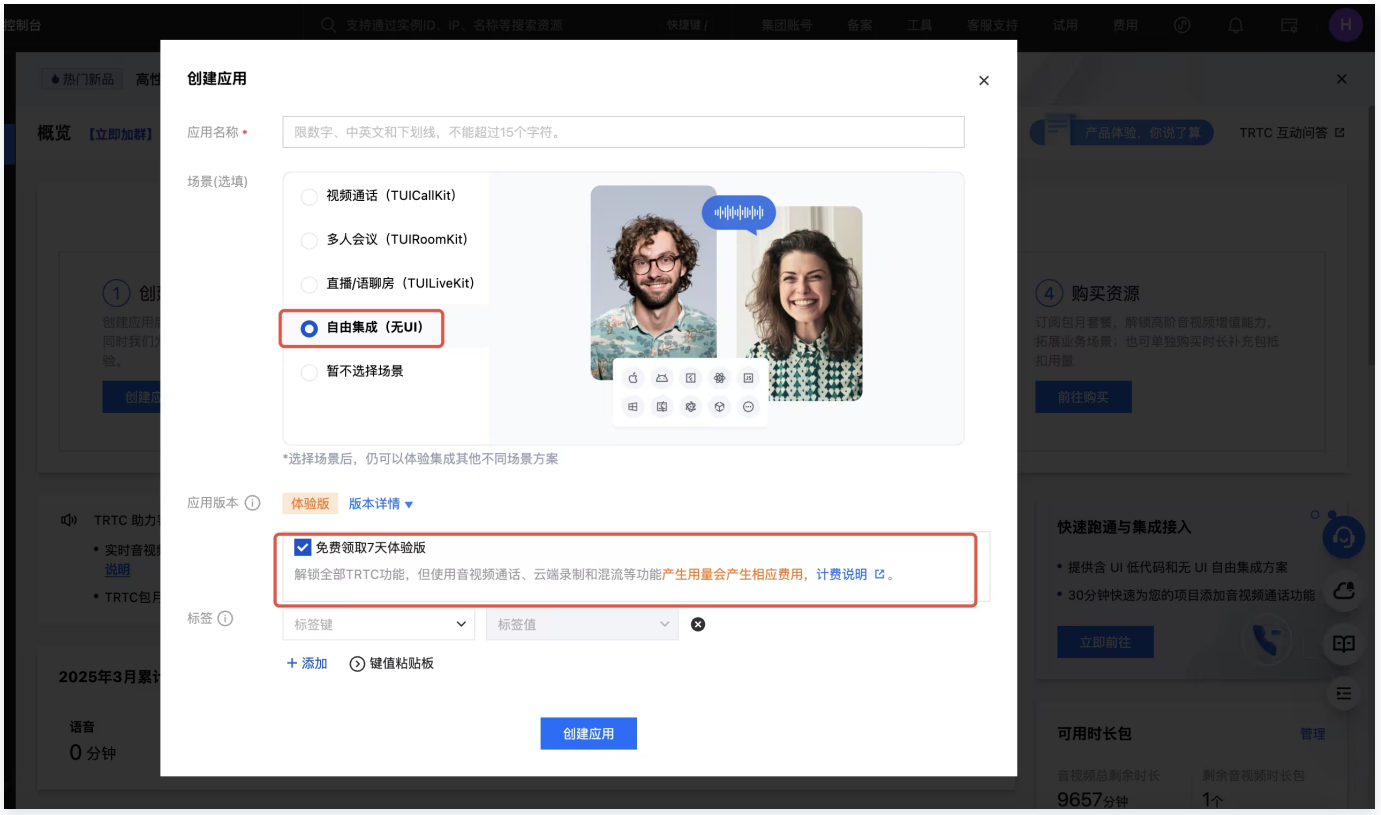
本文介绍如何 开通体验版 、开通正式版 服务，您可以根据 AI 实时对话计费说明 了解所需要的服务及费用，并参考下述指引免费开通 AI 实时对话服务。

## 开通体验版

1. 登录 实时音视频 TRTC 控制台，单击**创建应用**。




2. 在弹窗中输入您的应用名称，选择**自由集成（无 UI）**场景，勾选领取7天体验版，单击**创建应用**。




3. 进入应用管理页面，成功开通**自由集成（无 UI）**的7天试用服务。领取完成后，可以看到体验版的基本信息，包括服务状态、版本信息和功能详情、到期时间。这里的 SDKAppID 以及 SDK 密钥（SDKSecretKey）会在后续步骤中使用到。




> **注意：**
> - 领取体验版后仅开通自由集成（无 UI）7天的体验资格，测试过程中所产生的音视频时长等资源消耗（例如音频通话、实时对话服务仍然会产生后付费产品费用，具体可参见 AI 实时对话费用详情），仍会按照实时音视频 TRTC 标准计费规则计费。
> - 每个 SDKAppID 可免费体验2次，每次有效期均为7天；同时一个账号下所有 SDKAppID 的体验总次数为10次。
> - 新账号首次可前往 试用中心 免费领取10000分钟音视频时长。

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

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

## 开通正式版

AI 实时对话服务的费用 包含四个方面：音频通话费用、AI 实时对话服务费用以及语音转文字和文字转语音费用。其中，语音转文字功能可开启后付费后无门槛使用，购买 AI 智能语音资源包可享受折扣优惠，详情见 AI 智能语音计费说明，本章节将详细说明如何购买 AI 智能语音资源包。

> **说明：**
> 音频通话费用以及 AI 实时对话服务无开通过程，在使用过程中将默认开启后付费。

1. 登录 实时音视频购买页 。
2. 根据实际业务需求选择配置。接入方案选择**无 UI 自定义开发**，以及选择需要购买 AI 智能语音资源包的 SDKAppID，购买后无法修改。




3. 资源包规格选择 **AI 智能语音资源包**，并选择您需要开通的版本，AI 智能语音资源包详情可参见 AI 智能语音详细说明 。




4. 购买完成后，您可以回到 实时音视频 TRTC 控制台 ，在 时长包管理 页面查看具体的 AI 智能语音资源包的使用情况。


音视频时长补充包
录制&混流时长补充包
AI 智能语音资源包
AI 内容理解资源包
购买时长包
有效
无效
时长包ID
时长包名称
时长包时长(分钟)
剩余时长(分钟)/(百分比)
生效时间
过期时间
状态
操作
AI智能语音资源包-600,000分钟
600,000
600,000 / 100.00%
2026-10-13 10:23:45
2027-10-13 10:23:45
使用中
告警设置
抵扣明细
10 条 / 页
1
/ 1 页

# 快速开始
# 发起 AI 实时语音对话（三段式）

最近更新时间：2026-11-06 16:43:30

## 功能概述

本文介绍三段式（Cascade，`ASR → LLM → TTS`）模式的**最短接入路径**，五步跑通一轮语音对话：

**签发 `UserSig` → 客户端集成 `SDK` 进房 → 服务端创建 `AI` 对话任务 → 监听事件回调 → 结束任务**

创建任务后，AI Bot 以虚拟用户身份加入 TRTC 房间并订阅指定用户的音频流，用户开口说话即开始一轮「识别 → 推理 → 播报」的实时对话，全程可打断。

## 前提条件与配额

| 项目 | 要求 |
| --- | --- |
| TRTC 服务 | 已开通并完成实名认证。 |
| SdkAppId | TRTC 应用 ID，用于所有服务端接口调用。 |
| 房间号 | 支持数字房间号（`RoomIdType=0`）与字符串房间号（`RoomIdType=1`，推荐）。 |
| 接口频率 | 20 次/秒。 |
| 单应用并发 | 100 个任务。 |
| 空闲回收 | 默认 30 秒（`AgentConfig.MaxIdleTime` 可调，范围 5~86400 秒）。 |

> **说明：**
> 超出频率或并发上限时接口返回 `FailedOperation.RestrictedConcurrency` / `FailedOperation.RequestLimitExceeded`，请实现指数退避重试。

## 接入步骤

### 步骤1：签发 UserSig

业务后端为 App 用户和 AI Bot 各签发一份 UserSig：

- **Bot 的 UserSig**：即创建任务时 `AgentConfig.UserSig`，由业务后端保管；
- **App 用户的 UserSig**：由业务后端下发给客户端，用于进房。

UserSig 生成方式参见 TRTC UserSig 生成 。

## 步骤2：客户端集成 TRTC SDK 并进房

客户端接入 TRTC 的基本流程参见 TRTC 官网接入文档，核心代码（以 Android 为例）：

```
// 1. 创建 TRTC 实例
TRTCCloud trtc =
TRTCCloud.sharedInstance(context.getApplicationContext());

// 2. 开启音量回调与 VAD 通知，用于 UI 展示用户是否正在说话
TRTCAudioVolumeEvaluateParams volumeParams = new
TRTCAudioVolumeEvaluateParams();
volumeParams.enableVadDetection = true;
trtc.enableAudioVolumeEvaluation(true, volumeParams);

// 3. 开启音频采集（通话音质）
trtc.setSystemVolumeType(TRTCSystemVolumeTypeMedia);
trtc.startLocalAudio(TRTC_AUDIO_QUALITY_SPEECH);

// 4. 如需静音采集，可设置采集音量为 0
trtc.setAudioCaptureVolume(0);

// 5. 进房
TRTCParams params = new TRTCParams();
params.sdkAppId = <sdkAppId>;
params.userId   = <localUserId>;
params.userSig  = <userSig>;
params.roomId   = <roomId>;
params.role     = TRTCRoleAnchor;
trtc.enterRoom(params, TRTC_APP_SCENE_VIDEOCALL);
```

**进房成功条件：** `TRTCCloudListener.onEnterRoom(result > 0)` 。

## 步骤3：服务端创建 AI 对话任务

业务后端调用**创建任务**接口（完整定义见 发起 AI 对话任务 ），创建 Cascade 模式任务：

```
{
    "SdkAppId": 14000000,
    "RoomId": "12345",
```

```
    "RoomIdType": 0,
    "ConversationMode": "Cascade",
    "AgentConfig": {
        "UserId": "test2",
        "UserSig": "eJyrVgrxCdYrzkxXslJQyvEs...",
        "TargetUserId": ["Trtc_User_0"],
        "MaxIdleTime": 60,
        "SubtitleMode": 0,
        "TurnDetectionMode": "Auto",
        "TTSPushRoomParams": { "UserId": "tts-bot-001" }
    },
    "STTConfig": {
        "Provider": "trtc",
        "Params": "
{\"engine_model_type\":\"bigmodel\",\"language\":\"zh\",\"vad_silence_ti
me\":\"1000\"}"
    },
    "LLMConfig": {
        "LLMType": "openai",
        "Model": "gpt-4o-mini",
        "APIKey": "<YOUR_API_KEY>",
        "APIUrl": "https://api.openai.com/v1/chat/completions",
        "SystemPrompt": "你是一个简洁的对话助手。",
        "Timeout": 20000,
        "Temperature": 0.7
    },
    "TTSConfig": {
        "Provider": "trtc",
        "Params": "{\"Model\":\"flow_02_turbo\",\"VoiceId\":\"v-female-
R2s4N9qJ\",\"Speed\":1.0}"
    },
    "DialogConfig": {
        "InterruptMode": 0,
        "InterruptSpeechDuration": 200,
        "WelcomeMessage": "你好，我是AI助手，有什么可以帮你的？"
    }
}
```

**关键字段速览：**

| 字段 | 说明 |
| --- | --- |
| `AgentConfig.UserId` / `UserSig` | AI Bot 的房间身份与进房凭证。 |
| `AgentConfig.TargetUserId` | Bot 订阅哪些用户的音频（当前最多 1 个）。 |
| `AgentConfig.MaxIdleTime` | 房间无人说话时的空闲回收时间。 |
| `STTConfig` / `LLMConfig` / `TTSConfig` | 三段链路配置，均可更换厂商。 |
| `DialogConfig.WelcomeMessage` | Bot 进房后自动播报的欢迎语（便于验证链路）。 |
| `DialogConfig.InterruptMode` | 打断模式（0=自动 / 1=手动 / 2=关键词）。 |

接口返回 `TaskId` 即表示任务创建成功，Bot 已进房并订阅目标用户音频。配置了 `WelcomeMessage` 时，Bot 进房后会自动播报欢迎语，可直接验证 TTS 链路；随后对 Bot 说一句话，即开始一轮完整对话。

> **说明：**
> - 三段链路的详细配置说明（支持的厂商、参数约定、热更新能力）分别见 ASR 配置 、LLM 配置 、TTS 配置 。
> - `TTSPushRoomParams.UserId` 用于指定 TTS 语音播报流所使用的独立用户 ID。如果省略该参数，则默认使用 `AgentConfig.UserId` 作为播报方。使用独立 ID 可便于客户端区分 Bot 的主逻辑身份和播报身份。

## 步骤4：配置事件回调并校验签名

在控制台（或通过工单）配置业务后端的**回调 URL** 与**回调密钥**，并实现 `X-TRTC-Signature` 签名校验，即可落库 ASR / LLM 结果、AI 状态流转、任务启动 / 结束等事件。
回调报文结构、全部事件类型与签名校验方法详见 AI 对话回调事件 。

## 步骤5：结束任务

主播离线或会话结束时，业务后端调用**删除任务**接口（完整定义见 停止 AI 对话任务 ），Bot 离开房间并释放任务资源。

> **说明：**
> 也可依赖**空闲回收**：房间内持续 `MaxIdleTime` 无有效音频，任务自动结束。

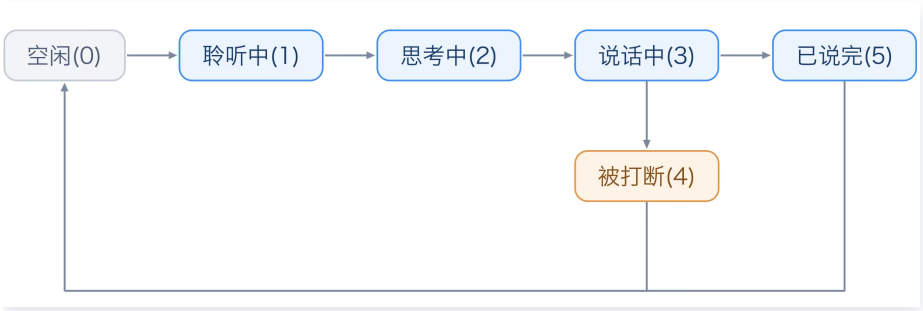
## 验证对话链路

任务运行期间向 Bot 说一句话，可通过三个通道确认链路工作正常：

| 验证通道 | 现象 | 参见 |
| --- | --- | --- |

| | | |
|---|---|---|
| 客户端听感 | Bot 自动播报欢迎语，随后对答如流，说话时可随时打断。 | （客户端）打断指令 |
| 客户端字幕 | 收到用户语音的 ASR 字幕与 AI 回复的 LLM 字幕。 | （客户端）实时字幕 |
| 服务端回调 | 依次收到 ASR 识别结果、LLM 回复结果、AI 状态流转事件。 | AI 对话回调事件 |

**一轮对话中机器人状态按以下顺序流转（完整说明见 获取 AI 机器人状态以及上下文 ）：**



# 发起 AI 实时语音对话（S2S）

最近更新时间：2026-11-06 16:43:30

## 功能概述

本文介绍端到端（S2S，Speech-to-Speech）模式的**最短接入路径**。与三段式不同，S2S 模式下用户音频**直接送入端到端语音大模型**，由模型直接返回音频与转写文本，**无需分别配置 ASR / LLM / TTS 三段**，天然低延迟。

| 对比项 | 三段式（Cascade） | 端到端（S2S） |
| --- | --- | --- |
| 配置项 | STTConfig + LLMConfig + TTSConfig | 仅 S2SConfig |
| 链路 | ASR → LLM → TTS 分段 | 音频直入、音频直出 |
| 视觉理解（截图问答） | 支持 | **不支持** |
| 对话历史查询 | 支持 | **不支持** |

两种模式的选择建议与 S2S 的详细配置说明见 S2S 配置 。三段式接入见 发起 AI 实时语音对话（三段式） 。

## 前提条件与配额

| 项目 | 要求 |
| --- | --- |
| TRTC 服务 | 已开通并完成实名认证。 |
| SdkAppId | TRTC 应用 ID，用于所有服务端接口调用。 |
| 房间号 | 支持数字房间号（ `RoomIdType=0` ）与字符串房间号（ `RoomIdType=1` ，推荐）。 |
| 接口频率 | 20 次/秒。 |
| 单应用并发 | 100 个任务。 |
| 空闲回收 | 默认 30 秒（ `AgentConfig.MaxIdleTime` 可调，范围 5-86400 秒）。 |

> **说明：**
> 超出频率或并发上限时接口返回 `FailedOperation.RestrictedConcurrency` / `FailedOperation.RequestLimitExceeded` ，请实现指数退避重试。

# 接入步骤

## 步骤1：签发 UserSig

业务后端为 App 用户和 AI Bot 各签发一份 UserSig：

- **Bot 的 UserSig**：即创建任务时 `AgentConfig.UserSig`；
- **App 用户的 UserSig**：由业务后端下发给客户端，用于进房。

UserSig 生成方式参见 TRTC UserSig 生成。

## 步骤2：客户端集成 TRTC SDK 并进房

客户端接入流程与三段式完全一致，参见 TRTC 官网接入文档，核心代码（以 Android 为例）：

```
// 1. 创建 TRTC 实例
TRTCCloud trtc =
TRTCCloud.sharedInstance(context.getApplicationContext());

// 2. 开启音量回调与 VAD 通知，用于 UI 展示用户是否正在说话
TRTCAudioVolumeEvaluateParams volumeParams = new
TRTCAudioVolumeEvaluateParams();
volumeParams.enableVadDetection = true;
trtc.enableAudioVolumeEvaluation(true, volumeParams);

// 3. 开启音频采集（通话音质）
trtc.setSystemVolumeType(TRTCSystemVolumeTypeMedia);
trtc.startLocalAudio(TRTC_AUDIO_QUALITY_SPEECH);

// 4. 进房
TRTCParams params = new TRTCParams();
params.sdkAppId = <sdkAppId>;
params.userId   = <localUserId>;
params.userSig  = <userSig>;
params.roomId   = <roomId>;
params.role     = TRTCRoleAnchor;
trtc.enterRoom(params, TRTC_APP_SCENE_VIDEOCALL);
```

**进房成功条件：** `TRTCCloudListener.onEnterRoom(result > 0)`。

| 参数 | 说明 |
| --- | --- |

| | |
|---|---|
| volumeParams.enableVadDetection | 是否开启本地人声检测。注意：仅反馈本地检测结果，且自身角色必须是主播（步骤 4 里 TRTCRoleAnchor 满足此条件）；结果通过 onUserVoiceVolume 回调带出 |
| volumeParams.interval | 代码未显式设置，走默认值。单位 ms，最小 100，≤0 关闭回调，建议 300 |
| enableAudioVolumeEvaluation(enable, params) | enable=true 开启音量回调，false 关闭；params 即上面构造的参数对象 |

> **说明：**
> S2S 模式为纯语音交互，无需开启视频预览与截图相关能力。

## 步骤3：服务端创建 AI 对话任务

业务后端调用**创建任务**接口（完整定义见 **发起 AI 对话任务**），将 `ConversationMode` 设置为 `S2S`，并通过 `S2SConfig` 配置端到端模型（**无需**再传 STTConfig / LLMConfig / TTSConfig）：

```
{
    "SdkAppId": 140000000,
    "RoomId": "ai-room-001",
    "RoomIdType": 1,
    "ConversationMode": "S2S",
    "AgentConfig": {
        "UserId": "ai-bot-001",
        "UserSig": "eJyrVgrxCdYr...",
        "TargetUserId": ["user-123"],
        "TTSPushRoomParams": { "UserId": "tts-bot-001" }
    },
    "S2SConfig": {
        "Provider": "volc_doubao_realtime",
        "Params": "{\"Model\":\"doubao-seed-realtime\",\"APIKey\":\"
<YOUR_API_KEY>\",\"SystemPrompt\":\"你是一个亲切的语音助手。
\",\"VoiceId\":\"zh_female_qingxin\",\"Language\":\"zh-CN\"}"
    }
}
```

**支持的接入方式（`Provider`）：**

| 接入方式 | 说明 |
|---|---|

| | |
|---|---|
| `volc_doubao_realtime` | 火山豆包实时语音（`doubao-seed-realtime`），国内接入。 |
| OpenAI GPT-Live | OpenAI 端到端实时语音模型，海外接入。 |
| TRTC 自研 S2S | 腾讯自研端到端语音大模型，国内 + 海外。 |
| `custom` | 自建 S2S 网关。 |

接口返回 `TaskId` 即表示任务创建成功，Bot 已进房并订阅目标用户音频，用户开口即开始端到端对话。

> **说明：**
> - S2S 模式下模型直接输出音频，无需配置 `TTSConfig`；但该音频仍需由房间内一个 Bot 身份推流播放。`TTSPushRoomParams.UserId` 用于指定播放 Bot 的身份。
> - `S2SConfig` 字段与各厂商 `Params` 参数的完整说明见 S2S 配置。

## 步骤4：配置事件回调并校验签名

在控制台（或通过工单）配置业务后端的**回调 URL** 与**回调密钥**，并实现 `X-TRTC-Signature` 签名校验。

> **S2S 模式注意：**
> 对话历史查询（`DescribeType=1`）返回为空，如需留存用户与 AI 的对话内容，请通过回调事件（ASR 识别结果 / LLM 回复结果）自行落库。详见 AI 对话 2.0 服务端回调。

## 步骤5：结束任务

会话结束时，业务后端调用**删除任务**接口（完整定义见 停止 AI 对话任务），Bot 离开房间并释放任务资源。

> **说明：**
> 也可依赖**空闲回收**：房间内持续 `MaxIdleTime` 无有效音频，任务自动结束。

## 验证对话链路

任务运行期间向 Bot 说一句话，可通过三个通道确认链路工作正常：

| 验证通道 | 现象 | 参见 |
|---|---|---|
| 客户端听感 | Bot 实时应答，延迟明显低于三段式；播报中开口可直接打断（模型原生 barge-in）。 | （客户端）打断指令 |
| 客户端字幕 | 收到用户语音的 ASR 字幕与 AI 回复的 LLM 字幕。 | （客户端）实时字幕 |
| 服务端回调 | 依次收到 ASR 识别结果、LLM 回复结果、AI 状态流转事件。 | AI 对话回调事件 |

## 注意事项

1. **模式创建后不可切换：** `ConversationMode` 不支持运行时修改，Cascade 与 S2S 互切需销毁旧任务后重建；`S2SConfig`（Provider / Params）支持热更新。
2. **不支持视觉理解：**S2S + 截图问答组合不可用，需要**边通话边看屏**场景请使用三段式 + VLM。
3. **对话记录依赖回调：**S2S 不支持查询对话历史，请在回调侧落库留存。
4. **打断策略：**端到端模型默认原生 barge-in（用户开口即打断）；如需业务侧强制打断，可通过控制任务接口下发 `Interrupt` 指令。

版权所有：腾讯云计算（北京）有限责任公司

# 基础功能
# 实时字幕（客户端）

最近更新时间：2026-10-29 21:43:33

## 功能概述

TRTC AI 实时对话 2.0 提供了显示实时字幕的能力。实时字幕通过 TRTC 的自定义消息发送，可以实现与音频对话毫秒级别的同步。
通过 实验性接口事件回调 监听 `onAIConversationEvent` 回调，在客户端上监听回调来接收实时 AI 字幕的数据。

## 字幕接收

### 连接机器人

调用前需已通过 `connect` 连接该机器人，详细接入方法见：（客户端）接收 AI 机器人状态 。

### 监听字幕

完成连接之后，字幕会通过 `onExperimentalEvent` 实时回调， `eventName` 为 `onAIConversationEvent` 。 `params.event` 为 `asrSubtitle` 时是用户说话的识别结果，为 `llmSubtitle` 时是机器人的回复文本。

## 字幕模式

机器人侧字幕的出现时机由创建任务时的 `AgentConfig.SubtitleMode` 决定。

| SubtitleMode | 说明 |
| --- | --- |
| 0（默认） | 尽快显示。LLM 文本生成后即回调，不与音频播放同步，字幕会先于声音出现。 |
| 1 | 句子级同步。按当前正在播放的句子回调已播报的完整回复文本；当前句播完、开始播下一句时，再更新文本。 |

## ASR 字幕

用户说话的识别结果。 `text` 为这一句截至当前的完整文本， `end` 为 `true` 表示该句识别完成。

```
{
  "event": "asrSubtitle",
  "robotid": "ai_bot_001",
  "data": {
```

```
    "type": 10100,
    "robotid": "ai_bot_001",
    "sender": "user_001",
    "payload": {
      "text": "你好。",
      "start_time_ms": 4446039,
      "end_time_ms": 4456039,
      "start_utc_ms": 1785396254452,
      "end_utc_ms": 1785396264452,
      "end": false,
      "roundid": "D7E2053C-6E1F-4941-9761-FEE0407016EF",
      "taskid": "task-id"
    }
  }
}
```

| 字段 | 类型 | 说明 |
| --- | --- | --- |
| event | String | 固定为 `asrSubtitle`。 |
| robotid | String | 当前连接的 AI 机器人 UserId。 |
| data.type | Number | 消息类型，ASR 字幕为 `10100`。 |
| data.robotid | String | AI 对话机器人 UserId。 |
| data.sender | String | 被识别的用户 UserId。 |
| data.payload.text | String | 该句截至当前的完整识别文本。 |
| data.payload.start_time_ms | Number | 该句语音在音频流中的起始时间，相对时间，单位毫秒。 |
| data.payload.end_time_ms | Number | 该句语音在音频流中的结束时间，相对时间，单位毫秒。 |
| data.payload.start_utc_ms | Number | 该句语音起始的 UTC 时间戳，单位毫秒。 |

| | | |
|---|---|---|
| data.payload.end_utc_ms | Number | 该句语音结束的 UTC 时间戳，单位毫秒。 |
| data.payload.end | Boolean | `false` 为中间结果，`true` 为该句识别完成。 |
| data.payload.roundid | String | 对话轮次 ID，同一轮的 ASR、LLM 字幕用它关联。 |
| data.payload.taskid | String | AI 对话任务 ID。 |

> **说明：**
> 做字幕与音频对齐时，同一路音频流内的句子直接用 `start_time_ms` 排序对齐即可；需要与业务侧绝对时间（如消息发送时间）对齐时，使用 `start_utc_ms`。

## LLM 字幕

机器人的回复文本。`text` 为本轮回复截至当前的完整文本，`end` 为 `true` 表示本轮回复结束。`roundid` 与触发本轮回复的 ASR 字幕相同。

```
{
  "event": "llmSubtitle",
  "robotid": "ai_bot_001",
  "data": {
    "type": 10101,
    "robotid": "ai_bot_001",
    "sender": "tts_bot_001",
    "payload": {
      "text": "你好，有什么可以帮你的？",
      "end": false,
      "roundid": "D7E2053C-6E1F-4941-9761-FEE0407016EF",
      "taskid": "task-id"
    }
  }
}
```

| 字段 | 类型 | 说明 |
|---|---|---|
| event | String | 固定为 `llmSubtitle`。 |

| | | |
|---|---|---|
| robotid | String | 当前连接的 AI 机器人 UserId。 |
| data.type | Number | 消息类型，LLM 字幕为 `10101` 。 |
| data.robotid | String | AI 对话机器人 UserId。 |
| data.sender | String | 对应的 TTS 机器人 UserId。 |
| data.payload.text | String | 本轮回复截至当前的完整文本。 |
| data.payload.end | Boolean | `false` 为中间结果， `true` 为本轮回复结束。 |
| data.payload.roundid | String | 对话轮次 ID，与触发本轮回复的 ASR 字幕相同。 |
| data.payload.taskid | String | AI 对话任务 ID。 |

# 打断指令（客户端）

最近更新时间：2026-10-29 21:43:33

## 功能概述

客户端可以主动打断 AI 当前的播报：立刻停止正在播放的语音，并清空尚未播放的内容。效果与服务端 `ControlVoiceConversation` 的 `Interrupt` 指令相同。

## 调用方式

### 连接机器人

> ⚠ **注意：**
> 调用前需已通过 `connect` 连接该机器人，详细接入方法见：（客户端）接收 AI 机器人状态 。

### 发起打断

调用 `TRTCCloud.callExperimentalAPI`：

```
{
  "api": "aiConversation",
  "params": {
    "command": "interrupt",
    "robotid": "ai_bot_001"
  }
}
```

`robotid` 填当前已连接的 AI 机器人 UserId。
打断成功不回调。失败时通过 `TRTCCloudListener.onExperimentalEvent` 回调，`eventName` 为 `onAIConversationEvent`，`params` 为：

```
{
  "event": "onCommand",
  "robotid": "ai_bot_001",
  "command": "interrupt",
  "errCode": 1004,
  "errMsg": "invalid state"
}
```

| errCode | 含义 |
| --- | --- |
| -2 | 参数错误，缺少 robotid。 |
| -3 | 传入的 robotid 与当前连接的机器人不一致。 |
| 1001 | 任务已结束，不再响应。 |
| 1002 | 服务端处理失败，可再次调用。 |
| 1003 | 指令参数非法。 |
| 1004 | 当前状态不允许打断。例如 AI 未在播报、任务已结束、会话尚未开始等状态。 |

## 与打断模式的关系

创建任务时，`DialogConfig.InterruptMode` 决定什么情况下由服务端自动打断。客户端 `interrupt` 和服务端 `ControlVoiceConversation` 的 `Interrupt` 都是主动强制打断，不受该模式限制。

| InterruptMode | 说明 |
| --- | --- |
| 0（默认） | 自动打断。识别到用户说话即打断当前播报，最短语音时长由 `InterruptSpeechDuration` 控制，范围 200～3000 毫秒，默认 200。 |
| 1 | 手动打断。只在收到服务端 `Interrupt` 或客户端 `interrupt` 时打断。 |
| 2 | 关键词打断。用户说出 `InterruptKeywords` 中的词（最多 128 个）时打断。 |

# 接收 AI 机器人状态（客户端）

最近更新时间：2026-10-29 21:43:33

## 功能概述

TRTC AI 实时对话 2.0 提供了丰富的状态回调功能。这些状态回调通过 TRTC 的自定义消息发送，使得在客户端上可以方便地实现丰富的状态切换，例如"聆听中"、"思考中"、“说话中”等状态。
通过 实验性接口事件回调 监听 `onAIConversationEvent` 回调，在客户端上监听回调来接收实时 AI 状态的数据。

## 状态接收

### 连接机器人

如果需要接收字幕信息，需要先通过 实验性接口（callExperimentalAPI） 调用 `connect` 连接对应的 AI 机器人。
`TRTCCloud.callExperimentalAPI`：

```
{
  "api": "aiConversation",
  "params": {
    "command": "connect",
    "robotid": "ai_bot_001"
  }
}
```

`robotid` 是通过 REST API `CreateVoiceConversation` 创建任务时填入的 `AgentConfig.UserId`。
连接结果通过 `TRTCCloudListener.onExperimentalEvent` 回调，`eventName` 为 `onAIConversationEvent`，`params` 为：

```
{
  "event": "onCommand",
  "robotid": "ai_bot_001",
  "command": "connect",
  "errCode": 0,
  "errMsg": "ok"
}
```

| errCode | 含义 |
| --- | --- |

| | |
|---|---|
| 0 | 连接成功。 |
| -2 | 参数错误，缺少 robotid。 |
| 2001 | 任务不存在或已结束。 |
| 2002 | SDK 版本过低。 |

## 监听状态

完成连接之后，机器人状态会通过 `onExperimentalEvent` 实时回调，`eventName` 为 `onAIConversationEvent`。`params.event` 为 `agentState`。

## 消息格式

```
{
  "event": "agentState",
  "robotid": "ai_bot_001",
  "data": {
    "type": 10102,
    "robotid": "ai_bot_001",
    "sender": "tts_bot_001",
    "payload": {
      "taskid": "task-id",
      "state": {
        "code": 2,
        "desc": "thinking"
      },
      "timestamp": 1751196060800
    }
  }
}
```

| 字段 | 类型 | 说明 |
|---|---|---|
| event | String | 固定为 `agentState`。 |
| robotid | String | 当前连接的 AI 机器人 UserId。 |

| | | |
|---|---|---|
| data.type | Number | 消息类型，机器人状态为 `10102` 。 |
| data.robotid | String | AI 对话机器人 UserId（ `AgentConfig.UserId` ）。 |
| data.sender | String | 实际发送方 UserId，状态消息为对应的 TTS 机器人。 |
| data.payload.taskid | String | AI 对话任务 ID。 |
| data.payload.state.code | Number | 当前状态码，见下表。 |
| data.payload.state.desc | String | 状态描述。 |
| data.payload.timestamp | Number | 状态切换的 UTC 时间戳，单位毫秒。 |

**说明：**
`data.payload`（类型：Object，说明：状态负载）与 `data.payload.state`（类型：Object，说明：状态对象）为嵌套对象。

| state.code | 含义 |
|---|---|
| 0 | 空闲 |
| 1 | 聆听中 |
| 2 | 思考中 |
| 3 | 说话中 |
| 4 | 被打断 |
| 5 | 已说完 |

# ASR 配置

最近更新时间：2026-10-29 21:43:33

## 功能概述

在 Cascade（三段式）模式下，ASR（自动语音识别）是对话链路的第一段：AI 服务端订阅房间内用户的音频流，实时将语音转写为文本，交给 LLM 进行推理。
ASR 段采用 `Provider + Params` 接入模式，支持：

- **TRTC 自研 ASR（trtc，默认）**：即开即用，无需开通第三方服务，推荐绝大多数场景直接使用。
- **第三方 ASR（例如：微软 azure）**：满足指定识别引擎或特定语种的需求。

> **说明：**
> S2S（端到端）模式下，用户音频直接送入端到端语音大模型，不存在独立的 ASR 段，无需配置 `STTConfig`。当前第三方 ASR 厂商支持微软 Azure，如需接入其他 ASR 厂商，请联系商务。

## 配置结构

在创建 AI 对话任务时，通过 `STTConfig` 指定 ASR 配置：

| 字段名 | 必选 | 类型 | 描述 |
| --- | --- | --- | --- |
| `Provider` | 是 | String | ASR 厂商标识：<br>• trtc（默认）；<br>• azure。 |
| `Params` | 是 | String | 厂商专属参数，JSON 字符串。 |

# TRTC 自研 ASR

## Params 参数说明（Provider = trtc）

| 字段名 | 类型 | 必选 | 描述 |
| --- | --- | --- | --- |
| `engine_model_type` | String | 是 | ASR 模型类型，示例值 `bigmodel`。**优先推荐 `bigmodel` 系列大模型引擎**，详见 engine_model_type 与 language 说明。 |
| `language` | String | 否 | 识别语言，取值与 `engine_model_type` 中的语言标识一致，例如 `zh` / `en`。 |

| | | | |
|---|---|---|---|
| vad_silence_time | Integer | 否 | VAD 静音判停时长（毫秒），范围 200-2000，默认 1000。调小可加快断句响应，但过小容易把句中停顿误判为说完。 |
| max_speak_time | Integer | 否 | 最长断句时长（毫秒），范围 10000-90000，默认 60000。超过该时长强制断句。 |

## engine_model_type 与 language 说明

engine_model_type 为 ASR 使用的模型，语音转文本不同套餐版本支持的引擎如下：

| 引擎值 | 说明 |
|---|---|
| bigmodel | 大模型引擎，**推荐使用**。 |
| 16k_zh_en | 传统引擎, 16k 中英模型引擎，同时支持中文、英语、多种中文方言口音的识别。 |

**language 支持的语言**，可填写：

| 语言标识 | 语言 | 语言标识 | 语言 |
|---|---|---|---|
| zh | 中文 | tr | 土耳其语 |
| en | 英语 | hi | 印地语 |
| yue | 粤语 | ms | 马来语 |
| ar | 阿拉伯语 | nl | 荷兰语 |
| de | 德语 | sv | 瑞典语 |
| fr | 法语 | da | 丹麦语 |
| es | 西班牙语 | fi | 芬兰语 |
| pt | 葡萄牙语 | pl | 波兰语 |
| id | 印度尼西亚语 | cs | 捷克语 |
| it | 意大利语 | fil | 菲律宾语 |
| ko | 韩语 | fa | 波斯语 |
| ru | 俄语 | el | 希腊语 |

| | | | |
|---|---|---|---|
| th | 泰语 | hu | 匈牙利语 |
| vi | 越南语 | mk | 马其顿语 |
| ja | 日语 | ro | 罗马尼亚语 |

**说明：**
language 为识别语言，取值使用上述语言标识（如 zh / en），需与 engine_model_type 中指定的语言保持一致。对应引擎可选语言可参考 接入文档 。

### 识别效果

在开源测试集上，TRTC 自研 ASR 中英文识别**字准均 ≥96%，首包耗时约 400ms**：

| 测试集 | 语种 / 场景 | 字准 |
|---|---|---|
| aishell | 中文普通话 | ≥96% |
| test_meeting | 中文会议 | ≥96% |
| test_net | 中文网络音频 | ≥96% |
| librispeech_clean | 英文（清晰朗读） | ≥96% |
| librispeech_other | 英文（其他口音/噪声） | ≥96% |

### 领域热词

内置领域热词，覆盖**地图 POI、影视娱乐、医疗、AI 科技、汽车、游戏、零售**等领域，可显著提升专有名词的识别准确率。

## 第三方 ASR 接入

将 Provider 设置为对应厂商标识，并在 Params 中传入该厂商的专属参数（鉴权、模型、语种等），参数规范以 对应厂商文档 为准。
目前支持以下字段设置：

| 字段名 | 含义 |
|---|---|
| language | 语言。 |
| subscription_key | API key，从账号获取。 |
| endpoint | 接入点，从账号获取。 |

| | |
|---|---|
| profanity_mode | 脏词过滤，支持设置 none/masked/removed/raw。 |
| endpoint_id | 自定义模型 endpoint ID（用于定制化语音模型），为空则使用默认模型。 |

```
{
    "STTConfig": {
        "Provider": "azure",
        "Params": "{\"provider\":\"azure\",\"language\":\"zh-
CN\",\"subscription_key\":\"p7o9fD7xEIszb***BLf6\", \"endpoint\":
\"https://southeastasia.api.cognitive.microsoft.com\"}"
    }
}
```

## 配置示例

以 TRTC 自研 ASR 为例，创建任务时传入（推荐使用 `bigmodel`）：

```
{
    "STTConfig": {
        "Provider": "trtc",
        "Params": "
{\"engine_model_type\":\"bigmodel\",\"language\":\"zh\",\"vad_silence_ti
me\":1000}"
    }
}
```

## 注意事项

1. **ASR 配置不支持热更新**：任务创建后无法通过修改任务接口变更 `STTConfig`；如需更换识别引擎或参数，请结束当前任务并重新创建。
2. **仅 Cascade 模式需要配置**：S2S 模式由端到端模型完成语音理解，无需 `STTConfig`。
3. **识别结果获取方式**：
   - **服务端**：一句话识别完成时，通过 ASR 识别结果回调收到该句终态文本（含起止时间戳与轮次 ID），详细参见：AI 对话回调事件。
   - **客户端**：通过 SDK 的字幕事件实时获取识别文本，详细参见：（客户端）实时字幕。

# LLM 配置

最近更新时间：2026-10-29 21:43:33

## 功能概述

在 **Cascade（三段式）**模式下，LLM（大语言模型）是对话链路的第二段，也是 AI 机器人的“大脑”：接收 ASR 转写出的用户文本进行推理，流式输出回复文本，交给 TTS 合成播报。
LLM 段按 **OpenAI 兼容协议**统一对接，不限制厂商：

- 支持 **运行时热更新**，可在对话进行中切换模型、调整提示词。
- 支持 **MCP 工具调用（Function Calling）**，可接入 RAG、业务查询等复杂场景。

**支持的 LLMType：**

| LLMType | 说明 |
| --- | --- |
| `openai`（推荐） | OpenAI 兼容协议，可对接 GPT 系列、DeepSeek、阿里千问等。 |
| `hunyuan` | 腾讯混元。 |
| `deepseek` | DeepSeek。 |
| `custom` | 自建 LLM 网关（需符合 OpenAI 兼容协议）。 |

> **说明：**
> **S2S（端到端）模式**下语音理解与生成由端到端模型完成，不存在独立的 LLM 段，无需配置 `LLMConfig`。

## 配置结构

创建 AI 对话任务时，通过 `LLMConfig` 指定 LLM 配置。`LLMConfig` 采用**平铺字段**结构，厂商扩展参数通过 `CustomFields` / `CustomHeaders` 透传：

| 字段名 | 必选 | 类型 | 描述 |
| --- | --- | --- | --- |
| `LLMType` | 是 | String | LLM 接入类型：`openai`（推荐）/ `hunyuan` / `deepseek` / `custom`。 |
| `Model` | 是 | String | 模型名（例如 `gpt-4o-mini` / `hunyuan-lite`）。 |
| `APIKey` | 否 | String | 厂商 API Key。<br>> **说明：** |

| | | | |
|---|---|---|---|
| | | | 条件必填，配置测试账号时可省略，生产环境必须传入。 |
| APIUrl | 否 | String | OpenAI 兼容 endpoint。<br>**说明：**<br>条件必填，配置测试账号时可省略，生产环境必须传入。 |
| SystemPrompt | 否 | String | 系统提示词，长度 ≤ 5000。 |
| FailureMessage | 否 | String | LLM 失败后的兜底 TTS 文本，避免对话断档。 |
| MaxTokens | 否 | Integer | 单次回复最大 Token，默认 1024，0=不限制。 |
| ThinkingMode | 否 | Boolean | 是否开启思考模式，默认 false。 |
| MaxHistory | 否 | Integer | 缓存对话轮次，范围 0-1024，默认 50，建议 10-30。 |
| MaxToolRound | 否 | Integer | MCP 工具调用最大轮次，范围 0-50，默认 10。 |
| Timeout | 否 | Integer | 请求超时（毫秒），范围 1000-60000，默认 10000。 |
| Temperature | 否 | Float | 采样温度 0.0-2.0，默认 1.0。值越大输出越随机，越小越确定。 |
| TopP | 否 | Float | 采样概率 0.0-1.0，默认 1.0。值越小结果确定性越强。 |
| McpServer | 否 | Object | MCP 工具定义，见 McpServer 定义 。 |
| CustomHeaders | 否 | String | JSON 字符串，自定义 HTTP 请求头。 |
| CustomFields | 否 | String | JSON 字符串，厂商扩展参数黑盒透传。 |
| HistoryMessages | 否 | Array of Object | 初始历史消息，每项含 Role （ user / assistant / system ）/ Content。 |

# LLM 厂商接入示例

## OpenAI 兼容协议（推荐）

适用于 GPT 系列，以及 DeepSeek、阿里千问等任何提供 OpenAI 兼容接口的模型：

```
"LLMConfig": {
    "LLMType": "openai",
    "Model": "gpt-4o-mini",
    "APIKey": "<your-api-key>",
    "APIUrl": "https://api.openai.com/v1/chat/completions",
    "SystemPrompt": "你是一个简洁的对话助手。",
    "Timeout": 20000,
    "Temperature": 0.7
}
```

## 腾讯混元

```
"LLMConfig": {
    "LLMType": "hunyuan",
    "Model": "hunyuan-lite",
    "APIKey": "<your-api-key>",
    "APIUrl": "<混元 OpenAI 兼容 endpoint，以官方文档为准>",
    "SystemPrompt": "你是一个简洁的对话助手。",
    "Temperature": 0.7
}
```

## DeepSeek

```
"LLMConfig": {
    "LLMType": "deepseek",
    "Model": "deepseek-chat",
    "APIKey": "<your-api-key>",
    "APIUrl": "<DeepSeek OpenAI 兼容 endpoint，以官方文档为准>",
    "SystemPrompt": "你是一个简洁的对话助手。",
    "Temperature": 0.7
}
```

> **说明：**

> DeepSeek、阿里千问等厂商同样兼容 OpenAI 协议，也可直接使用 `LLMType=openai` 接入。

## 自定义 LLM 网关（custom）

适用于自建 LLM 服务、私有化部署模型，或需要在网关层实现 RAG 检索、联网搜索、多模型编排后再整合输出的场景：

1. 实现一个符合 OpenAI Chat Completions 协议的 HTTP 接口，作为 `APIUrl` 接入。
2. 网关内部可自由实现业务逻辑（查询知识库、调用子模型等），只需最终按协议返回流式响应。
3. 通过 `CustomHeaders` 传入网关鉴权所需的自定义请求头。

```
"LLMConfig": {
    "LLMType": "custom",
    "Model": "your-model",
    "APIKey": "<your-gateway-key>",
    "APIUrl": "https://your-custom-llm-gateway/v1/chat/completions",
    "SystemPrompt": "请根据知识库内容用友好的语气回答用户问题。",
    "Temperature": 0.7
}
```

# 上下文管理

## 初始历史消息（HistoryMessages）

任务创建时可通过 `HistoryMessages` 注入初始上下文（例如用户画像、此前的对话摘要），AI 机器人首轮对话即可“带着记忆”开始：

```
"HistoryMessages": [
    { "Role": "user", "Content": "我叫小王，我是一名程序员。" },
    { "Role": "assistant", "Content": "好的，很高兴认识你，小王！" }
]
```

## 对话轮次缓存（MaxHistory）

后台自动缓存最近 `MaxHistory` 轮对话作为上下文：

- 范围 0–1024，默认 50，建议 0–30。
- 取值过大：Token 消耗增加、推理延迟上升；取值过小：多轮对话中丢失前文记忆。

## 主动注入文本

除语音输入外，还可以通过控制接口的文本注入指令，在运行时向对话上下文插入文本并触发一轮 LLM 回复，详细参见 主动送文本给 LLM 或者 TTS 。

## 输出控制

| 手段 | 说明 |
| --- | --- |
| `MaxTokens` | 限制单次回复长度，控制时延与成本；0=不限制。 |
| `FailureMessage` | LLM 调用失败时播报的兜底文本，避免对话断档。 |
| `FilterBracketsContent` | 过滤 LLM 返回中的括号内容，避免动作、表情等标记被 TTS 念出。 |

`FilterBracketsContent` 在 `AgentConfig` 中配置：

| 值 | 过滤内容 |
| --- | --- |
| `1` | 中文括号（） |
| `2` | 英文括号() |
| `3` | 中文方括号【】 |
| `4` | 英文方括号[] |
| `5` | 英文花括号{} |

> **说明：**
> 最多可选 5 个。典型用法：LLM 人设类模型常以 `（微笑）`、`[开心]` 标注动作情绪，配置后可避免 TTS 将其朗读出来。

# Function Calling / MCP 工具

`McpServer` 支持配置 MCP（Model Context Protocol）工具服务，适合需要业务侧 Function Calling、RAG 检索的复杂对话场景：

```
"McpServer": {
    "Name": "weather-tools",
    "Url": "https://your-mcp-server/sse",
    "Timeout": 5000,
    "Headers": "{\"Authorization\":\"Bearer xxx\"}",
    "ToolList": ["get_weather", "get_forecast"]
}
```

| 字段名 | 类型 | 描述 |
| --- | --- | --- |
| `Name` | String | MCP 唯一标识符。 |
| `Url` | String | 接入地址。 |
| `Timeout` | Integer | 请求超时（毫秒），默认 5000。 |
| `Headers` | String | JSON 字符串，自定义请求头（鉴权等）。 |
| `ToolList` | Array of String | 工具使用白名单。 |

工具调用最大轮次由 `MaxToolRound` 控制（范围 0–50，默认 10）。

## 运行时热更新

`LLMConfig` **全部字段支持热更新**——任务运行中可通过修改任务接口更新，包括：

- 运行时切换模型（ `Model` / `LLMType` ）
- 调整系统提示词（ `SystemPrompt` ）、采样参数（ `Temperature` / `TopP` ）
- 更新上下文缓存轮次（ `MaxHistory` ）

> ⚠ **注意：**
> 更新请求需携带 `SequenceNumber`（单调递增序号），后台保证按序处理；序号非递增的更新将被拒绝。这与 ASR / TTS / Vision 配置不同（三者均不支持热更新，需重建任务）。

## 注意事项

1. 生产环境必须传入 `APIKey` / `APIUrl`，否则 LLM 调用将失败。
2. **VLM 视觉理解的模型要求：**启用视觉理解时， `Model` 必须是多模态模型（例如 `hunyuan-vision` / `gpt-4o` / `gpt-4o-mini` / `qwen-vl-plus` ），详细参见 视觉理解能力（截图问答）。
3. **回复获取方式：**
   - **服务端：**LLM 单轮回复完成时，通过 LLM 回复回调收到该轮终态完整文本（含轮次 ID），详细参见 AI 对话回调事件。
   - **客户端：**通过 SDK 的字幕事件实时获取 LLM 回复文本，详细参见 （客户端）实时字幕。
4. **超时设置：** `Timeout` 默认 10000ms；开启思考模式或模型推理较慢时，可适当调大，避免超时导致回复中断。
5. **成本控制：** `MaxTokens` 与 `MaxHistory` 直接影响 Token 消耗，建议按业务实际需求设置。

# TTS 配置

最近更新时间：2026-10-29 21:43:33

## 功能概述

在 Cascade（三段式）模式下，TTS（文本转语音）是对话链路的第三段，也是 AI 机器人的“嗓音”：接收 LLM 流式输出的回复文本，实时合成语音并回传到 TRTC 房间播报，同时支持被打断。
TTS 段采用 `Provider + Params` 接入模式，支持：

- **TRTC 自研 TTS（`trtc`，默认）**：即开即用，无需开通第三方服务，推荐绝大多数场景直接使用。
- **第三方 TTS（`minimax` / `azure`）**：满足指定音色、特定厂商音质的需求。
- **自定义 TTS（`custom`）**：自研 TTS、私有化部署 TTS，基于 OpenAI TTS 标准协议对接。

> **说明：**
> S2S（端到端）模式下，音频由端到端模型直接生成，不存在独立的 TTS 段，无需配置 `TTSConfig`。

## 配置结构

创建 AI 对话任务时，通过 `TTSConfig` 指定 TTS 配置：

| 字段名 | 必选 | 类型 | 描述 |
| --- | --- | --- | --- |
| `Provider` | 是 | String | TTS 厂商标识：`trtc`（默认）/ `minimax` / `azure` / `custom`。 |
| `Params` | 是 | String | 厂商专属参数，JSON 字符串。 |

# TRTC 自研 TTS

## Params 参数说明（Provider = trtc）

| 字段名 | 类型 | 必选 | 描述 |
| --- | --- | --- | --- |
| `Model` | String | 是 | 音色模型（TTS 模型参数，仅支持填写"flow_02_turbo"）。 |
| `VoiceId` | String | 是 | 音色 ID。对于不同的语言，需要填写特定的音色 ID，详细参见 VoiceId 音色说明。 |
| `Speed` | Float | 否 | 语速，默认 1.0。 |

| | | | |
|---|---|---|---|
| Volume | Float | 否 | 音量，范围 0.0–2.0，默认 1.0。 |
| Pitch | Float | 否 | 音高（半音），默认 0.0。 |

## VoiceId 音色说明

VoiceId 为音色 ID，不同语言需填写对应的音色 ID，支持的音色如下：

| 语言 | 男生音色 | 女生音色 |
|---|---|---|
| 中文（zh） | v-male-s5NqE0rZ | v-female-R2s4N9qJ |
| 英语（en） | v-male-X6h4TvP9 | v-female-P6q9LmR2 |
| 日语（ja） | v-male-J3n8DxK2 | v-female-J3k7NxR2 |
| 粤语（yue） | v-male-D7p4XcL2 | v-female-C5t1QxH9 |
| 印度尼西亚语（id） | v-male-B5f2CnS3 | v-female-W5n8HgB2 |
| 泰语（th） | v-male-B5f2CnS3 | v-female-W5n8HgB2 |
| 越南语（vi） | v-male-B5f2CnS3 | v-female-W5n8HgB2 |
| 马来语（ms） | v-male-Q6p8ZxL3 | v-female-C8k4NxL6 |

> **说明：**
> 印度尼西亚语（id）、泰语（th）、越南语（vi）共用同一组音色。

## 配置示例

```
"TTSConfig": {
    "Provider": "trtc",
    "Params": "{\"Model\":\"flow_02_turbo\",\"VoiceId\":\"v-female-R2s4N9qJ\",\"Speed\":1.0}"
}
```

## 第三方 TTS 接入

将 Provider 设置为对应厂商标识（minimax / azure），并在 Params 中传入该厂商的专属参数（鉴权、模型、音色、语速等），参数规范以对应厂商文档为准。如需接入其他 TTS 厂商，请联系商务。

## Minimax 接入示例（Provider = minimax）

点击查看 Minimax 官网文档 。
**配置示例：**

```
"TTSConfig": {
    "Provider": "minimax",
    "Params": "{\"provider\":\"minimax\",\"api_key\":\"<your-api-key>\",\"model\":\"speech-2.6-hd\",\"url\":\"https://api.minimaxi.com/v1/t2a_v2\",\"voice_setting\":{\"voice_id\":\"male-qn-qingse\"}}"
}
```

## Azure 接入示例（Provider = azure）

`Params` 字段说明：

| 字段名 | 含义 |
| --- | --- |
| subscription_key | 密钥。 |
| url | API 地址。 |
| voice | 音色 ID。 |
| lang | 语言。 |
| rate | 语速。 |
| volume | 音量。 |
| pitch | 语调。 |
| style | 情感风格。 |

点击查看 Microsoft 官网 。
**配置示例：**

```
"TTSConfig": {
    "Provider": "azure",
    "Params": "{\"voice\":\"zh-CN-XiaoxiaoNeural\",\"subscription_key\":\"<your-subscription-key>\",\"url\":\"https://southeastasia.api.cognitive.microsoft.com\"}"
```

```
}
```

# 自定义 TTS 接入（Provider = custom）

**客户可提供自研 TTS、私有化部署 TTS 或第三方 TTS 服务作为 AI 对话的语音合成模块。接口协议基于 OpenAI TTS 标准协议构建，并针对实时对话场景做了必要扩展（如 PCM 流式输出、采样率/声道控制）。**

## 协议规范

| 项目 | 要求 |
| --- | --- |
| 请求方式 | HTTP POST |
| 请求路径 | `/v1/audio/speech`（OpenAI TTS 标准路径） |
| 请求格式 | JSON Body |
| 鉴权 | `Authorization: Bearer {api_key}`（`api_key` 由 Params 透传） |
| 响应格式 | HTTP 200 + PCM 音频二进制流（Chunked Transfer Encoding） |
| 音频格式 | 有符号 16-bit Little-Endian PCM |

## Params 参数说明（Provider = custom）

| 字段名 | 类型 | 必选 | 描述 |
| --- | --- | --- | --- |
| `url` | String | 是 | TTS 服务 HTTP 接入点。 |
| `api_key` | String | 否 | 鉴权密钥（映射为 `Authorization: Bearer`）。 |
| `model` | String | 是 | 模型标识（如 `cosyvoice-v1`）。 |
| `voice` | String | 否 | 音色 ID。 |
| `speed` | Float | 否 | 语速，范围 0.25-4.0，默认 1.0。 |
| `sample_rate` | Integer | 否 | 采样率：8000 / 16000（默认）/ 24000 / 48000。 |

| | | | |
|---|---|---|---|
| `channel` | Integer | 否 | 声道：<br>• 1（默认，单声道）；<br>• 2（双声道）。 |
| `custom_headers` | Object | 否 | 自定义请求头。 |
| `custom_fields` | Object | 否 | 厂商扩展参数黑盒透传。 |

## 配置示例

```
"TTSConfig": {
    "Provider": "custom",
    "Params": "{\"url\":\"https://your-tts-service/v1/audio/speech\",\"api_key\":\"<your-api-key>\",\"model\":\"cosyvoice-v1\",\"voice\":\"your-voice-id\",\"speed\":1.0,\"sample_rate\":16000,\"channel\":1}"
}
```

## 接入要点

1. TRTC 侧对 `Params` 做 **JSON 合法性校验但不解析内部字段语义**，具体字段映射关系由 AI 后台的自定义 TTS 适配器实现。
2. 建议业务侧 TTS 服务做好**流式输出**，避免首包延迟过大影响对话体验。
3. 自定义 TTS 服务需按协议返回 HTTP 200 + PCM 二进制流（Chunked），非 200 响应视为合成失败。

# 音频回传房间

TTS 合成的音频由后台通过**回推机器人**推入 TRTC 房间，回推机器人通过 `AgentConfig.TTSPushRoomParams` 指定：

| 字段名 | 必选 | 类型 | 描述 |
|---|---|---|---|
| `UserId` | 是 | String | 回推机器人的 UserId，长度 1-120（UserSig 由后台自动生成）。 |

```
"AgentConfig": {
    "UserId": "ai-bot-001",
    "TTSPushRoomParams": { "UserId": "tts-bot-001" }
```

```
}
```

> **说明：**
> 对话机器人（`AgentConfig.UserId`）与回推机器人（`TTSPushRoomParams.UserId`）需使用不同的用户 ID，客户端订阅回推机器人的音频流即可听到 AI 的回复。

## 播报内容控制

除 LLM 回复文本外，以下文本同样经 TTS 合成播报：

| 内容 | 配置位置 | 说明 |
| --- | --- | --- |
| **欢迎语** | `DialogConfig.WelcomeMessage` | 主播进房后自动播报。 |
| **静音追问** | `DialogConfig.SilenceFollowUpText` | 静音超时后 Bot 主动追问。 |
| **LLM 失败兜底** | `LLMConfig.FailureMessage` | LLM 调用失败时播报，避免对话断档。 |
| **括号内容过滤** | `AgentConfig.FilterBracketsContent` | 过滤 `（微笑）`、`[开心]` 等动作情绪标记，避免被 TTS 念出，详细参见 LLM 配置 > 输出控制 。 |

## 注意事项

1. **TTS 配置不支持热更新**：任务创建后无法通过修改任务接口变更 `TTSConfig`；如需更换音色或厂商，请结束当前任务并重新创建。
2. **仅 Cascade 模式需要配置**：S2S 模式由端到端模型直接生成音频，无需 `TTSConfig`。
3. **音色与语言需匹配**：`VoiceId` 需填写与对话语言对应的音色 ID，语言与音色不匹配会导致合成失败或发音异常。
4. **打断配合**：TTS 播报中收到打断指令（自动 / 手动 / 关键词打断）会立即停止播放并清空待播队列，详细参见（客户端）打断指令 。

# S2S 配置

最近更新时间：2026-10-29 21:43:33

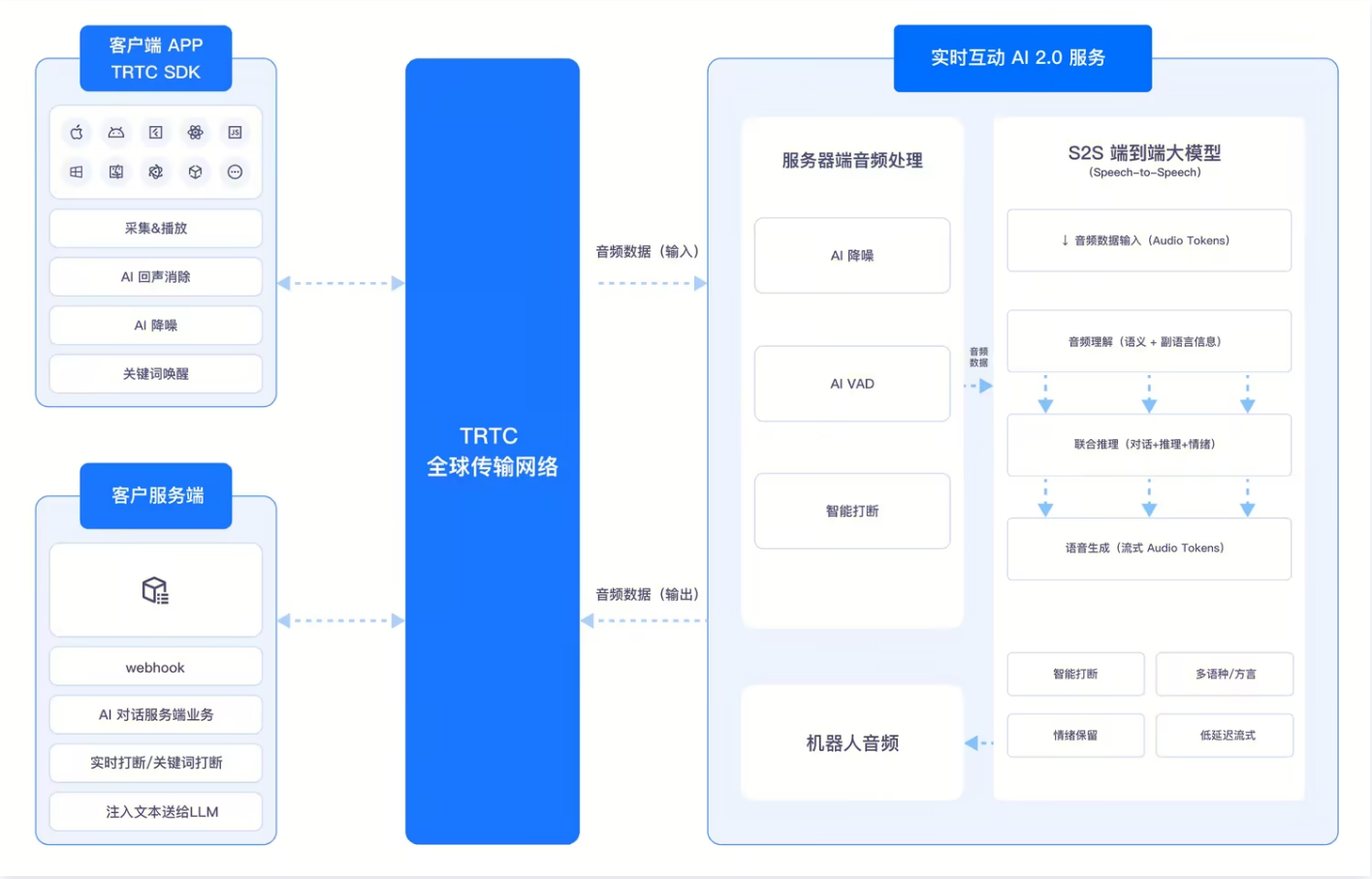
## 功能概述

S2S（Speech-to-Speech，端到端）模式下，用户音频**直接送入端到端语音大模型**，模型直接返回音频与转写文本，**不再经过独立的 ASR / LLM / TTS 三段**。
与三段式（Cascade）模式的对比：

| 对比项 | 三段式（Cascade） | 端到端（S2S） |
| --- | --- | --- |
| **链路结构** | ASR → LLM → TTS 分段式 | 音频直入、音频直出的端到端语音大模型 |
| **配置要求** | 需分别配置 `STTConfig` / `LLMConfig` / `TTSConfig` | 仅需配置 `S2SConfig` |
| **延迟特性** | 三段累加 | 天然低延迟 |
| **视觉理解（截图问答）** | 支持（需多模态模型） | **暂不支持** |
| **对话上下文查询** | 支持（查询接口 `DescribeType=1`） | **暂不支持**，返回为空 |
| **模型能力定制** | 各段可独立更换厂商 / 模型 | 由端到端模型整体决定 |

两种模式按需选择：追求极低延迟、纯语音交互选 S2S；需要视觉理解、深度定制各段能力、上下文管理选三段式。

## 架构图




## 配置入口

**创建任务时将** `ConversationMode` **设置为** `S2S`**，并通过** `S2SConfig` **配置端到端模型；此时无需再传** `STTConfig` / `LLMConfig` / `TTSConfig`**：**

```
{
    "ConversationMode": "S2S",
    "AgentConfig": {
        "UserId": "ai-bot-001",
        "UserSig": "eJyrVgrxCdYr...",
        "TargetUserId": ["user-123"],
        "TTSPushRoomParams": { "UserId": "tts-bot-001" }
    },
    "S2SConfig": {
        "Provider": "volc_doubao_realtime",
        "Params": "{\"Model\":\"doubao-seed-realtime\",\"APIKey\":\"volc-ak-xxx\",\"SystemPrompt\":\"你是一个亲切的语音助手。\",\"VoiceId\":\"zh_female_qingxin\",\"Language\":\"zh-CN\"}"
    }
```

```
}
```

> **说明：**
> `AgentConfig` 的说明（Bot 身份、空闲回收、字幕模式等）与三段式一致，详细参见 **快速开始 > 端到端 S2S 接入**。

## 支持的接入方式

| 接入方式 | 说明 |
| --- | --- |
| `volc_doubao_realtime` | 火山豆包实时语音（`doubao-seed-realtime`），国内接入。 |
| OpenAI GPT-Live | OpenAI 端到端实时语音模型，海外接入。 |

## S2SConfig 字段说明

| 字段名 | 必选 | 类型 | 描述 |
| --- | --- | --- | --- |
| `Provider` | 是 | String | S2S 端到端厂商标识（如 `volc_doubao_realtime` / `custom`）。 |
| `Params` | 否 | String | 厂商专属参数，JSON 字符串，内部字段由各厂商适配器解析。 |

## Params 字段约定（Provider = volc_doubao_realtime）

| 字段名 | 类型 | 必选 | 描述 |
| --- | --- | --- | --- |
| `Model` | String | 是 | 模型名（如 `doubao-seed-realtime`）。 |
| `APIKey` | String | 否 | 火山 API Key。 |
| `SystemPrompt` | String | 否 | 系统提示词，长度 ≤ 8000。 |
| `VoiceId` | String | 否 | 输出音色 ID。 |
| `Language` | String | 否 | 主语种（BCP-47 格式，如 `zh-CN`）。 |

| | | | |
|---|---|---|---|
| `Timeout` | Integer | 否 | 建连超时（毫秒），默认 15000。 |

## 打断策略

S2S 模式下的打断由两层配合：

| 打断方式 | 说明 |
|---|---|
| **模型侧原生打断（barge-in）** | 端到端模型自带打断能力，用户开口即可打断当前播报，无需额外配置。 |
| **服务端强制打断** | 业务侧可通过控制任务接口下发 `Interrupt` 指令，立即停止当前播报并清空待播队列。 |

服务端强制打断的触发方式详细参见 **（客户端）打断指令** 。

## 运行时更新

`S2SConfig` **支持热更新（** `Provider` / `Params` **），运行中可通过更新任务接口切换模型或调整参数；但不允许跨模式切换，**Cascade 与 S2S 之间互切需要**重建任务**。
各配置项的热更新支持情况：

| 配置 | 可更新 | 说明 |
|---|---|---|
| `S2SConfig` | Provider / Params | 运行时切换端到端模型。 |
| `STTConfig` / `TTSConfig` / `VisionConfig` | \ | 需重建任务。 |

## 注意事项

1. **不支持视觉理解**：S2S + VLM 组合不可用，截图问答仅在 Cascade 模式下支持，详细参见 **视觉理解能力（截图问答）** 。
2. **不支持查询对话历史**：S2S 模式下查询接口的 `DescribeType=1` 返回为空，如需对话记录请通过三段式回调（ASR / LLM 结果事件）自行落库，详细参见 **AI 对话回调事件** 。
3. **模式互斥**：`ConversationMode` 创建后不可修改，跨模式需销毁旧任务后重新创建。
4. **Params 为 JSON 字符串**：传入前确保 JSON 合法，TRTC 侧仅做格式校验，内部字段语义由厂商适配器解析。
5. **密钥安全**：`APIKey` 等鉴权信息通过服务端接口传入，不要在客户端保存或下发。

# 主动送文本给 LLM 或者 TTS

最近更新时间：2026-10-29 21:43:33

## 功能概述

大语言模型（LLM）本身并不支持主动输出，需要业务侧基于一定规则**主动触发 AI 说话**，从而提升实时互动中的沉浸感。例如：用户长时间没有说话，让 AI 主动追问一句；业务侧检测到订单发货、时间到点等关键事件时，让 AI 立刻向用户播报提醒。
TRTC AI 实时对话提供三类 **AI 主动说话**能力：

| 能力 | 实现方式 | 特点 |
| --- | --- | --- |
| **主动调用 LLM** | 任务运行中，通过控制任务接口下发文本注入指令（`InvokeLLM`）。 | 注入文本进入对话上下文，触发一轮 LLM 回复；回复结合上下文动态生成，经 TTS 合成播报。 |
| **文本直送 TTS** | 任务运行中，通过控制任务接口下发文本播报指令（`InvokeTTS`）。 | 文本不经过 LLM 推理，直接由 TTS 合成播报，AI 说出的内容与业务下发的一字不差。 |
| **固定话术播报** | 创建任务时通过 `DialogConfig` 配置欢迎语、静音追问。 | 直接播报固定文本，由后台在对应时机自动触发，无需业务侧运行时干预。 |

> **说明：**
> 上述文本注入能力面向 **Cascade（三段式）**模式；**S2S（端到端）**模式下语音理解与生成由端到端模型完成，不存在独立的 LLM / TTS 段。

# 主动调用 LLM（文本注入）

## 使用方式

任务运行期间，业务后端通过**控制任务接口**（任务生命周期五类标准操作之一）下发 `Command=InvokeLLM` 指令：

- 注入的文本**进入当前对话上下文**，触发一轮 LLM 回复；
- LLM 基于上下文（含注入文本）生成回复，回复文本经 TTS 合成后推入房间播报；
- 回复内容可通过客户端**字幕事件**与服务端 **LLM 回复回调**获取（见下文）。

该接口同时支持强制打断指令（`Interrupt`）与文本直送 TTS 指令（`InvokeTTS`），三者共用同一接口，详细参见下文及 （客户端）打断指令 。

## 输入参数

| 参数名 | 必选 | 类型 | 描述 |
| --- | --- | --- | --- |
| `SdkAppId` | 是 | Integer | TRTC 应用 ID。 |
| `TaskId` | 是 | String | 任务 ID。 |
| `Command` | 是 | String | 指令名：`InvokeLLM`。 |
| `InvokeLLM` | 条件必填 | Object | `Command=InvokeLLM` 时必填，字段见下表。 |

**InvokeLLM 对象：**

| 字段名 | 必选 | 类型 | 描述 |
| --- | --- | --- | --- |
| `Content` | 是 | String | 注入的文本内容，长度 ≤ 1000。 |
| `Priority` | 否 | Integer | 优先级，决定注入文本与当前交互的冲突处理策略：<br>• `1`=高优先级；<br>• `2`=中优先级；<br>• `3`=低优先级；<br>各个档的具体行为详细参见 Priority 优先级说明。 |

## Priority 优先级说明

`InvokeLLM` 与 `InvokeTTS` 共用同一套 `Priority` 优先级定义，用于处理传入文本与 AI 当前正在进行的交互之间的冲突：

| Priority | 优先级 | 行为 |
| --- | --- | --- |
| `1` | 高优先级 | **立即终止当前 AI 的交互**（例如正在进行的播报），并立即执行传入的指令。对于 `InvokeTTS`，会立即播报指定文本；对于 `InvokeLLM`，会立即将文本注入 `LLM` 并触发新一轮回复的生成与播报。 |
| `2` | 中优先级 | **等当前交互结束后**，再播放传入的文本内容。 |
| `3` | 低优先级 | 若**正在交互则直接丢弃**传入文本；**未在交互**则播放传入的文本内容。 |

> **选型建议：**
> 紧急提醒（例如风控告警、审核拦截提示）用高优先级即时插播；一般性事件通知用中优先级排队播报；可有可无的背景信息用低优先级，宁可不播也不打断当前对话。

### 请求示例

```
{
    "SdkAppId": 1400000000,
    "TaskId": "-nG0jIFU7q3wqZnqm8xB5TY0n43U5rLb...",
    "Command": "InvokeLLM",
    "InvokeLLM": {
        "Content": "提醒用户订单已发货",
        "Priority": 1
    }
}
```

### 回复的获取

| 获取方式 | 说明 |
| --- | --- |
| 客户端（字幕事件） | LLM 回复文本通过字幕事件实时下发，端侧直接整句渲染，详细参见 （客户端）实时字幕 。 |
| 服务端（LLM 回复回调） | LLM 单轮回复完成时，回调推送该轮终态完整文本（含轮次 ID），详细参见 AI 对话回调事件 。 |

## 主动调用 TTS（文本直送播报）

### 使用方式

任务运行期间，业务后端通过**控制任务接口**下发 `Command=InvokeTTS` 指令：

- 下发的文本**不经过 LLM 推理**，直接送 TTS 合成，音频由回推机器人推入 TRTC 房间播报；
- AI 说出的内容与业务下发的文本**一字不差**，适合需要精确控制播报内容的场景：系统公告、审核拦截提示、运营活动话术、固定播报文案等；
- 播报遵循统一的打断控制——播报过程中用户开口（自动打断模式）或下发 `Interrupt` 指令即可打断。

### 与 InvokeLLM 的区别

| 对比项 | `InvokeLLM`（送 LLM） | `InvokeTTS`（送 TTS） |
| --- | --- | --- |

| | | |
|---|---|---|
| **链路** | 文本注入上下文 → LLM 动态生成回复 → TTS 播报 | 文本直接送 TTS 合成播报，**不经过 LLM**。 |
| **播报内容** | LLM 结合上下文动态生成，内容不确定。 | 业务下发的原文，一字不差。 |
| **典型用途** | 让 AI 结合上下文主动开口、事件提醒转述。 | 公告、提示、运营话术等**固定内容**的精确播报。 |

## 输入参数

| 参数名 | 必选 | 类型 | 描述 |
|---|---|---|---|
| `SdkAppId` | 是 | Integer | TRTC 应用 ID。 |
| `TaskId` | 是 | String | 任务 ID。 |
| `Command` | 是 | String | 指令名：`InvokeTTS`（不区分大小写）。 |
| `InvokeTTS` | 条件必填 | Object | `Command=InvokeTTS` 时必填，字段见下表。 |

### InvokeTTS 对象：

| 字段名 | 必选 | 类型 | 描述 |
|---|---|---|---|
| Content | 是 | String | 播报的文本内容，长度 ≤ 1000。 |
| Priority | 否 | Integer | 优先级，决定播报文本与当前交互的冲突处理策略：<br>• 1=高优先级；<br>• 2=中优先级；<br>• 3=低优先级；<br>各档具体行为详细参见 Priority 优先级说明 。 |

## 请求示例

```
{
    "SdkAppId": 1400000000,
    "TaskId": "-nG0jIFU7q3wqZnqm8xB5TY0n43U5rLb...",
    "Command": "InvokeTTS",
```

```
    "InvokeTTS": {
        "Content": "今天你有什么想关注的",
        "Priority": 1
    }
}
```

### 播报效果的获取

| 获取方式 | 说明 |
| --- | --- |
| 客户端（音频流） | 订阅回推机器人的音频流即可听到播报内容。 |
| 客户端（状态事件） | 播报期间机器人状态流转为：说话中，播报完成流转为：已说完，详细参见 （客户端）接收 AI 机器人状态 。 |
| 服务端（状态回调） | 通过 AI 状态流转回调跟踪播报起止，详细参见 AI 对话回调事件 。 |

## 固定话术播报（欢迎语与静音追问）

固定话术无需 LLM 动态生成，由后台在特定时机直接送 TTS 播报。通过创建任务时的 `DialogConfig` 配置：

| 字段名 | 必选 | 类型 | 描述 |
| --- | --- | --- | --- |
| `WelcomeMessage` | 否 | String | 欢迎语，主播进房后自动播报。 |
| `SilenceFollowUp` | 否 | Integer | 静音后 Bot 主动追问的时间（秒），范围 0-300，`0` =关闭（默认）。 |
| `SilenceFollowUpText` | 否 | String | 静音追问的文本内容。 |

**配置示例：**

```
"DialogConfig": {
    "InterruptMode": 0,
    "InterruptSpeechDuration": 200,
    "SilenceFollowUp": 50,
    "SilenceFollowUpText": "你还在吗？有想问的直接告诉我。",
    "WelcomeMessage": "你好，我是AI助手，有什么可以帮你的？"
}
```

> **说明：**
> 除上述三类外，`LLMConfig.FailureMessage`（LLM 调用失败时的兜底播报文本）同样属于固定话术，详细参见 LLM 配置。

## 典型应用场景

| 场景 | 实现方式 | 说明 |
| --- | --- | --- |
| **进房欢迎语** | `DialogConfig.WelcomeMessage` | 每次对话开始时 AI 自动播报一句欢迎语。 |
| **静音追问（避免冷场）** | `SilenceFollowUp` + `SilenceFollowUpText` | 用户长时间未说话，AI 主动追问，例如：你还在吗？有想问的直接告诉我。 |
| **业务事件主动提醒** | 运行时 `InvokeLLM` 注入 | 订单发货、活动开始、时间到点等关键节点，注入“提醒用户订单已发货”，由 AI 播报。 |
| **主动发起话题** | 运行时 `InvokeLLM` 注入 | 注入“用户一段时间没有说话了，请主动说一句话”，AI 结合上下文主动开口。 |
| **固定内容精确播报** | 运行时 `InvokeTTS` 注入 | 系统公告、审核拦截提示、运营活动话术等需要**一字不差**的播报。 |
| **确定性播报（跳过 LLM）** | 运行时 `InvokeTTS` 注入 | 对时延或内容确定性敏感的提醒，绕过 LLM 推理直接合成播报。 |

## 注意事项

1. **注入文本会进入对话上下文（仅 InvokeLLM）：** 主动调用 LLM 注入的内容作为上下文参与该轮及后续对话，请避免注入与业务无关的文本，以免干扰对话质量；`InvokeTTS` 文本不经过 LLM，直接合成播报。
2. **长度限制：** `InvokeLLM.Content` 与 `InvokeTTS.Content` 长度均 ≤ 1000 字符。
3. **按优先级选择冲突策略：** 高优先级会**立即终止当前交互**（用户可能听到半截话被截断），请仅在确实需要抢播时使用；对时延不敏感的通知建议用中优先级，避免频繁打断影响对话体验。
4. **播报可被打断：** 主动说话的 TTS 播报同样遵循打断控制——自动打断模式下用户开口即可打断 AI 的主动播报，详细参见（客户端）打断指令。
5. **仅对运行中的任务生效：** 任务结束或 Bot 离开房间后指令不再响应，建议通过状态流转回调维护任务生命周期。
6. **固定话术不可运行时修改：** 欢迎语与静音追问在创建任务时配置；`DialogConfig` 中仅 `InterruptMode` 支持通过修改任务接口热更新。
7. **音色跟随任务配置：** `InvokeTTS` 播报使用任务 `TTSConfig` 中配置的音色与厂商，指令本身不指定音色；如需更换音色请重建任务（`TTSConfig` 不支持热更新），详细参见 TTS 配置。

# 获取 AI 机器人状态以及上下文

最近更新时间：2026-11-06 17:00:41

## 功能概述

任务运行期间，业务后端有两种途径获取 AI 机器人的状态与对话数据：

| 获取方式 | 途径 | 可获取内容 | 实时性 |
| --- | --- | --- | --- |
| **查询接口（主动拉取）** | 任务生命周期五类标准操作中的**查询任务接口**。 | 任务运行状态、对话上下文历史。 | 按需查询，返回查询时刻的快照。 |
| **服务端回调（被动推送）** | AI 状态流转回调。 | 机器人的对话状态实时流转。 | 实时推送，无轮询延迟。 |

> **说明：**
> 客户端 SDK 也有对应的状态获取通道（状态事件），与服务端回调**共用同一套状态码定义**，端侧接入详细参见 （客户端）接收 AI 机器人状态 。

## 通过查询接口获取

查询任务接口通过 `DescribeType` 区分两类查询：

| DescribeType | 查询内容 | 返回字段 |
| --- | --- | --- |
| `0`（默认） | 任务运行状态 | `Status` / `StartTime` |
| `1` | 对话上下文历史 | `HistoryContents`（用户与 AI 的历史对话列表） |

## 查询任务运行状态（DescribeType = 0）

**输入参数：**

| 参数名 | 必选 | 类型 | 描述 |
| --- | --- | --- | --- |
| `SdkAppId` | 是 | Integer | TRTC 应用 ID。 |
| `TaskId` | 是 | String | 任务 ID。 |

| | | | |
|---|---|---|---|
| DescribeType | 否 | Integer | 0：查询任务状态（默认）。 |

**输出参数：**

| 参数名 | 类型 | 描述 |
|---|---|---|
| Status | String | 任务状态，见下方状态表。 |
| StartTime | Integer | 任务启动时间（UTC 毫秒）。 |
| RequestId | String | 请求 ID。 |

**任务状态（Status）取值：**

| Status | 含义 |
|---|---|
| InProgress | 任务运行中。 |
| Idle | 空闲（任务创建后、正式启动前）。 |
| Exited | 已退出（任务已结束）。 |

**请求示例：**

```
{
    "SdkAppId": 1400704311,
    "TaskId": "-m9jX35U7sY4p8GEKc0d4EnvzOlwCUTb...",
    "DescribeType": 0
}
```

**响应示例：**

```
{
    "Response": {
        "Status": "InProgress",
        "StartTime": 1751196050000,
        "TaskId": "-m9jX35U7sY4p8GEKc0d4EnvzOlwCUTb...",
        "RequestId": "req-xxx"
```

```
    }
}
```

## 查询对话上下文历史（DescribeType = 1）

返回当前任务内用户与 AI 的历史对话内容，可用于客服质检、审计落库、用户重连后恢复上下文等场景。
**输出参数：**

| 参数名 | 类型 | 描述 |
| --- | --- | --- |
| `HistoryContents` | Array of Object | 对话历史列表，每项含 `Role`（`user`=用户 / `assistant`=AI）与 `Content`（对话文本）。 |
| `Status` | String | 任务状态。 |
| `StartTime` | Integer | 任务启动时间（UTC 毫秒）。 |

**请求示例：**

```
{
    "SdkAppId": 1400704311,
    "TaskId": "-m9jX35U7sY4p8GEKc0d4EnvzOlwCUTb...",
    "DescribeType": 1
}
```

**响应示例：**

```
{
    "Response": {
        "StartTime": 1790542649581,
        "Status": "InProgress",
        "TaskId": "-nE5D7FU7k85T+dH1Mf-xdCO5s+...",
        "HistoryContents": [
            {"Role": "user", "Content": "今天天气怎么样？帮我查一下北京的温
度。"},
            {"Role": "assistant", "Content": "我无法获取实时天气，建议用天气
App查北京温度。"},
            {"Role": "user", "Content": "请把下面这句话原样读出来：高优插播"},
            {"Role": "assistant", "Content": "高优插播"},
            {"Role": "user", "Content": "今天天气怎么样？帮我查一下北京的温
度。"},
```

```
            {"Role": "assistant", "Content": "无法获取实时天气，请用天气App查
询北京温度。"}
        ],
        "RequestId": "req-xxx"
    }
}
```

> ⚠ **注意：**
> S2S（端到端）模式不支持查询对话历史，`DescribeType=1` 时返回为空。

# 通过服务端回调获取机器人状态

## AI 状态流转回调

任务运行期间，机器人每发生一次对话状态流转，TRTC 后台即向业务回调 URL **实时推送**一条 AI 状态流转事件，无需业务侧轮询。

**回调 Payload 关键字段：**

| 字段名 | 类型 | 描述 |
| --- | --- | --- |
| `State.Code` | Integer | 机器人对话状态码，见下表。 |
| `State.Desc` | String | 状态描述。 |
| `Timestamp` | Integer | 状态流转时间戳（毫秒）。 |

**回调示例（EventInfo.Payload 部分）：**

```
{
    "Payload": {
        "State": { "Code": 2, "Desc": "AI 思考中（LLM 推理）" },
        "Timestamp": 1751196060800
    }
}
```

> ⓘ **说明：**
> 回调报文的完整结构（事件组、事件类型枚举、回调 URL 配置与 `X-TRTC-Signature` 签名校验）详细参见 AI 对话回调事件。

## 机器人对话状态码（State.Code）

`State.Code` 描述的是**一轮对话**中机器人的实时状态，客户端 SDK 的状态事件与服务端状态流转回调共用同一套定义：

| State.Code | 含义 | 对应链路阶段 |
| --- | --- | --- |
| `0` | AI 空闲中 | 等待用户开口。 |
| `1` | AI 聆听中 | **ASR 识别中**（用户说话中），一轮对话的开始信号。 |
| `2` | AI 思考中 | **LLM 推理中**。 |
| `3` | AI 说话中 | **TTS 播报中**。 |
| `4` | AI 被打断 | 轮次**提前终止**。 |
| `5` | AI 已说完 | 轮次**正常结束**。 |

**一轮对话的典型状态流转：**




- 正常路径：用户开口（1）→ LLM 推理（2）→ TTS 播报（3）→ 播报完成（5）→ 回到空闲（0）等待下一轮。
- 打断路径：播报中（3）被用户打断或收到打断指令，直接流转为被打断（4）→ 回到空闲（0）。打断的触发方式详细参见 （客户端）打断指令 。

## 两类状态的区分

查询接口返回的 `Status`（任务状态）与回调推送的 `State.Code`（对话状态）是**两个不同层级的状态**，注意区分：

| 对比项 | 任务状态（ `Status` ） | 对话状态（ `State.Code` ） |
| --- | --- | --- |
| **描述层级** | 任务生命周期。 | 单轮对话生命周期。 |
| **获取方式** | 查询接口（ `DescribeType=0` ）/ 任务结束回调。 | AI 状态流转回调 / 客户端状态事件。 |

| | | |
|---|---|---|
| **类型与取值** | String：<br>• `InProgress` 运行中；<br>• `Idle` 空闲；<br>• `Exited` 已退出。 | Integer：<br>• `0` 空闲；<br>• `1` 聆听中；<br>• `2` 思考中；<br>• `3` 说话中；<br>• `4` 被打断；<br>• `5` 已说完。 |
| **典型用途** | 任务级监控、存活巡检、故障告警。 | 对话过程展示、打断检测、轮次统计。 |

> **说明：**
> 任务级的状态变化（启动完成、结束及退出原因等）通过任务启动 / 结束回调推送，详细参见 AI 对话回调事件。

## 典型应用场景

| 场景 | 实现方式 | 说明 |
|---|---|---|
| **对话质检与审计** | 查询接口 `DescribeType=1` | 会话结束后拉取完整对话上下文，落库用于质检、审计与合规。 |
| **任务监控告警** | 查询接口 `DescribeType=0` 轮询 / 任务结束回调 | 发现任务 `Exited`（已退出）及时告警，结合退出码判断是否为异常退出。 |
| **实时状态大屏** | 订阅 AI 状态流转回调 | 回调实时驱动机器人状态展示（聆听中 / 思考中 / 说话中），无需轮询。 |
| **轮次统计分析** | 累计状态流转回调 | 基于 `1→5` 完整流转统计轮次数，基于 `4` 统计打断率。 |
| **重连恢复上下文** | 查询接口 `DescribeType=1` | 用户断线重连或切换设备后，拉取历史对话恢复上下文继续服务。 |

## 注意事项

1. **S2S 模式不支持查询对话历史**：端到端模式下 `DescribeType=1` 返回为空；如需对话记录请通过字幕回调 / 状态回调自行落库。
2. **查询返回的是快照**：查询接口返回查询时刻的状态，如需实时感知状态变化请订阅状态流转回调，不要依赖轮询（轮询既有时延又消耗接口 QPS 配额）。
3. **回调必须校验签名**：业务后端收到回调后须校验 `X-TRTC-Signature` 签名防止伪造，配置方式见 AI 对话回调事件。

4. **任务退出排查**：查询到 `Status` 为 `Exited` 时，建议结合任务结束回调的退出码与 ASR / LLM 回调定位退出原因（区分正常结束与异常失败）。
5. **状态回调可能重复**：网络重传等情况下回调可能重复投递，业务侧需按 `TaskId` + `Timestamp` 做幂等处理。

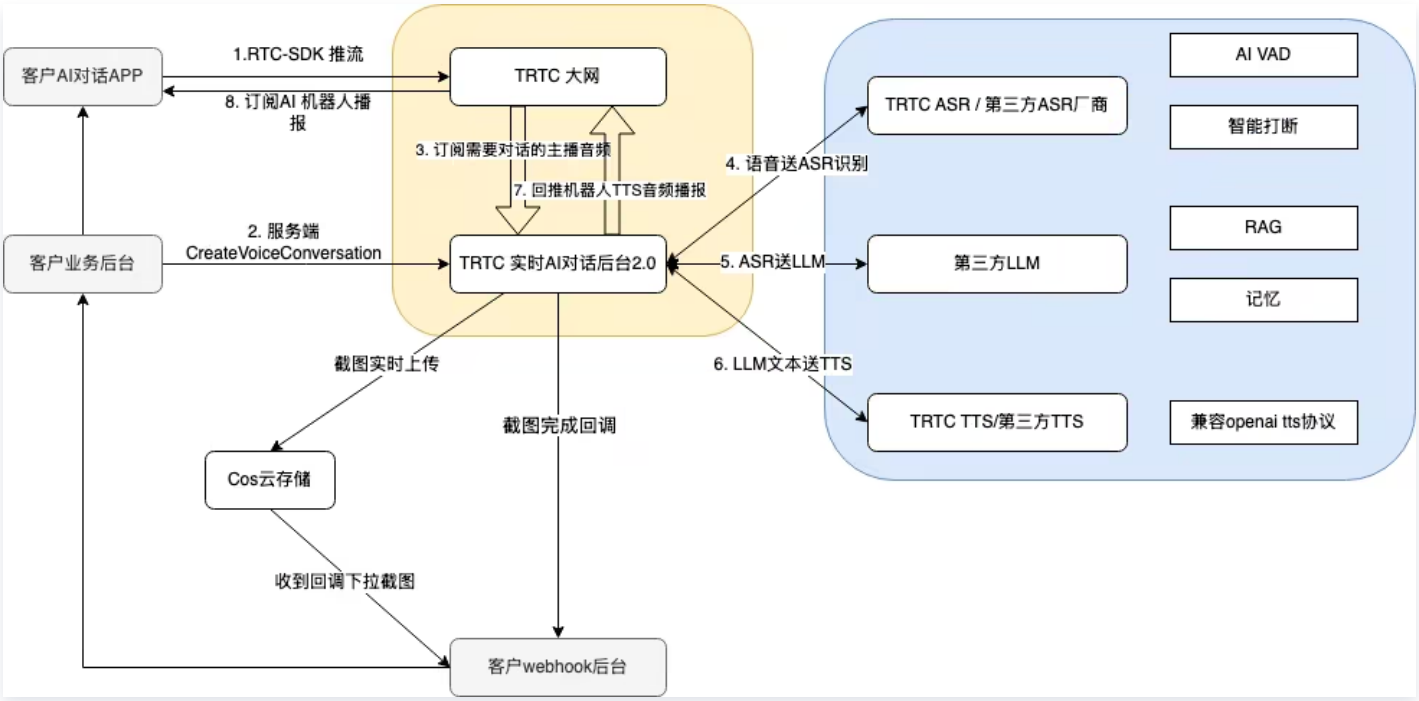
# 高级功能
# 视觉理解能力（截图问答）

最近更新时间：2026-11-06 17:00:24

## 功能概述

为支撑 **“边打电话边看屏”** 一类的 AI 实时语音助手场景，TRTC AI 实时对话在音频对话链路的基础上，提供**截图 + 视频上行的视觉理解（VLM）能力**：客户端按需截取屏幕 / 摄像头画面并上行，AI 后台将截图连同用户的语音提问一同送入多模态大模型，LLM 基于“用户语音 + 画面内容”共同理解并回复。

**截图对话(三段式)架构图：**




## 功能原理

视觉理解链路不进行实时逐帧推理，而是围绕“按需截取、按需理解”设计：

| 环节 | 说明 |
| --- | --- |
| **截图策略** | 基于环境变化、陀螺仪感知等触发，截取屏幕 / 摄像头画面。 |
| **后台缓存 + 存储** | TRTC 后台接收后**缓存最近若干帧**（滑动缓冲区）；可选将截图异步上传到云存储归档。 |
| **按需送 LLM** | 发生交互时（用户语音提问、周期性触发），后台从缓存队列取出截图，连同用户文本/指令发送给多模态 LLM 分析。 |

| | |
|---|---|
| VLM 回复 | LLM 基于“用户语音 + 截图内容”共同理解并回复，例如：“你截图里的这个按钮点不动，是因为它当前是灰色禁用状态”。 |

典型的交互形态包括：**视频截图对话**（对当前画面提问）、**定帧提问**（对指定帧提问）、**本地相册提问**（对本地图片提问）。

## 配置结构

创建 AI 对话任务时，通过 `VisionConfig` 启用视觉理解：

```
"VisionConfig": {
    "Enable": true,
    "StorageType": 0,
    "CloudStorage": null
}
```

### VisionConfig 字段说明：

| 字段名 | 必选 | 类型 | 描述 |
|---|---|---|---|
| `Enable` | 是 | Boolean | 是否启用 VLM。**运行时不可切换，需重建任务。** |
| `StorageType` | 否 | Integer | 存储类型：<br>• `0` =不存储（默认）；<br>• `1` =指定云存储（需填 `CloudStorage`）。<br>**不支持更新。** |
| `CloudStorage` | 否 | Object | 云存储归档配置，见下表。 |

### CloudStorage 字段说明：

| 字段名 | 必选 | 类型 | 描述 |
|---|---|---|---|
| `Vendor` | 是 | Integer | • `0` =腾讯云 COS；<br>• `1` =AWS S3。 |
| `Region` | 是 | String | 存储地域。 |
| `Bucket` | 是 | String | 存储桶名称。 |
| `AccessKey` | 是 | String | 访问密钥（COS 对应 SecretId）。 |

版权所有：腾讯云计算（北京）有限责任公司

| | | | |
|---|---|---|---|
| `SecretKey` | 是 | String | 密钥（COS 对应 SecretKey）。 |
| `FileNamePrefix` | 否 | Array of String | 存储路径前缀数组，总长度 ≤ 128。 |

## 启用条件

| 条件 | 说明 |
|---|---|
| **多模态模型** | `LLMConfig.Model` 必须是多模态模型，例如 `hunyuan-vision` / `gpt-4o` / `gpt-4o-mini` / `qwen-vl-plus` 。 |
| **仅 Cascade 模式** | 视觉理解仅三段式模式支持，S2S + VLM 组合暂不可用。 |
| **SDK 版本** | 客户端 TRTC SDK 需升级到支持截图功能的版本。 |
| **不支持热更新** | `VisionConfig` 不支持运行时更新，变更需结束任务并重新创建。 |

## 客户端接入

### 步骤1：进房

客户端集成 TRTC SDK 后，以主播角色进入房间（成功条件：`TRTCCloudListener.onEnterRoom(result > 0)`）：

```
TRTCParams params = new TRTCParams();
params.sdkAppId = <sdkAppId>;
params.userId   = <localUserId>;
params.userSig  = <userSig>;
params.roomId   = <roomId>;
params.role     = TRTCRoleAnchor;

trtc.enterRoom(params, TRTC_APP_SCENE_VIDEOCALL);
```

### 步骤2：连接后台机器人

```
trtc.callExperimentalAPI(JSON);
JSON:
{
  "api": "aiConversation",
```

```
  "params": {
    "command": "connect",
    "robotid": "<Robot User Id>"
  }
}
```

连接结果通过 `onExperimentalEvent` 的 `onCommand` 事件回调：

| code | 含义 |
| --- | --- |
| `0` | 通道建立成功。 |
| `-2` | 参数错误，缺少 robotid。 |
| `2001` | 任务不存在或已结束。 |
| `2002` | SDK 版本过低，不支持所需能力。 |

## 步骤3：截图提问

使用 TRTC 截图 API 截取画面，压缩为 JPEG 后通过试验性接口发送给 AI：

```
trtc.snapshotVideo("", TRTCCloudDef.TRTC_VIDEO_STREAM_TYPE_BIG,
        TRTCCloudDef.TRTC_SNAPSHOT_SOURCE_TYPE_VIEW,
        new TRTCCloudListener.TRTCSnapshotListener() {
            @Override
            public void onSnapshotComplete(Bitmap bitmap) {
                if (bitmap == null) {
                    return;
                }
                try {
                    ByteArrayOutputStream out = new
ByteArrayOutputStream();
                    bitmap.compress(Bitmap.CompressFormat.JPEG, 80,
out);
                    // 将 JPEG 数据 Base64 编码后调用 sendImage 接口发送
                }catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
```

```
trtc.callExperimentalAPI(JSON);
JSON:
{
  "api": "aiConversation",
  "params": {
    "command": "sendImage",
    "robotid": "<Robot User Id>",
    "payload": {
      "format": "jpeg",
      "image": "<Base64 JPEG>"
    }
  }
}
```

**说明：**
`sendImage` 同样适用于**本地相册提问**，将相册图片读取并 Base64 编码后，通过同一接口发送即可。发送成功不回调，仅失败时通过 `onCommand` 事件返回错误码。

## 云存储归档

配置 `StorageType=1` + `CloudStorage` 后，后台将收到的每帧截图**异步上传至您的云存储桶**归档，对象路径规则：

```
vision/{SdkAppId}/{YYYYMMDD}/{TaskId}/{captureTs}_{reason}_{frameId}.jpg
```

| 路径字段 | 说明 |
| --- | --- |
| `SdkAppId` | 应用 ID。 |
| `YYYYMMDD` | 截图日期。 |
| `TaskId` | 任务 ID。 |
| `captureTs` | 截图时间戳。 |
| `reason` | 截图原因（例如 `scene_change`）。 |
| `frameId` | 帧 ID。 |

每批截图上传完成后，通过**截图上传回调**通知业务后端（含截图时间、原因、对象 Key、大小等），详细参见 AI 对话回调事件 。

> ⚠ **注意：**
> 不配置 `CloudStorage` 时，截图仅驻留后台内存用于当轮对话理解，不落盘、不归档。

## 注意事项

1. **多模态模型是硬性前提：** `LLMConfig.Model` 为纯文本模型时启用 VLM 会导致视觉内容无法被理解。
2. **不支持热更新：** `VisionConfig`（含 `Enable`、`StorageType`、`CloudStorage`）均不支持通过修改任务接口变更，需重建任务。
3. **仅 Cascade 模式支持：** S2S 模式下音频直接进端到端模型，暂无视觉理解链路。
4. **截图时机：** 后台对截图做缓存后按需取用，无需业务侧实时推送每一帧；建议结合环境变化、陀螺仪感知等策略截取关键画面，降低上行带宽。

版权所有：腾讯云计算（北京）有限责任公司

# AI 对话 2.0 服务端回调

最近更新时间：2026-10-13 15:50:47

本文用于介绍 AI 对话 2.0 服务相关的云 API 接口产生的事件，以 HTTP 请求的形式通知到您的服务器。任务运行期间，TRTC 将 AI 对话过程中的关键事件以回调（Webhook）方式实时推送到业务后端的回调 URL，业务侧无需轮询查询接口即可获取对话内容、状态流转与任务生命周期信息。

> **说明：**
> AI 对话 2.0 使用全新的事件组，与 1.0 的回调事件完全隔离，互不影响。

## 功能概述

| 能力 | 对应事件 | 业务价值 |
| --- | --- | --- |
| 对话内容落库 | ASR 识别结果、LLM 回复结果 | 用户与 AI 发言实时归档，用于质检、审计与后续分析。 |
| 状态同步 | AI 状态流转 | 机器人对话状态实时流转（聆听中/思考中/说话中…），驱动业务侧状态展示。 |
| 任务监控告警 | 任务启动完成、任务结束 | 启动结果与结束原因（LeaveCode）第一时间触达，便于及时告警与排障。 |
| 视觉理解归档 | 截图上传结果 | 截图上传云存储的结果通知，配合完成视觉素材归档。 |

## 配置信息

在实时音视频 TRTC 控制台（或通过腾讯云技术支持）配置业务后端的回调 URL 与回调密钥，配置完成后即可接收事件回调通知。详细操作指引请参见 回调配置 。

> **注意：**
> 您需要提前准备以下信息：
> - **必要项：**接收回调通知的 HTTP/HTTPS 服务器地址，需公网可达，仅支持 POST 方法。
> - **必要项：**用于 `Sign` 签名校验的回调密钥，需妥善保密。密钥仅用于服务端签名校验，不要下发到客户端或写入前端代码。

| 配置项 | 要求 |
| --- | --- |
| 回调 URL | 公网可达的 HTTP(S) 地址，仅支持 POST 方法。 |
| 回调密钥 | 用于 `Sign` 签名校验，需妥善保密。 |

## 超时重试

网络重传等情况下回调可能重复投递，业务侧需按 `TaskId` + 事件类型 + 时间戳（ `EventMsTs` ）做幂等去重。收到回调后请尽快返回 HTTP 200，耗时处理（落库、分析等）建议异步进行，避免阻塞导致重复投递。

## 事件回调消息格式

事件回调消息以 HTTP POST 请求发送给您的服务器，其中：

- **请求**：HTTP 请求的 header 中包含签名等信息，body 格式为 JSON。
- **应答**：HTTP STATUS CODE = 200，服务端忽略应答包具体内容，为了协议友好，建议客户应答内容携带 JSON： {"code":0}。
- **包体示例**：所有回调事件共用同一套报文结构，通过 `EventType` 区分具体事件，事件内容放在 `EventInfo.Payload` 中。下述为通用报文结构，各事件详情中的 JSON 为填充了对应 Payload 的完整报文。

> ⚠ **注意：**
> - 您的 HTTP 应答包长（header+body）需要控制在2000个字节以内。
> - POST 请求的 JSON 包体不会删减已有字段，但会根据业务需求添加新字段。在集成回调时，您需要适应这些新增字段的情况。

```
{
    "EventGroupId": 15,
    "EventType": 1501,
    "CallbackTs": 1751196050100,
    "EventInfo": {
        "EventMsTs": 1751196050080,
        "TaskId": "task-88a1b2c3",
        "RoomId": "room-user123",
        "RoomIdType": 1,
        "Payload": { }
    }
}
```

## 参数说明

### 回调消息参数

- 事件回调消息的 header 中包含以下字段：

| 字段名 | 含义 |
| --- | --- |

| | |
|---|---|
| Content-Type | application/json。 |
| Sign | 签名值。 |
| SdkAppId | 创建应用时控制台分配的 SdkAppId。 |

- 事件回调消息的 body 中包含以下字段：

| 字段名 | 类型 | 含义 |
|---|---|---|
| EventGroupId | Integer | 事件组 ID，AI 对话 2.0 事件组固定为 15。 |
| EventType | Integer | 回调通知的事件类型。 |
| CallbackTs | Integer | 事件回调服务器向您的服务器发出回调请求的 Unix 时间戳，单位为毫秒。 |
| EventInfo.EventMsTs | Integer | 事件发生的 Unix 时间戳，单位为毫秒。 |
| EventInfo.TaskId | String | AI 任务 ID。 |
| EventInfo.RoomId | String | TRTC 的房间 ID。 |
| EventInfo.RoomIdType | Integer | • 0：表示数字房间号；<br>• 1：表示字符串房间号。 |
| EventInfo.Payload | Object | 事件内容，随事件类型不同而不同，见下文各事件详情。 |

## 事件组 ID

| 字段名 | 值 | 含义 |
|---|---|---|
| EVENT_GROUP_CLOUD_CONVERSATION | 15 | AI 对话 2.0 事件组。 |

## 事件类型

| 字段名 | 值 | 含义 | 触发时机 | Payload 关键字段 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| EVENT_TYPE_CONVERSATION_SERVICE_START | 1501 | 任务启动完成 | AI 服务启动流程结束（成功或失败） | Status 、ConversationMode |
| EVENT_TYPE_CONVERSATION_SERVICE_STOP | 1502 | 任务结束 | 任务退出（主动结束/被踢/异常等） | LeaveCode |
| EVENT_TYPE_CONVERSATION_ASR_MSG | 1503 | ASR 识别结果 | 用户一句话识别完成（终态） | UserId 、Text 、RoundId 、起止时间 |
| EVENT_TYPE_CONVERSATION_LLM_MSG | 1504 | LLM 回复结果 | AI 单轮回复生成完成（终态） | Text 、RoundId |
| EVENT_TYPE_CONVERSATION_AI_STATUS | 1505 | AI 状态流转 | 机器人对话状态每次变化 | State.Code 、State.Desc 、Timestamp |
| EVENT_TYPE_CONVERSATION_VISION_FRAME_UPLOADED | 1506 | 截图上传结果 | 周期性截图上传云存储后 | Count 、Frames[] |

**稳态回调说明：**
ASR 识别结果与 LLM 回复结果事件只在终态投递一次完整文本，不推送中间流式片段。

## 1501：任务启动完成

AI 服务启动流程结束（Bot 进房、链路就绪）后推送。

| 字段名 | 类型 | 含义 |
|---|---|---|
| EventMsTs | Integer | 事件发生的 Unix 时间戳，单位为毫秒。 |
| TaskId | String | AI 任务 ID。 |
| RoomId | String | TRTC 的房间 ID。 |
| RoomIdType | Integer | • 0：表示数字房间号；<br>• 1：表示字符串房间号。 |
| Payload.Status | Integer | 启动结果：<br>• 0：启动 AI 任务成功； |

| | | |
|---|---|---|
| | | • 1：启动 AI 任务失败。 |
| Payload.ConversationMode | String | 对话模式：<br>• Cascade；<br>• S2S。 |

```
{
    "EventGroupId": 15,
    "EventType": 1501,
    "CallbackTs": 1751196050100,
    "EventInfo": {
        "EventMsTs": 1751196050080,
        "TaskId": "task-88a1b2c3",
        "RoomId": "room-user123",
        "RoomIdType": 1,
        "Payload": {
            "Status": 0,
            "ConversationMode": "Cascade"
        }
    }
}
```

## 1502：任务结束

**任务退出后推送，通过 `LeaveCode` 区分退出原因。**

| 字段名 | 类型 | 含义 |
|---|---|---|
| EventMsTs | Integer | 事件发生的 Unix 时间戳，单位为毫秒。 |
| TaskId | String | AI 任务 ID。 |
| RoomId | String | TRTC 的房间 ID。 |
| RoomIdType | Integer | • 0：表示数字房间号；<br>• 1：表示字符串房间号。 |
| Payload.LeaveCode | Integer | • 0：正常结束，业务后端调用「删除任务」接口后退出；<br>• 1：业务自行踢掉 Bot 后退出； |

| | | |
|---|---|---|
| | | • 2：业务解散房间后退出；<br>• 3：TRTC 服务端踢掉机器人；<br>• 4：TRTC 服务端解散房间；<br>• 98：内部异常错误，建议业务重试；<br>• 99：房间内除 Bot 外无其他用户流，超时退出。 |


```
{
    "EventGroupId": 15,
    "EventType": 1502,
    "CallbackTs": 1751196050100,
    "EventInfo": {
        "EventMsTs": 1751196050080,
        "TaskId": "task-88a1b2c3",
        "RoomId": "room-user123",
        "RoomIdType": 1,
        "Payload": {
            "LeaveCode": 0
        }
    }
}
```


## 1503：ASR 识别结果

**用户一句话识别完成（终态）后推送，`Text` 为该句完整识别文本。**

| 字段名 | 类型 | 含义 |
|---|---|---|
| EventMsTs | Integer | 事件发生的 Unix 时间戳，单位为毫秒。 |
| TaskId | String | AI 任务 ID。 |
| RoomId | String | TRTC 的房间 ID。 |
| RoomIdType | Integer | • 0：表示数字房间号；<br>• 1：表示字符串房间号。 |
| Payload.UserId | String | 说话用户的 ID。 |

| | | |
|---|---|---|
| Payload.Text | String | 识别出的完整文本（终态）。 |
| Payload.StartTimeMs | Integer | 句子开始时间，单位为毫秒。 |
| Payload.EndTimeMs | Integer | 句子结束时间，单位为毫秒。 |
| Payload.StartUtcMs | Integer | 句子开始的 UTC 时间，单位为毫秒。 |
| Payload.EndUtcMs | Integer | 句子结束的 UTC 时间，单位为毫秒。 |
| Payload.RoundId | String | 轮次 ID，用于关联同一轮的 ASR 与 LLM 结果。 |


```
{
    "EventGroupId": 15,
    "EventType": 1503,
    "CallbackTs": 1751196050100,
    "EventInfo": {
        "EventMsTs": 1751196050080,
        "TaskId": "task-88a1b2c3",
        "RoomId": "room-user123",
        "RoomIdType": 1,
        "Payload": {
            "UserId": "user123",
            "Text": "这个按钮为啥点不动？",
            "StartTimeMs": 1751196058500,
            "EndTimeMs": 1751196060100,
            "StartUtcMs": 1751196058500,
            "EndUtcMs": 1751196060100,
            "RoundId": "2b7d223d-35b8-4d29-bd1a-69ec8cb89f0b-0"
        }
    }
}
```


## 1504：LLM 回复结果

AI 单轮回复生成完成（终态）后推送，`Text` 为该轮完整回复文本。

| 字段名 | 类型 | 含义 |
| --- | --- | --- |
| EventMsTs | Integer | 事件发生的 Unix 时间戳，单位为毫秒。 |
| TaskId | String | AI 任务 ID。 |
| RoomId | String | TRTC 的房间 ID。 |
| RoomIdType | Integer | • 0：表示数字房间号；<br>• 1：表示字符串房间号。 |
| Payload.Text | String | LLM 单轮回复的完整文本（终态）。 |
| Payload.RoundId | String | 轮次 ID，与同轮的 ASR 识别结果事件一致。 |

```
{
    "EventGroupId": 15,
    "EventType": 1504,
    "CallbackTs": 1751196050100,
    "EventInfo": {
        "EventMsTs": 1751196050080,
        "TaskId": "task-88a1b2c3",
        "RoomId": "room-user123",
        "RoomIdType": 1,
        "Payload": {
            "Text": "这个按钮是灰色禁用状态，需要先勾选上方的\"同意协议\"才能激活。",
            "RoundId": "554772d4-5f23-4b5a-8361-764d0fc7bb0f-0"
        }
    }
}
```

## 1505：AI 状态流转

机器人每发生一次对话状态流转即推送一条，实时性最高。状态码与客户端 SDK 的状态事件共用同一套定义。

| 字段名 | 类型 | 含义 |
| --- | --- | --- |

| | | |
|---|---|---|
| EventMsTs | Integer | 事件发生的 Unix 时间戳，单位为毫秒。 |
| TaskId | String | AI 任务 ID。 |
| RoomId | String | TRTC 的房间 ID。 |
| RoomIdType | Integer | • 0：表示数字房间号；<br>• 1：表示字符串房间号。 |
| Payload.State.Code | Integer | 机器人对话状态码：<br>• 0：AI 空闲中；<br>• 1：AI 聆听中（ASR 识别中）；<br>• 2：AI 思考中（LLM 推理中）；<br>• 3：AI 说话中（TTS 播报中）；<br>• 4：AI 被打断（轮次提前终止）；<br>• 5：AI 已说完（轮次正常结束）。 |
| Payload.State.Desc | String | 状态描述。 |
| Payload.Timestamp | Integer | 状态流转时间戳，单位为毫秒。 |

```
{
    "EventGroupId": 15,
    "EventType": 1505,
    "CallbackTs": 1751196050100,
    "EventInfo": {
        "EventMsTs": 1751196050080,
        "TaskId": "task-88a1b2c3",
        "RoomId": "room-user123",
        "RoomIdType": 1,
        "Payload": {
            "State": {
                "Code": 5,
                "Desc": "AI Speaking Finished"
            },
            "Timestamp": 1790820920603
        }
```

```
    }
}
```

## 1506：截图上传结果

**启用视觉理解并配置云存储归档（ `StorageType=1` + `CloudStorage` ）后，后台将截图异步上传至云存储，并周期性推送上传结果。**

| 字段名 | 类型 | 含义 |
| --- | --- | --- |
| EventMsTs | Integer | 事件发生的 Unix 时间戳，单位为毫秒。 |
| TaskId | String | AI 任务 ID。 |
| RoomId | String | TRTC 的房间 ID。 |
| RoomIdType | Integer | • 0：表示数字房间号；<br>• 1：表示字符串房间号。 |
| Payload.PeriodStartMs | Integer | 本批次截图的时间区间起点，单位为毫秒。 |
| Payload.PeriodEndMs | Integer | 本批次截图的时间区间终点，单位为毫秒。 |
| Payload.Count | Integer | 本批次上传的截图数量。 |
| Payload.Frames | Array of Object | 截图列表，每项字段定义见下表。 |

`Payload.Frames` **元素字段：**

| 字段名 | 类型 | 含义 |
| --- | --- | --- |
| FrameId | String | 截图帧 ID。 |
| CaptureMs | Integer | 截图时间，单位为毫秒。 |
| Reason | String | 截图触发原因，如 scene_change 表示场景变化。 |
| ObjectKey | String | 云存储对象路径，格式为： `vision/{SdkAppId}/{日期}/{TaskId}/{captureTs}_{reason}_{frameId}.jpg` 。 |
| SizeBytes | Integer | 图片大小，单位为字节。 |

```
{
    "EventGroupId": 15,
    "EventType": 1506,
    "CallbackTs": 1751196050100,
    "EventInfo": {
        "EventMsTs": 1751196050080,
        "TaskId": "task-88a1b2c3",
        "RoomId": "room-user123",
        "RoomIdType": 1,
        "Payload": {
            "PeriodStartMs": 1751196060000,
            "PeriodEndMs": 1751196061000,
            "Count": 2,
            "Frames": [
                {
                    "FrameId": "F-8a3f2c1e-0007",
                    "CaptureMs": 1751196060120,
                    "Reason": "scene_change",
                    "ObjectKey": "vision/1400000000/20260812/task-
88a1b2c3/1751196060120_scene-change_F-8a3f2c1e-0007.jpg",
                    "SizeBytes": 52310
                }
            ]
        }
    }
}
```

## 计算签名

签名放在请求头 `Sign` 中。您的事件回调接收服务器收到回调消息后，通过同样的方式计算出签名，相同则说明是腾讯云实时音视频的事件回调，没有被伪造。签名的计算方式与 Java / Python / PHP / Golang 校验示例代码，详见 签名校验示例 。

> ⚠ **注意：**
> 计算签名时，body 须为您收到回调请求的原始包体，不要做任何转化，需完整保留 `\n` 、`\t` 等转义字符。

## 注意事项

1. **必须校验签名**：未校验 `Sign` 的回调接收端点存在被伪造请求攻击的风险。

2. **稳态回调**：ASR 识别结果与 LLM 回复结果只在终态投递一次完整文本，不推送中间流式片段；如需流式字幕请使用客户端字幕通道。
3. **幂等处理**：网络重传等情况下回调可能重复投递，业务侧需按 `TaskId` + 事件类型 + 时间戳（ `EventMsTs` / `Timestamp` ）做幂等去重。
4. **及时响应**：收到回调后尽快返回 HTTP 200，耗时处理（落库、分析等）建议异步进行，避免阻塞导致重复投递。
5. **回调密钥保密**：密钥仅用于服务端签名校验，不要下发到客户端或写入前端代码。
6. **截图回调的前提**：截图上传结果事件仅在配置云存储归档后才会推送，不配置时截图仅驻留内存、不落盘。

# AI 实时对话 1.0 功能介绍

最近更新时间：2025-12-19 15:54:51

## 功能概述

腾讯 AI 实时对话解决方案，支持客户灵活接入多家 AI 大模型服务，实现 AI 与用户之间的实时音视频互动，打造符合业务场景的 AI 实时对话能力。基于腾讯 RTC 全球低延迟传输，语音对话延迟低至1s，对话效果自然拟真，接入便捷，开箱即用。




## 计费说明

该功能的使用费用包含三个模块：音频通话费用、AI实时对话服务费用和语音转文字服务费用。
具体费用详情请参见 AI 实时对话计费说明 。

> **说明：**
> 该服务并发限制 100 QPS ，若您有更高的并发需求，请 联系我们 。

## Demo 体验

| Web 端 | Android 端 | iOS 端 |
| --- | --- | --- |
| AI 实时对话 Demo | Android 端 Demo | iOS 端 Demo |

## 无代码快速跑通 AI 对话

腾讯 RTC 提供了便捷的测试验证平台，支持您对接第三方 LLM 和 TTS 能力，快速配置，无代码快速跑通对话式 AI 功能进行验证测试，实现高效开发和部署。
配置地址：实时音视频控制台 > 快速跑通 AI 实时对话 。

## 应用场景

| 应用场景 | 说明 |
| --- | --- |
| 在线教育 | 在线教育场景中，实时互动和反馈是提升学习效果的关键。依托 AI 实时对话，平台可以创建虚拟教学助手，在课内课外提供全时的智能教学辅助。<br>• 课内，学生可以在老师讲课的同时，随时向虚拟教学助手提问，获得补充讲解，更充分地理解学习要点。<br>• 课外，虚拟教学助手可以根据不同学生的进度和需求，提供个性化的辅导建议和学习资源，并针对学生的作业、提问提供响应式反馈，以更自然亲和的方式陪伴学生。相比大段的文字解析，对话式的讲解可以更有效地引导学生，便于学生理解。 |
| 社交娱乐 | 在社交娱乐场景中，结合实时互动能力的 AI 实时对话能够精准理解用户意图并和用户进行语音互动，为用户带来更真实和个性化的社交娱乐体验。相比文字，AI 实时对话提供的虚拟陪伴服务能够通过语音与用户进行自然沟通，提供更为丰富且真实的情感价值。在线剧本杀、狼人杀等互动游戏中，AI 实时对话也可以扮演主持或 NPC 角色，与玩家进行动态 AI 实时对话并推动情节发展，让玩家享受沉浸式的游戏体验。 |
| 呼叫中心 | 在线客服、AI 销售顾问、智能外呼等场景都可以通过 AI 实时对话来提供更丰富、实时的客户服务体验，这样不仅可以有效降低运营成本，还能够显著提升服务效率，全天候为客户提供更快捷的服务支持。 |
| 高效办公 | 通过 AI 实时对话，用户可以使用语音，命令和控制应用程序，减少手动输入，使日常工作变得更轻松、高效。相比文字交互，对话式交互可以拓展各类办公助手的使用场景，无需在终端设备旁也能通过语音快速交流，完成工作。 |
| 医疗辅助 | 依托 AI 实时对话，远程诊断、医疗咨询等场景中，患者可以通过语音咨询提问，获取实时且个性化的建议，更接近真实场景问诊体验，这样可以消除用户的不信任感，大幅减轻患者 |

| | |
|---|---|
| | 焦虑。 |

> **说明：**
> 请查看行业实践教程：情感陪伴 、智能客服 。

## 功能优势

| 优势 | 说明 |
|---|---|
| **超低延迟的 AI 实时对话** | 在实时人工智能交互场景中，LLM 及时接收和处理用户的音视频数据至关重要。腾讯 RTC 的超低延迟通信确保了全球范围内音视频传输的端到端延迟低于300ms，同时将对话延迟保持在1000ms以下，媲美人类对话反应速度，让用户享受到流畅自然的互动体验，提升客户满意度。 |
| **接入简单，高效上线** | 提供无代码快速跑通平台，仅需10min，可预先快速验证解决方案。正式集成提供含 UI 的对接方式以及完整的 SDK 和 API 文档，简化开发流程，集成可在1 – 2天内完成，比传统方案节省1个月以上的开发工作，助力企业快速实现产品智能化升级，抢占市场先机。 |
| **自然拟真的对话** | 我们支持多种语言的输入，包括英语、西班牙语、日语、韩语、中文等130种国际语言。支持为最多三种指定语言提供模糊识别（不包括方言），确保了识别的高精度和适应性。结合自研 VAD 技术实现 AI 对话智能语义打断，更好的适应人类对话节奏与响应速度，带来超拟人的对话互动体验。 |
| **服务模型集成灵活性** | 与各种 LLM 和 TTS 模型无缝集成：我们提供了集成通道，允许用户轻松连接第三方 LLM 和 TTS 模型。用户只需要配置他们的 LLM 和 TTS 服务的账户凭证，就可以将它们无缝集成到我们的解决方案中。这促进了个性化和复杂的 AI 响应，增强了整体对话体验。 |
| **跨端兼容性** | 支持多个平台，包括 iOS、Android、Windows、macOS、Web、Flutter、Electron 和 React Native 等，兼容超过20,000种设备模型。 |
| **领先的音频处理** | 支持服务器端和客户端 AI 降噪，声纹识别和 3A 回声消除功能，可以根据不同的AI聊天模式进行定制，提高语音识别精度和 AI 通话质量，实现各种场景下的精准高清 AI 对话。 |
| **声纹识别** | 毫秒级响应，精准识别用户声纹特征，支持多场景实时身份核验。 深度结合 RTC 超低延迟通信，无缝融入对话流，跨平台兼容设备，端到端加密保障数据隐私。适用于多角色交互场景，让 AI 对话更智能、更安全。 |

# 开通服务

最近更新时间：2026-10-13 15:50:18

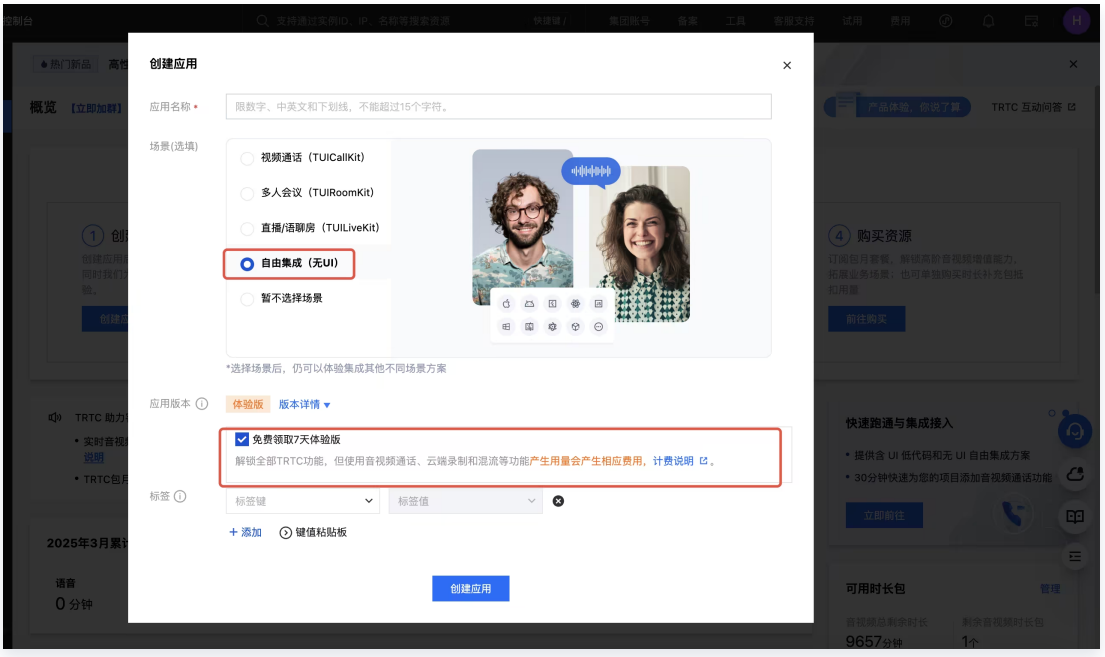
本文介绍如何 开通体验版 、开通正式版 和 续费正式版 服务，您可以根据 AI 实时对话计费说明 了解所需要的服务及费用，并参考下述指引免费开通和购买开通正式版 AI 实时对话服务。

## 开通体验版

1. 登录 实时音视频 TRTC 控制台，单击**创建应用**。




2. 在弹窗中输入您的应用名称，选择**自由集成（无 UI）**场景，勾选领取7天体验版，单击**创建应用**。




3. 进入应用管理页面，成功开通**自由集成（无 UI）**的7天试用服务。领取完成后，可以看到体验版的基本信息，包括服务状态、版本信息和功能详情、到期时间。这里的 SDKAppID 以及 SDK 密钥（SDKSecretKey）会在后续步骤中使用到。




> ⚠ **注意：**
>
> - 领取体验版后仅开通自由集成（无 UI）7天的体验资格，测试过程中所产生的音视频时长等资源消耗（例如音频通话、实时对话服务仍然会产生后付费产品费用，具体可参见 AI 实时对话费用详情），仍会按照实时音视频 TRTC 标准计费规则计费。
> - 每个 SDKAppID 可免费体验2次，每次有效期均为7天；同时一个账号下所有 SDKAppID 的体验总次数为10次。
> - 新账号首次可前往 试用中心 免费领取10000分钟音视频时长。

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

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

## 开通正式版

AI 实时对话服务的费用 包含三个方面：音频通话费用、AI 实时对话服务费用以及语音转文字费用。其中，语音转文字功能需购买 AI 智能语音套餐包（轻量版/基础版/尊享版）或包月套餐尊享版 Plus /旗舰版 Plus 任一版本，才可使用，本章节将详细说明如何开通正式版的 AI 智能语音套餐包并进行续费。音频通话费用 以及 AI 实时对话服务无开通过程，在使用过程中将默认开启后付费。

1. 登录 实时音视频购买页 。
2. 根据实际业务需求选择配置。接入方案选择**无 UI 自定义开发**，以及选择需要购买 AI 智能语音套餐的 SDKAppID，购买后无法修改。




3. 套餐类型选择 AI 智能语音套餐，并选择您需要开通的套餐版本，AI 智能语音套餐详情可参见 AI 智能语音详细说明 。




4. 购买完成后，您可以回到 实时音视频 TRTC 控制台 ，在 时长包管理 页面查看具体的 AI 智能语音套餐的使用情况。开通完成后，您可以根据 集成指引 文档进行集成。




# 续费正式版

您可对 AI 智能语音套餐进行续费，具体操作如下：

1. 登录 续费管理控制台 ，选择您想要续费的 AI 智能语音套餐，单击**续费**。




## 2. 按照指示完成续费即可。



# 准备工作

最近更新时间：2026-05-25 20:04:01

腾讯 TRTC 推出 AI 实时对话测试平台，支持快速集成第三方 LLM 与 TTS 能力。无需编写代码，即可零门槛配置并跑通 AI 实时通话场景，助您大幅提升开发验证与业务部署效率。本文将指导您如何通过控制台快速完成 AI 对话配置。

## 登录控制台

登录控制台配置地址：实时音视频控制台 > 快速跑通 AI 实时对话 。

## 资源配置

快速跑通 AI 实时对话支持对接多家 LLM、TTS 厂商，单击**开始使用**即可在此快速配置、测试并集成AI实时对话服务，实现高效开发与部署。

> **说明：**
> - 单击 体验 AI 对话 Demo 体验 AI 对话的能力。
> - 使用快速跑通 AI 实时对话测试工具将产生用量，计费详情请参见 AI 实时对话计费说明 。




## 步骤 1：基础配置

### 1. 选择应用

在**基础配置**页，您首先需要选择需要测试的应用。您可通过**领取体验版**或 **购买 AI 智能语音套餐包** 开通应用的 STT 服务来使用 STT 语音识别功能。




**示例：**

**当前应用未开通 STT 服务，可点击领取体验版，在弹框中免费领取。**




## 2. STT 配置项

我们提供了默认参数，您可以根据需求对 STT 配置项进行微调。配置完成后单击**下一步**。

- 填写机器人欢迎语。
- 选择打断模式。可基于特定规则自动打断机器人讲话，该模式下同时支持自动打断和手动打断，具体描述可参见 打断模式介绍 。
- 选择打断时长。

# 步骤 2：STT 语音识别配置

STT 语音识别配置模型仅可选择腾讯 STT，支持选择多种语言，VAD 时长可调整，设置范围为240 - 2000，更小的值会让语音识别分句更快。配置完成后单击**下一步**。

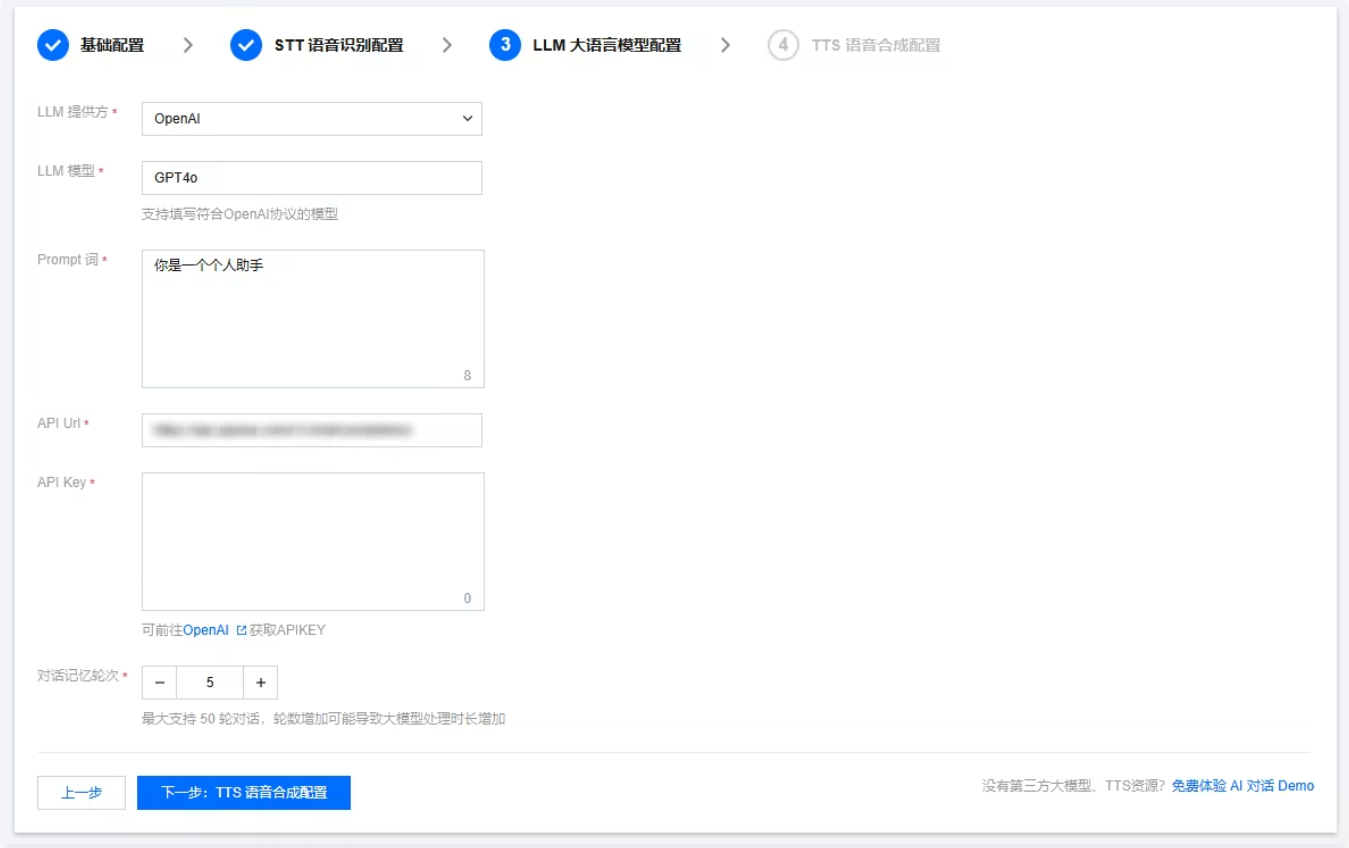



# 步骤 3：LLM 大语言模型配置

在本页面需要您自行申请第三方资源进行配置：LLM 提供方支持 OpenAI、Deepseek、MiniMax、混元、Coze 和 Dify。

## OpenAI

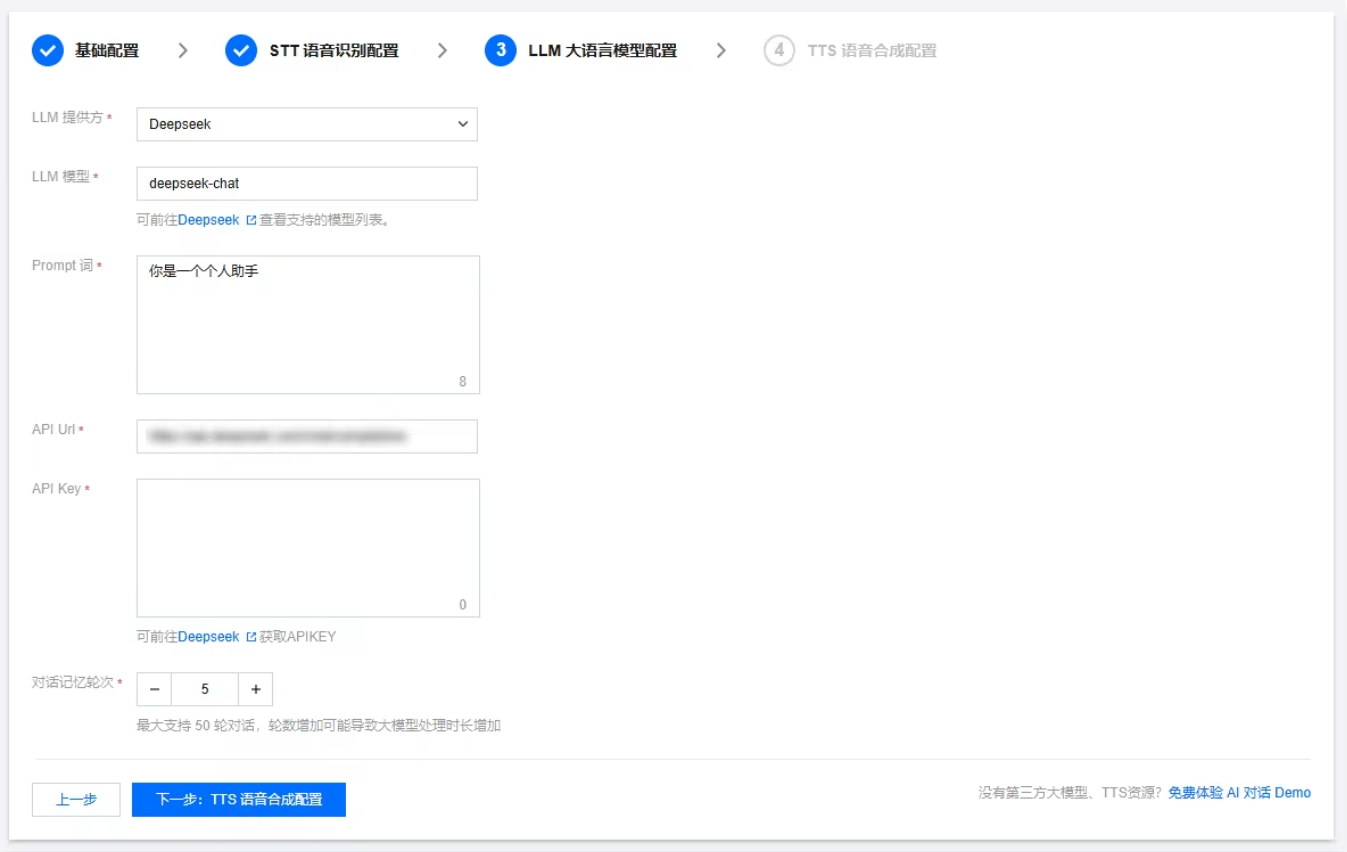
LLM 提供方选择 OpenAI 时，此处支持任何符合 OpenAI 标准协议的 LLM 模型，您可以自行调整下方的模型、URL 和 Key 进行调用。当您选择使用 OpenAI 提供的模型时，API Key 可前往 OpenAI 获取。



## Deepseek

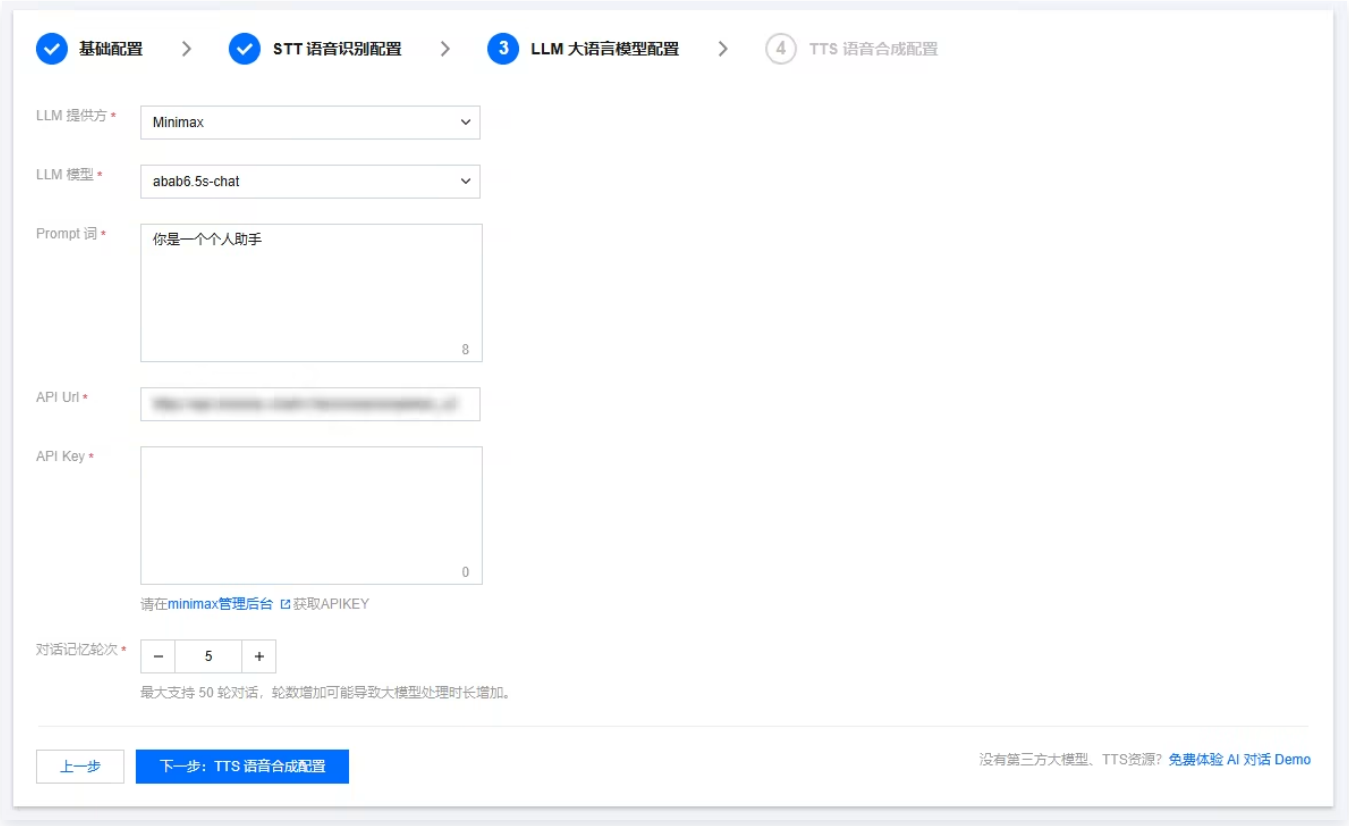
LLM 提供方选择 Deepseek 时，其他配置我们提供了默认参数，您可根据需求进行微调；LLM 模型可前往 Deepseek 查看支持的模型列表，API Key 可前往 Deepseek 获取。
除了Deepseek官方，您还可以选择调用第三方云平台部署的 Deepseek 服务：

- **腾讯云大模型知识引擎**
  - 调用 指引文档 。
  - LLM 模型名称填写：deepseek-v3 或 deepseek-r1（由于模型有思考过程，对话的实时性会有影响）。
  - API URL 填写：`https://api.lkeap.cloud.tencent.com/v1/chat/completions`
  - API Key 参考文档获取 。
- **硅基流动平台**
  - 调用 指引文档 。
  - LLM 模型名称 参考文档 。
  - API URL 填写：`https://api.siliconflow.cn/v1/chat/completions`
  - API Key 获取地址 。



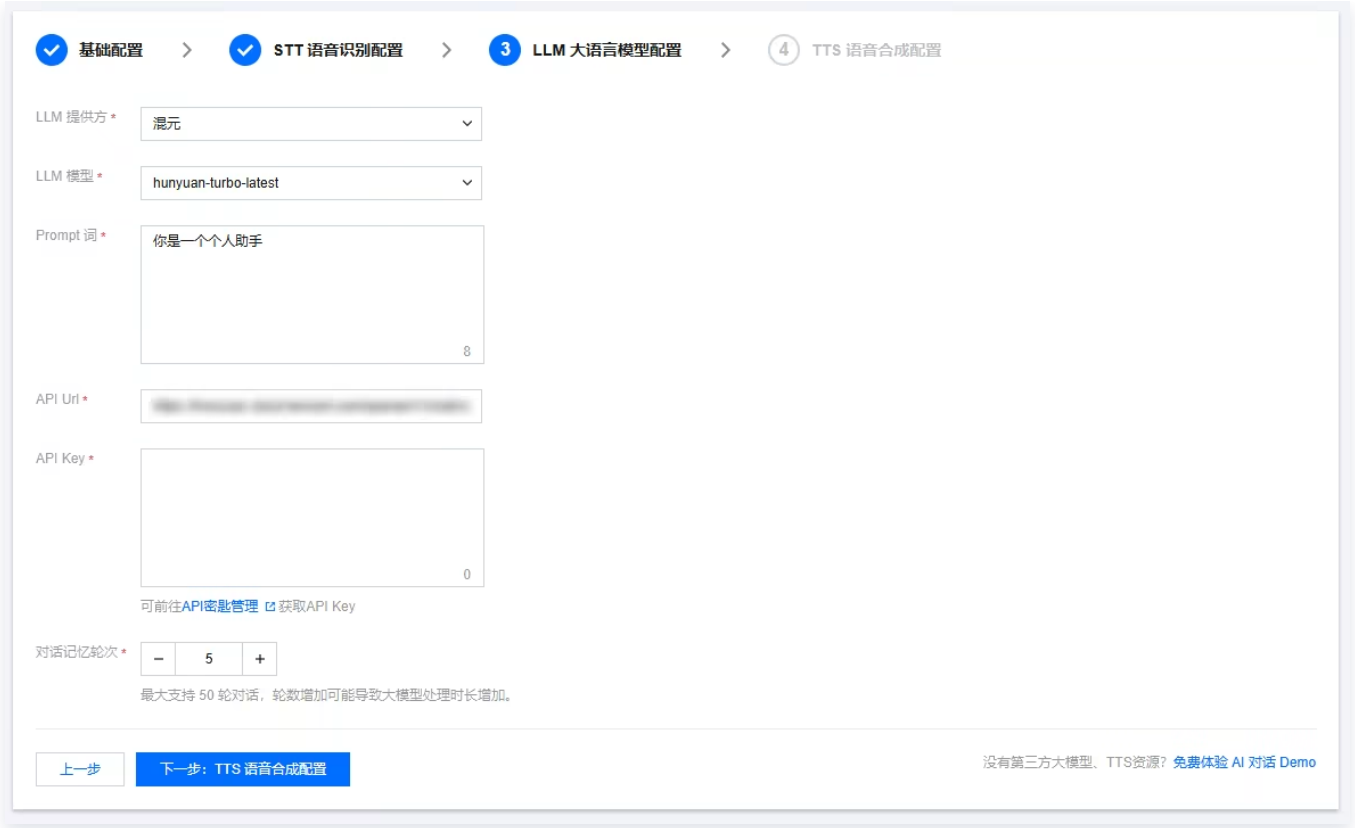

## MiniMax

LLM 提供方选择 MiniMax 时，其他配置我们提供了默认参数，您可根据需求进行微调；LLM 模型可下拉选项自由选择，API Key 可前往 MiniMax 管理后台 获取。



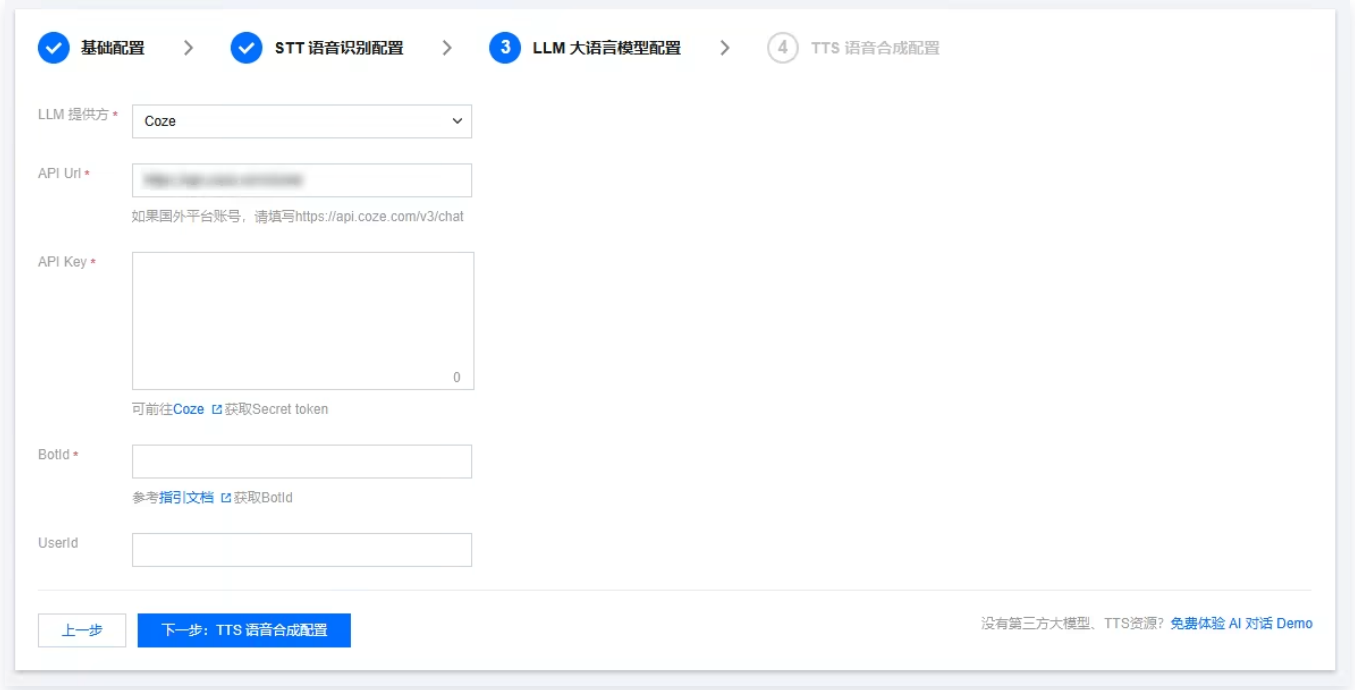

## 混元

LLM 提供方选择混元时，其他配置我们提供了默认参数，您可根据需求进行微调；LLM 模型可下拉选项自由选择，API Key 可前往 API 密钥管理 获取。



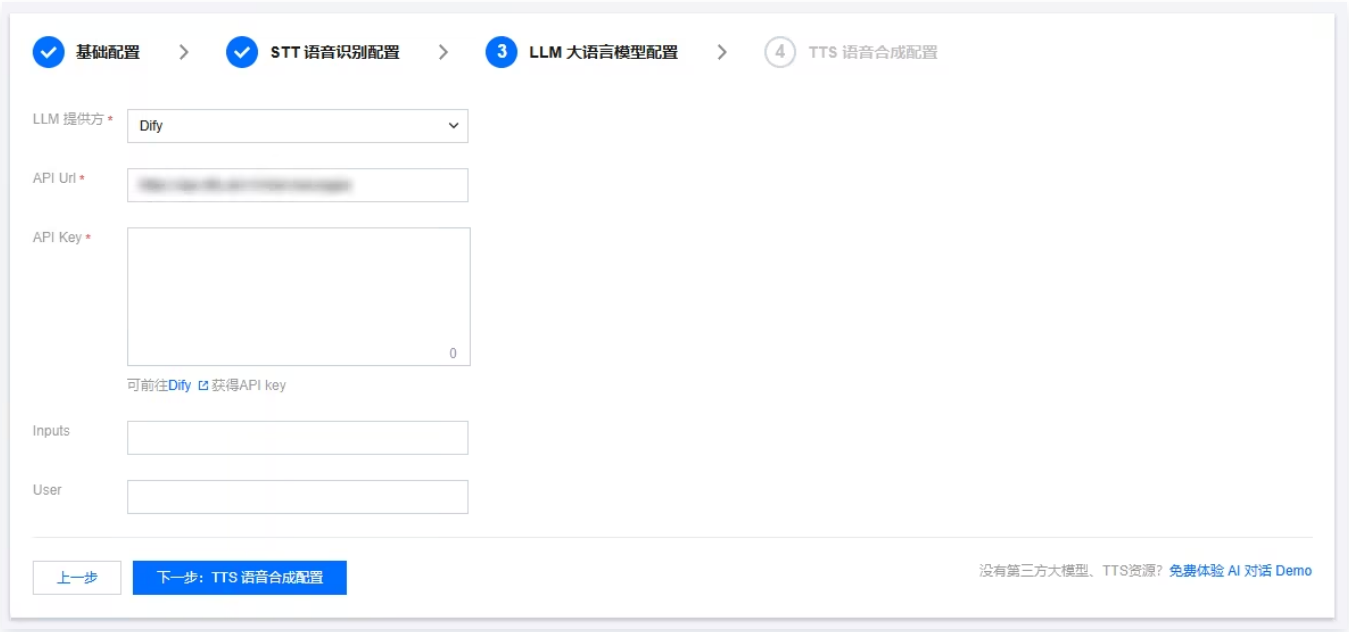

## Coze

**LLM 提供方选择 Coze 时，其他配置我们提供了默认参数，您可根据需求进行微调；API Key 可前往 Coze 获取 Secret token，参见 指引文档 获取 BotId。**



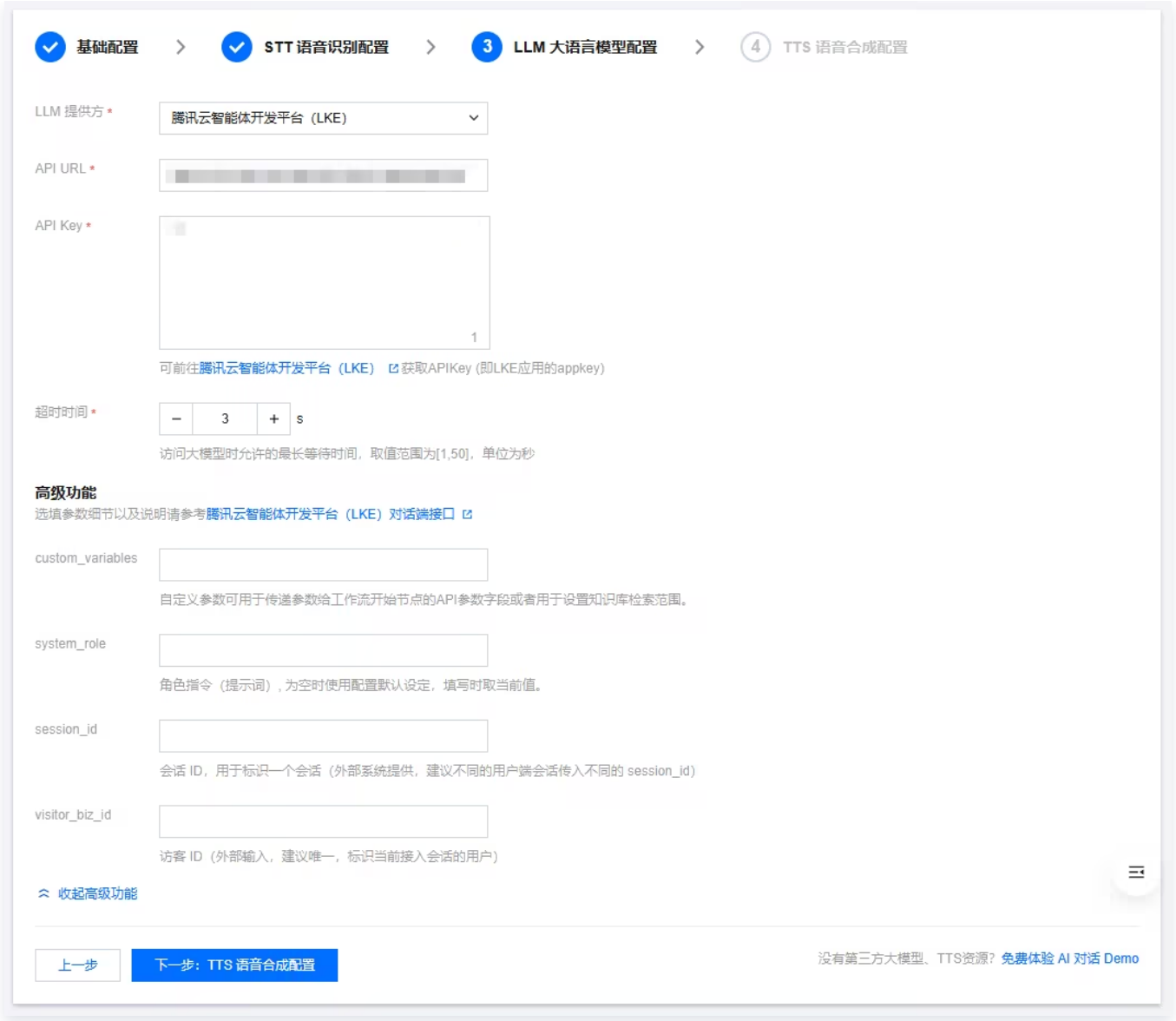
## Dify

LLM 提供方选择 Dify 时，其他配置我们提供了默认参数，您可根据需求进行微调；API Key 可前往 Dify 获取。




## 腾讯云智能体开发平台（LKE）

LLM 提供方选择腾讯云智能体开发平台（LKE）时，其他配置我们提供了默认参数，您可根据需求进行微调；API Key 可前往 腾讯云智能体开发平台（LKE） 获取，选填参数细节以及说明请参考 腾讯云智能体开发平台（LKE）对话端接口 。




配置完成后单击**下一步**。

# 步骤 4：TTS 语音合成配置

## TRTC TTS 服务

在 TTS 语音合成配置页面默认配置 TRTC 的 TTS 服务，无需额外配置 API Key 即可快速跑通。如果需接入第三方 TTS 请跳转至 配置第三方 TTS 服务 。

### TRTC

- TTS 提供方选择 TRTC_Flow_TTS 时为 TRTC 自有 TTS 服务。
- 无需额外开通账号和配置 API Key 和 API URL，快速跑通体验。
- 多种音色 ID 可供选择填入，其他配置我们提供了默认参数，您可根据需求进行微调。

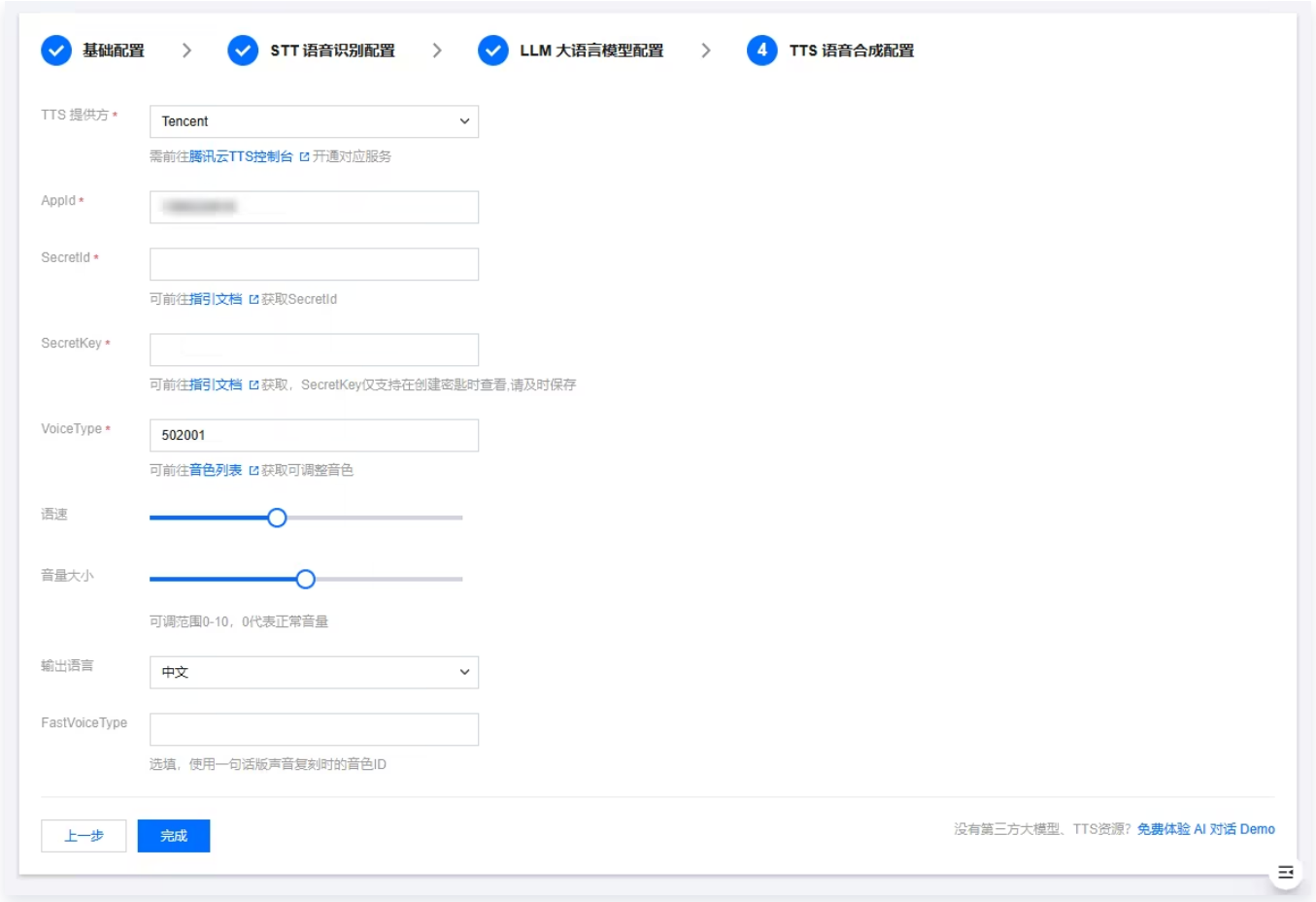



## 第三方 TTS

如果需要接入第三方 TTS：目前支持 Tencent、MiniMax、Azure、Cartesia、Elevenlabs 和 自定义。

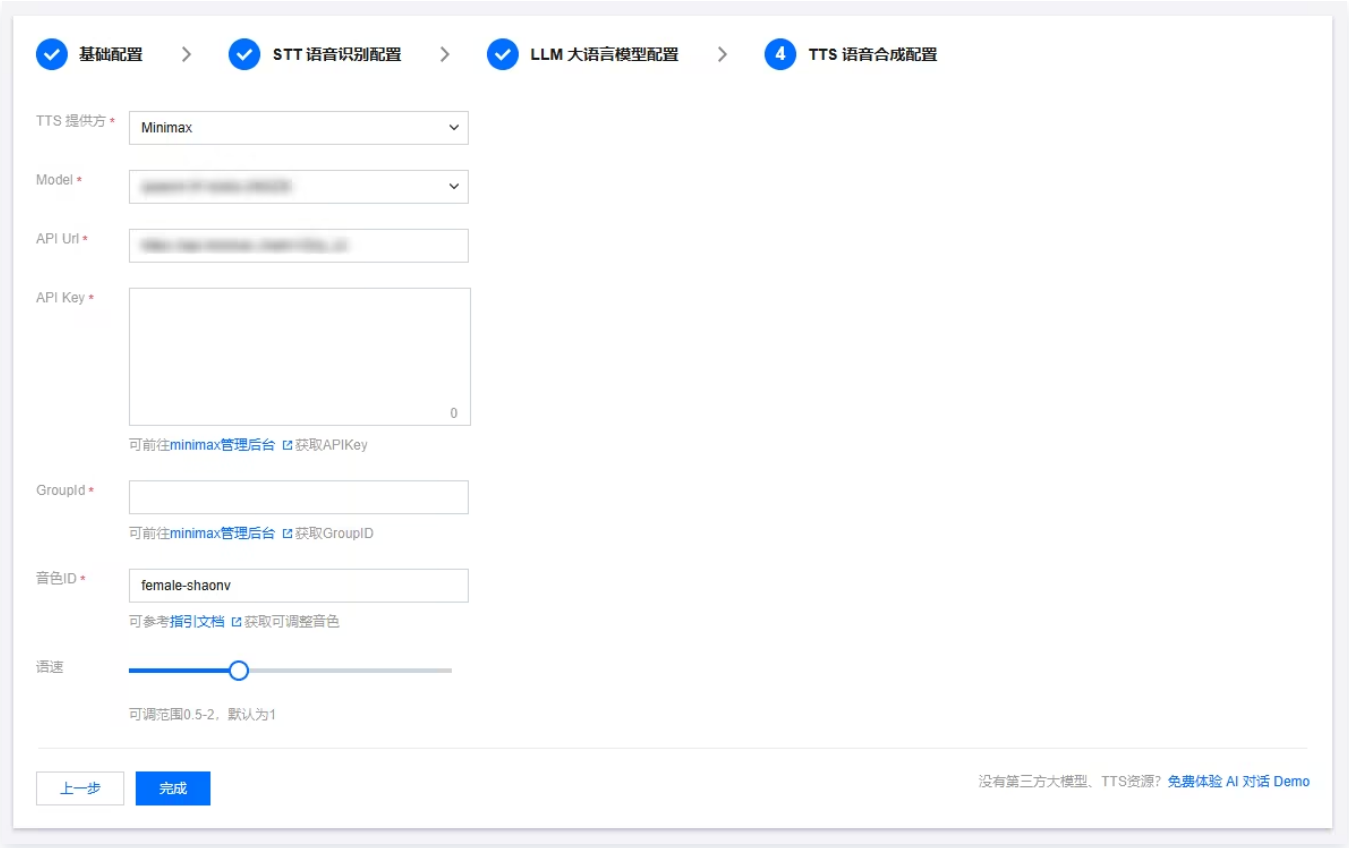
### Tencent

- TTS 提供方选择 Tencent 时，您需 开通应用的 TTS 服务 来使用 TTS 语音合成功能。您可以点击 免费领取语音合成资源包 。
- SecretId 可前往 指引文档 获取。
- SecretKey 可前往 指引文档 获取，SecretKey 仅支持在创建密钥时查看,请及时保存。
- 可前往 音色列表 获取可调整音色。
- 其他配置我们提供了默认参数，您可根据需求进行微调。



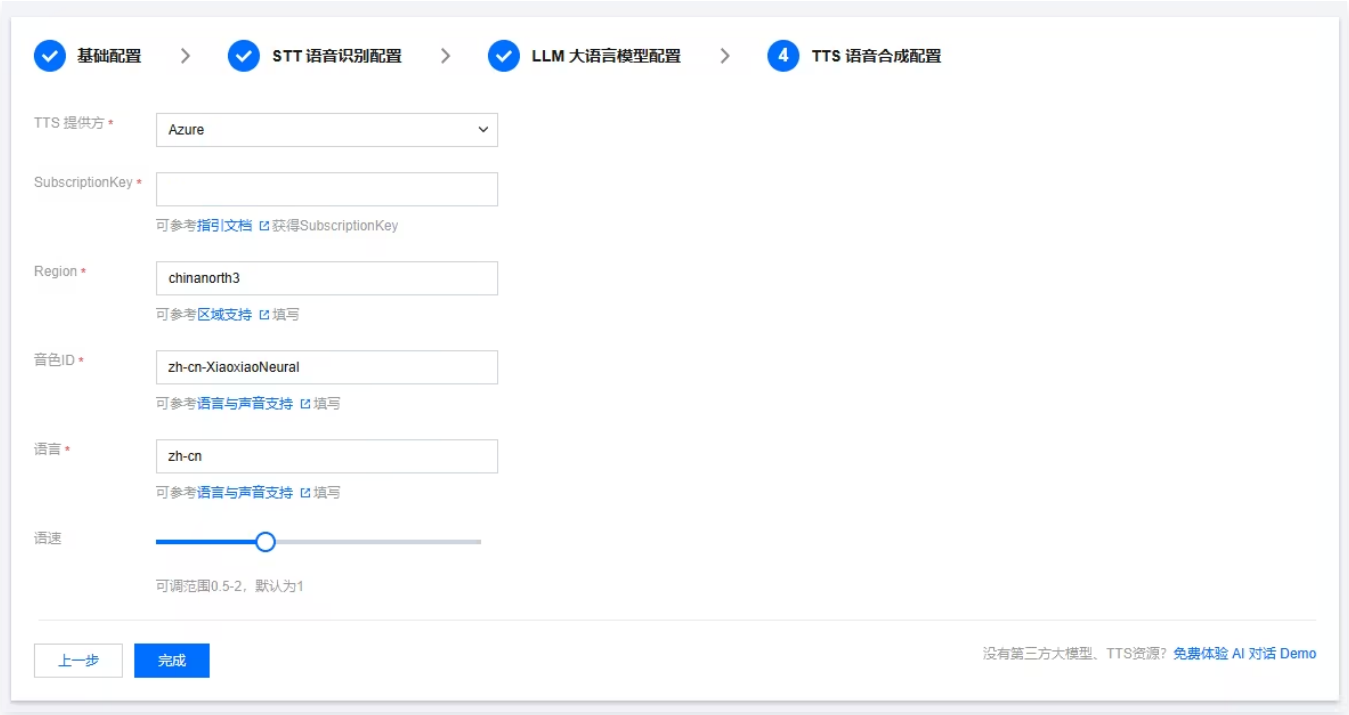

## MiniMax

- API Key 可前往 MiniMax管理后台 获取。
- GroupID 可前往 MiniMax管理后台 获取。
- 可参考 指引文档 获取可调整音色。
- 其他配置我们提供了默认参数，您可根据需求进行微调。



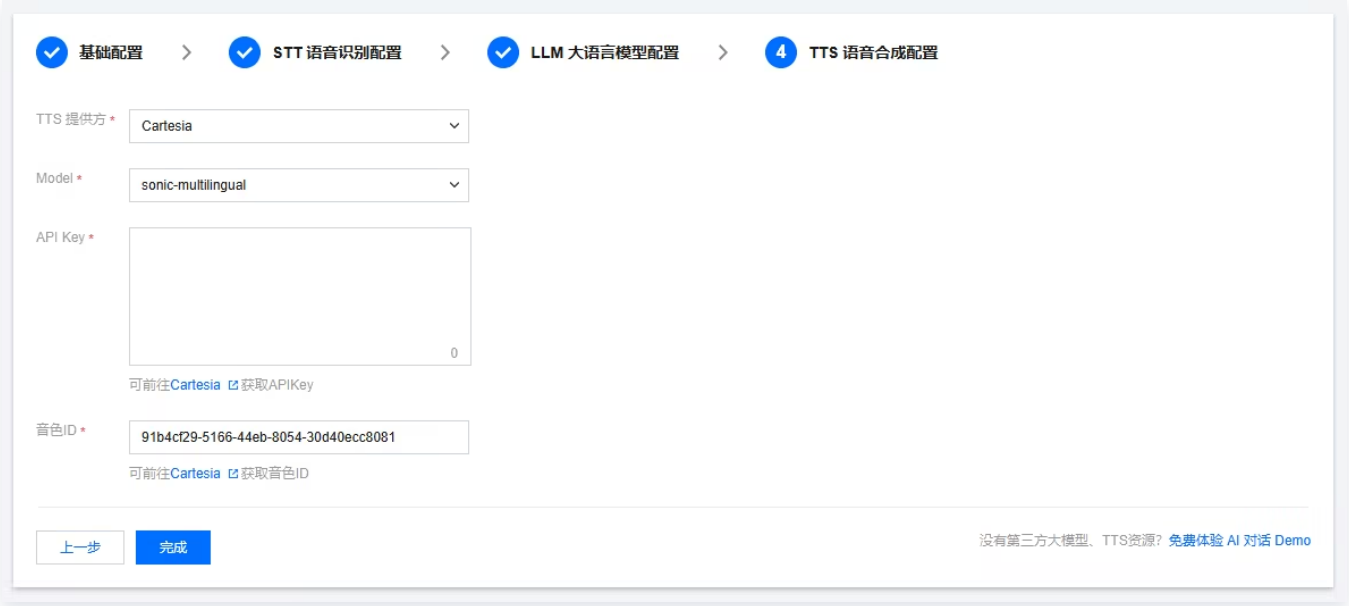

## Azure

- SubscriptionKey 可参考 **指引文档** 获得。
- Region 可参考 **区域支持** 填写。
- 语言可参考 **语言与声音支持** 填写。
- 其他配置我们提供了默认参数，您可根据需求进行微调。



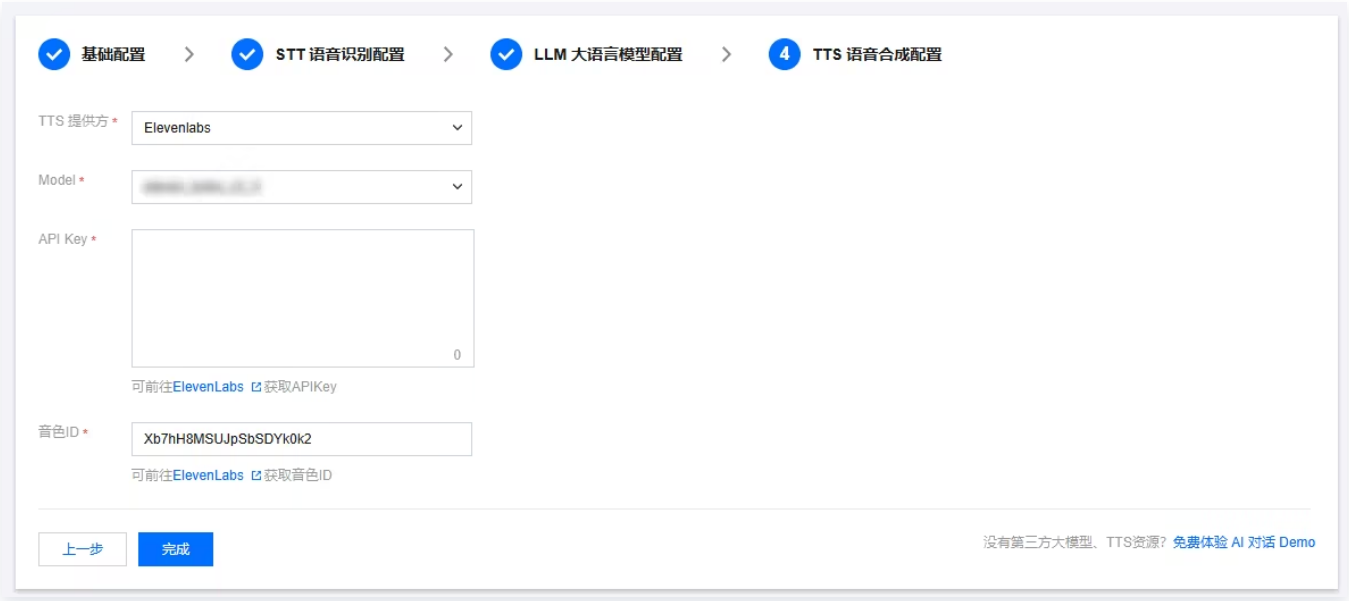

## Cartesia

- API Key 可前往 Cartesia 获取 API Key。
- 音色 ID 可前往 Cartesia 获取。
- 其他配置我们提供了默认参数，您可根据需求进行微调。



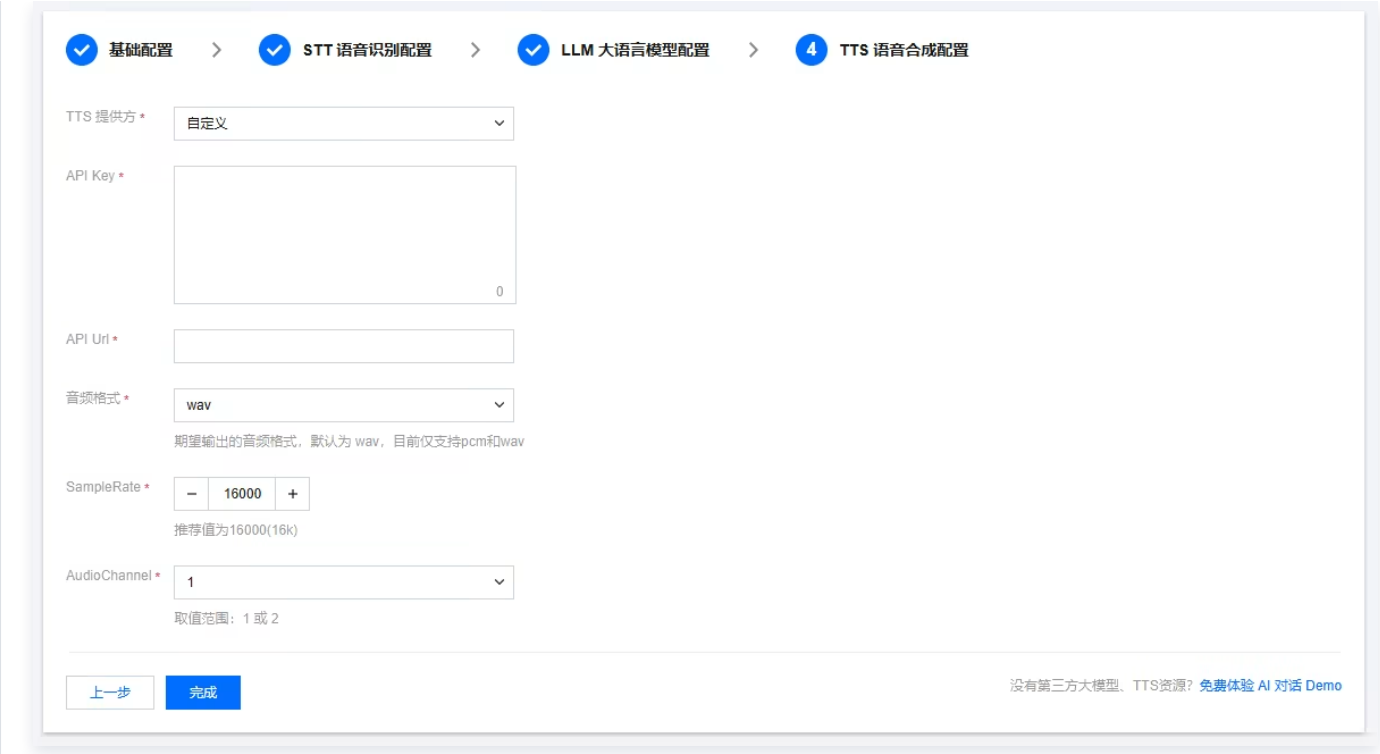

## Elevenlabs

- API Key 可前往 ElevenLabs 获取。
- 音色 ID 可前往 ElevenLabs 获取。
- 其他配置我们提供了默认参数，您可根据需求进行微调。




## 自定义

- 可参见 自定义 TTS 协议 配置。
- TTS 提供方选择自定义时，您可根据实际情况填写 API Key 和 API URL。
- 其他配置我们提供了默认参数，您可根据需求进行微调。

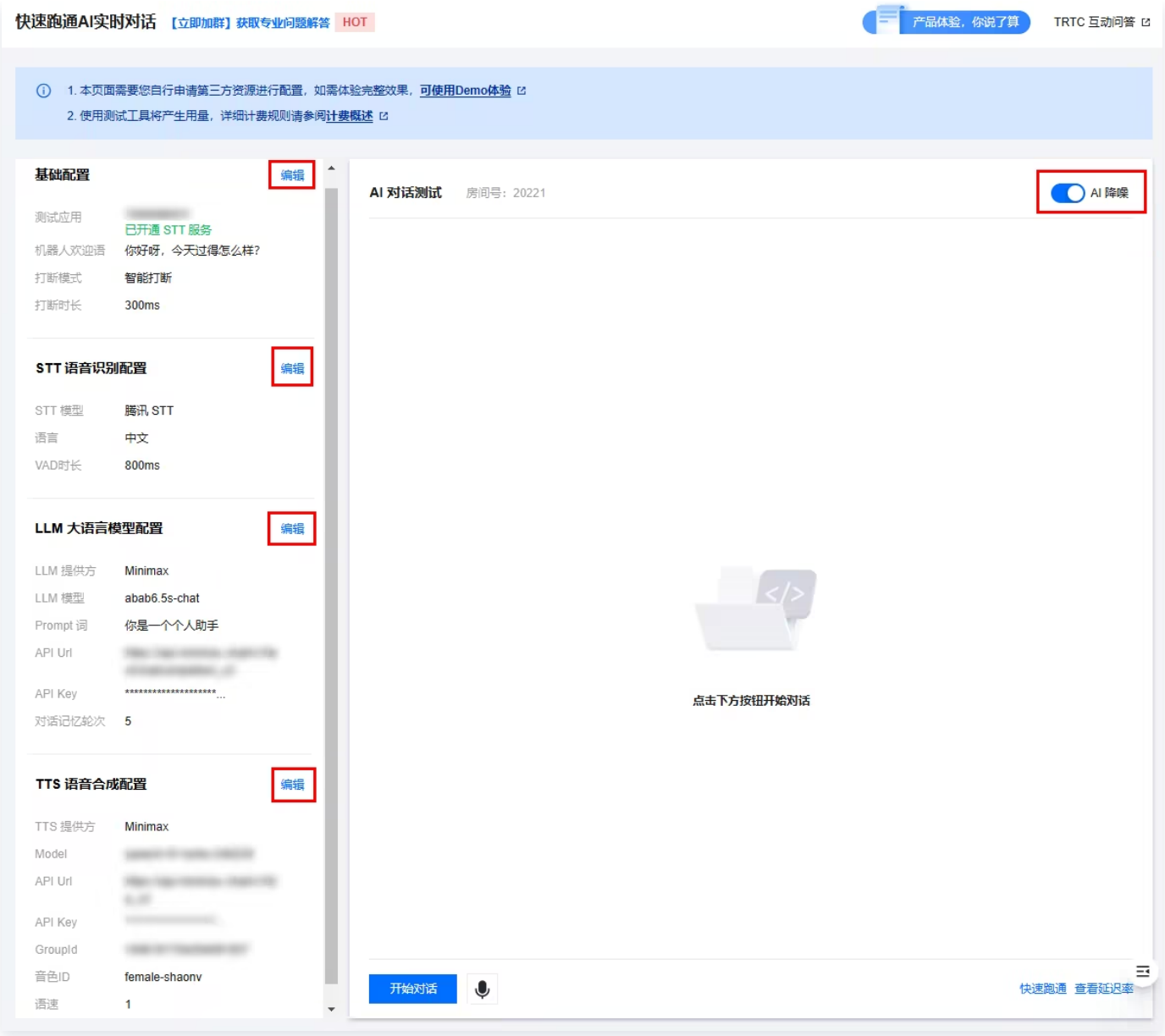



配置完成后单击**完成**。

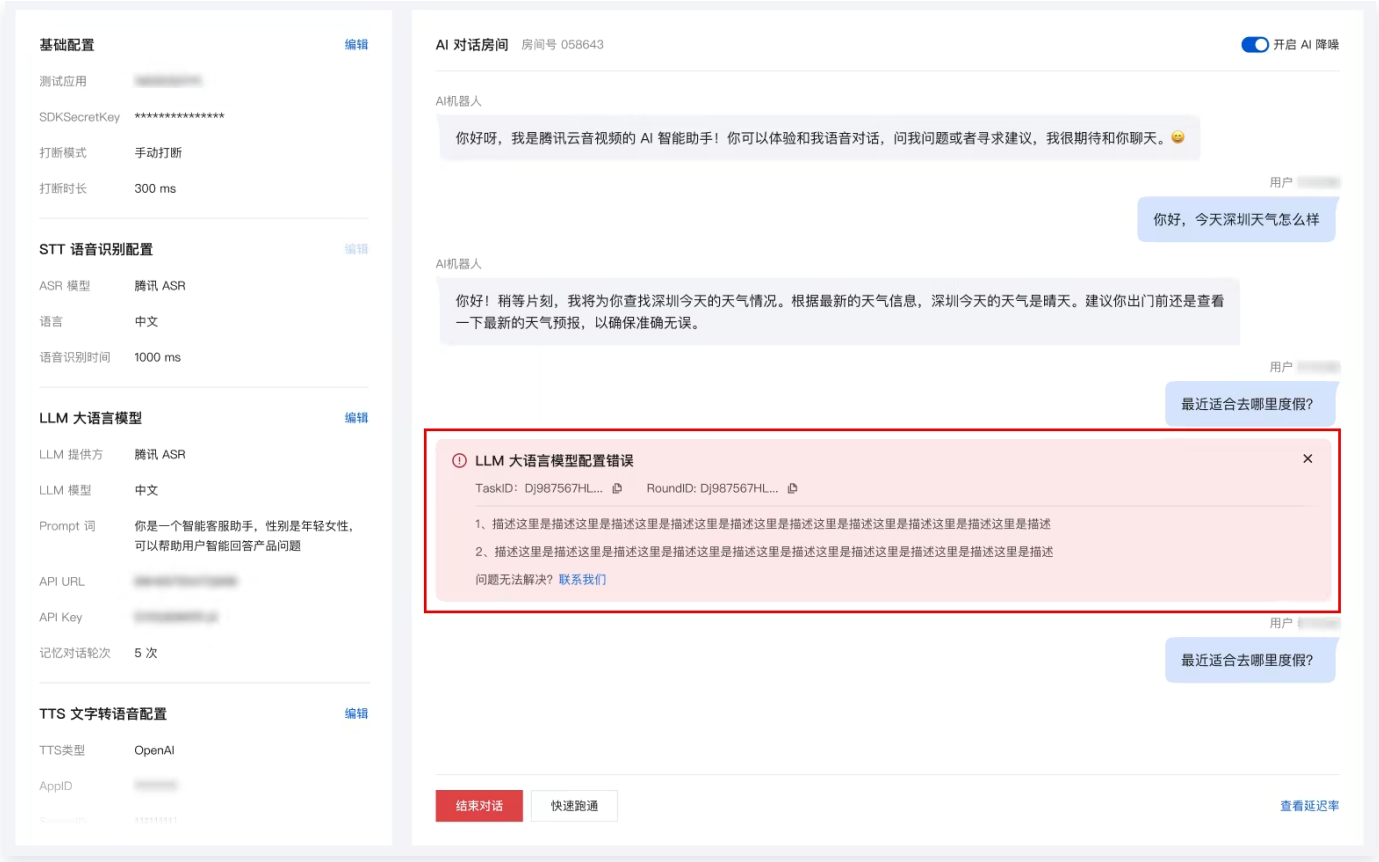
# 主页面介绍

## 开始对话

- AI 降噪：使用此功能需开通 实时音视频包月套餐包 尊享版或旗舰版。
- 对话过程中支持编辑修改打断时长、LLM 大语言模型配置和 TTS 语音合成配置，便于进行效果测评。

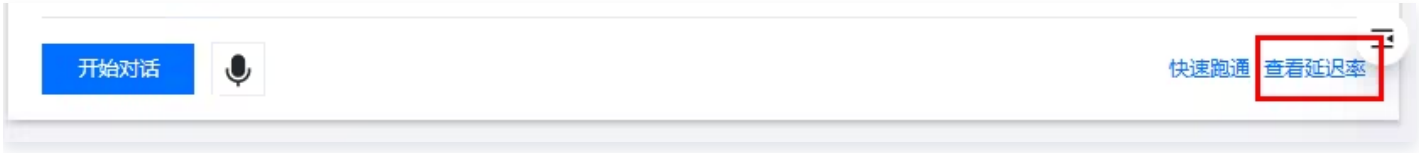



- **对话过程中若出现错误码的展示：请注意，错误码有些是阻塞对话进行的，请根据错误提示及时修改配置，如果无法解决，请复制 TaskID 和 RoundID 信息，提交工单 进行咨询。有一些错误码不影响对话正常进行，例如反应超时，您可以根据实际情况处理。**




## 查看延迟率

**您在对话时可随时查看延迟率，便于您选择低延迟模型。**



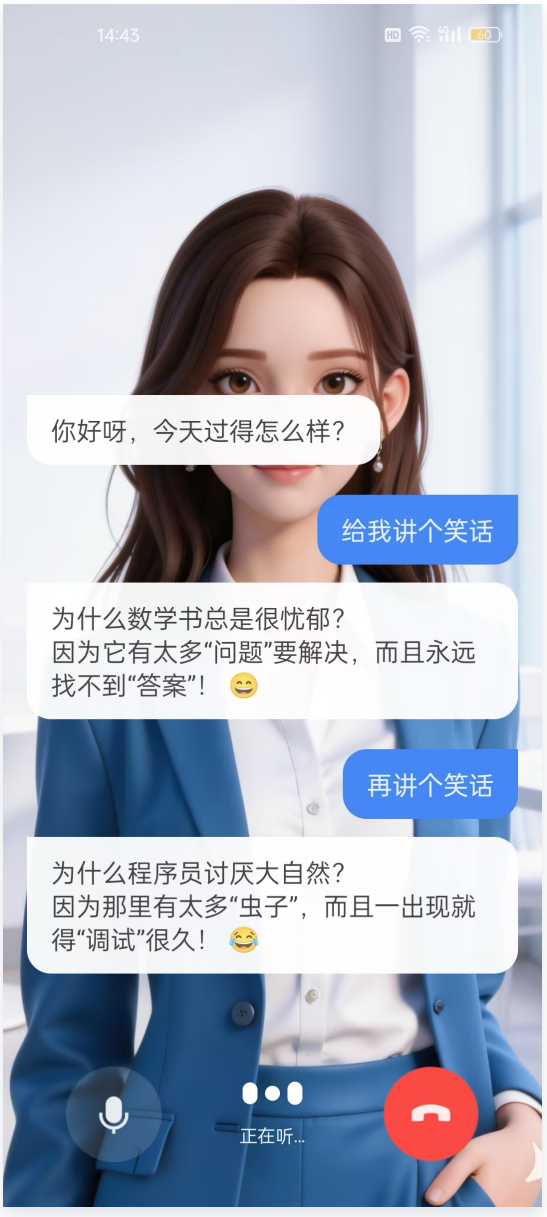

书申请验证流程繁琐？教你用边缘安全加速平台EO申请免费证书，限时活动套餐低至6.8元/月 查看详情 >
1. 本页面需要您自行申请第三方资源进行配置，如需体验完整效果，可使用Demo体验
2. 使用测试工具将产生用量，详细计费规则请参阅计费概述
STT 语音识别配置
编辑
STT 模型 腾讯 STT
语言 中文
VAD时长 800ms
LLM 大语言模型配置
编辑
LLM 提供方 Minimax
LLM 模型 abab6.5s-chat
Prompt 词 你是一个个人助手
API Url
API Key **********...
对话记忆轮次 5
TTS 语音合成配置
编辑
TTS 提供方 Minimax
Model speech-01-turbo-240228
API Url
API Key **********...
AI 对话测试
房间号：771664
你好呀，今天过得怎么样？
AI Bot
你好！有什么可以帮助你的吗?
AI Bot
你好！很高兴见到你。有什么我可以帮助你的吗?
AI Bot
你好！如果你有任何问题或需要帮助，请随时告诉我。我在这里为你服务。
聆听中
结束对话
延迟率
大模型延迟
当前延迟 413ms
13:54:03
延迟：1051ms
最高延迟 1051ms
平均延迟 602ms
最低延迟 342ms
TTS 延迟
当前延迟 438ms
最高延迟 861ms
平均延迟 582ms
最低延迟 435ms

# 跑通 Demo
# 跑通 Demo（Android）

最近更新时间：2026-03-16 18:02:44

**本文档主要介绍如何快速跑通 AIConversationKit 示例工程，体验高质量的 AI 对话。跟随本文档，您可以在 20 分钟内跑通 Demo，并最终体验一个包含完备 UI 界面的 AI 对话。**

**AI 对话界面**




## 环境准备

- Android 5.0 (SDK API level 21)及以上。

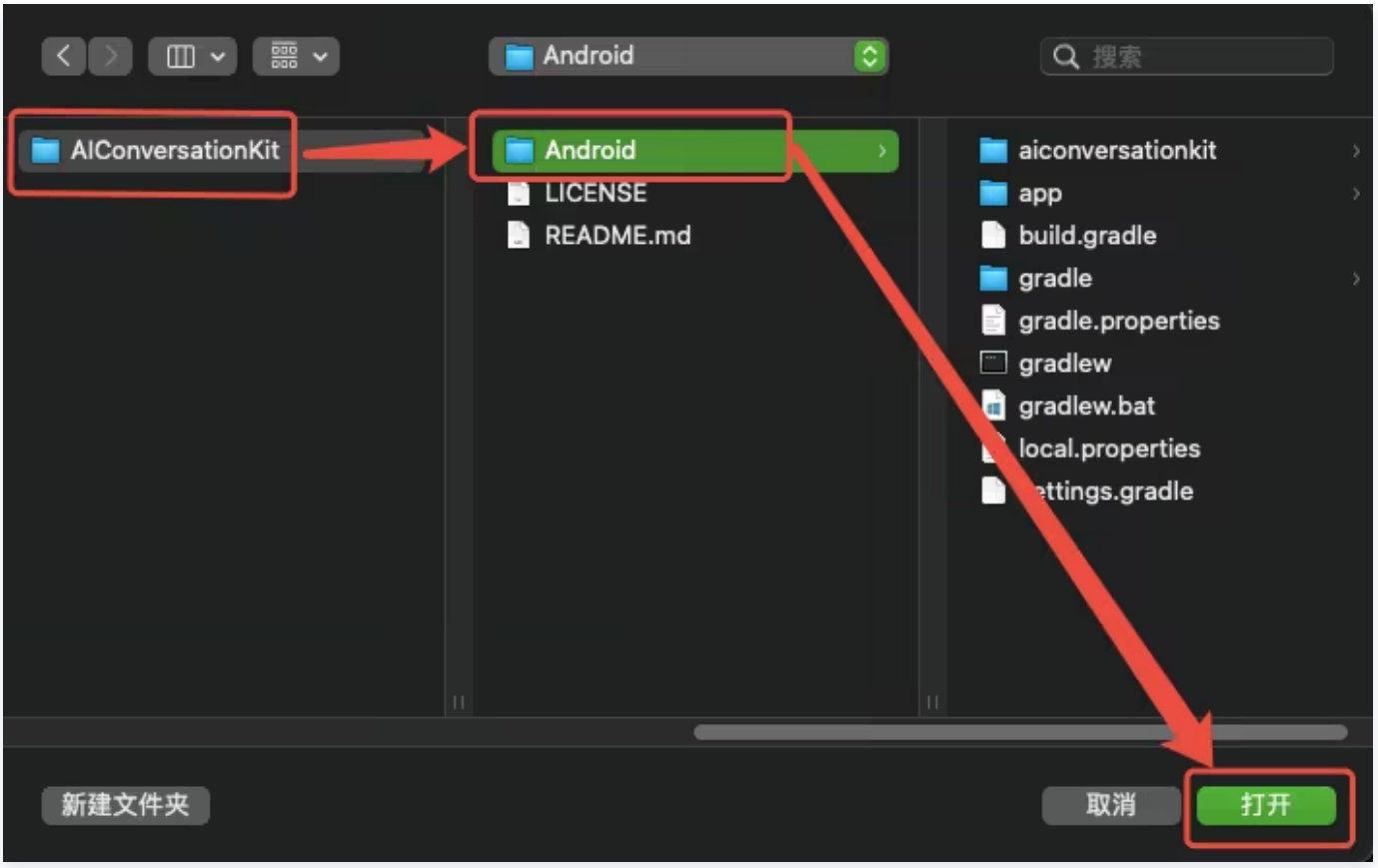
- Android Studio 3.5及以上版本。

## 下载 Demo

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

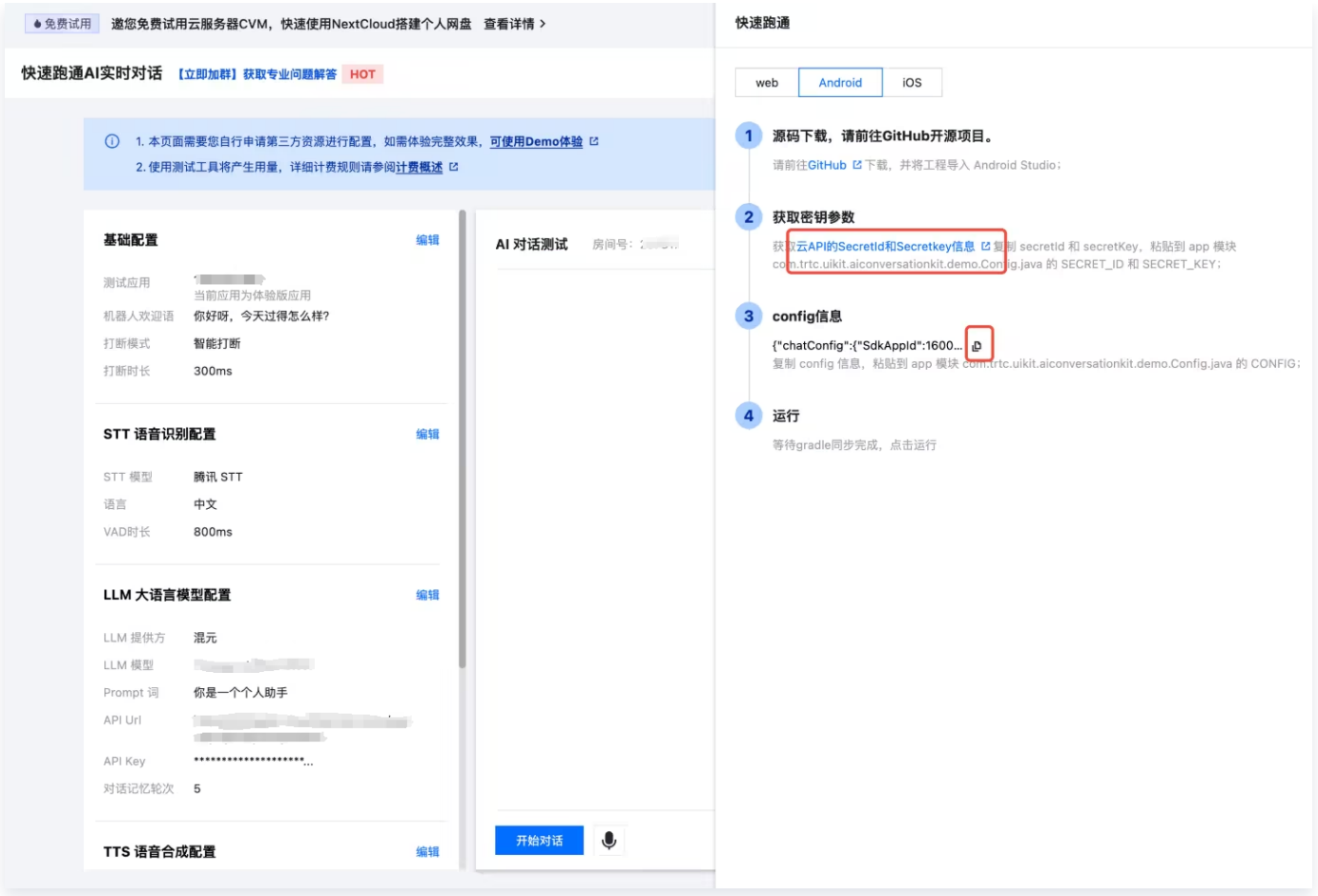
```
git clone https://github.com/Tencent-RTC/AIConversationKit.git
```

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




## 配置 Demo

1. 请到 开通服务 创建应用，领取 7 天免费试用版本。
2. 请到 准备工作 配置 AI 对话后台参数，点击右下角的**快速跑通**，切换到 Android，获取 SecretId、SecretKey 和 Config 参数。

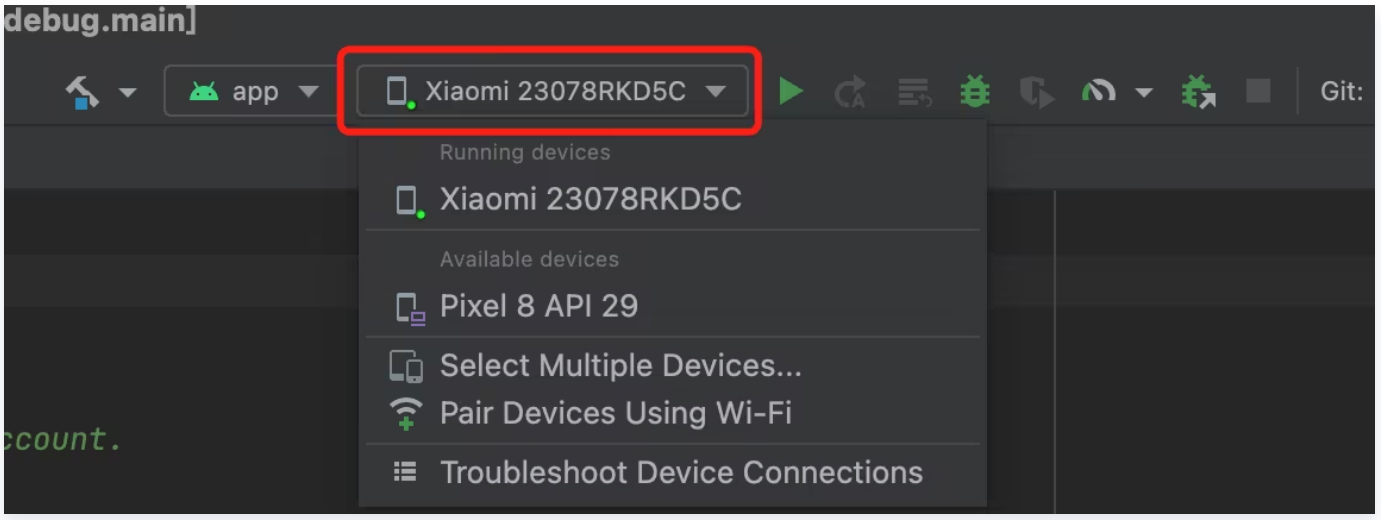



3. 打开工程，在工程内找到 `Android/app/src/main/java/com/trtc/uikit/aiconversationkit/demo/Config.java` 文件。将上一步中获取到的对应的 SecretId、SecretKey 和 Config 填入其中：

```
public class Config {
    public static final String SECRET_ID  = "";
    public static final String SECRET_KEY = "";
    public static final String CONFIG     = "";
}
```

# 跑通 Demo

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




2. 选择完成后单击运行，将 AIConversationKit Android Demo 运行到目标设备上。

## 开始您的第一次对话

开始和 AI 说话，比如让 AI 讲个笑话。


14:43
你好呀，今天过得怎么样？
给我讲个笑话
为什么数学书总是很忧郁？
因为它有太多“问题”要解决，而且永远找不到“答案”！
再讲个笑话
为什么程序员讨厌大自然？
因为那里有太多“虫子”，而且一出现就得“调试”很久！
正在听...

# 跑通 Demo（iOS）

最近更新时间：2026-03-16 18:02:50

**本文档主要介绍如何快速跑通 AIConversationKit 示例工程，体验高质量的 AI 对话。跟随本文档，您可以在 20 分钟内跑通 Demo，并最终体验一个包含完备 UI 界面的 AI 对话。**

**AI 对话界面**




## 环境准备

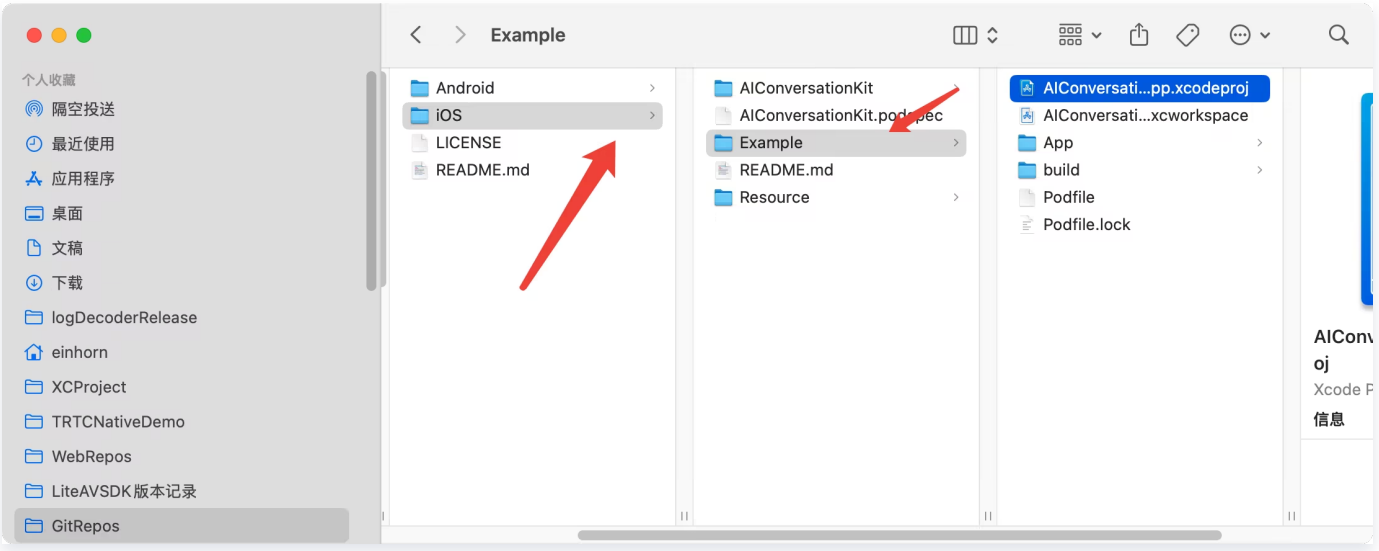
- **最低兼容 iOS 13，建议使用 iOS13 及以上版本。**
- **Xcode 13及以上版本。**

# 下载 Demo

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

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

2. 解压文件夹后，使用 Mac 的终端应用，cd 到 Example 下执行 pod install。

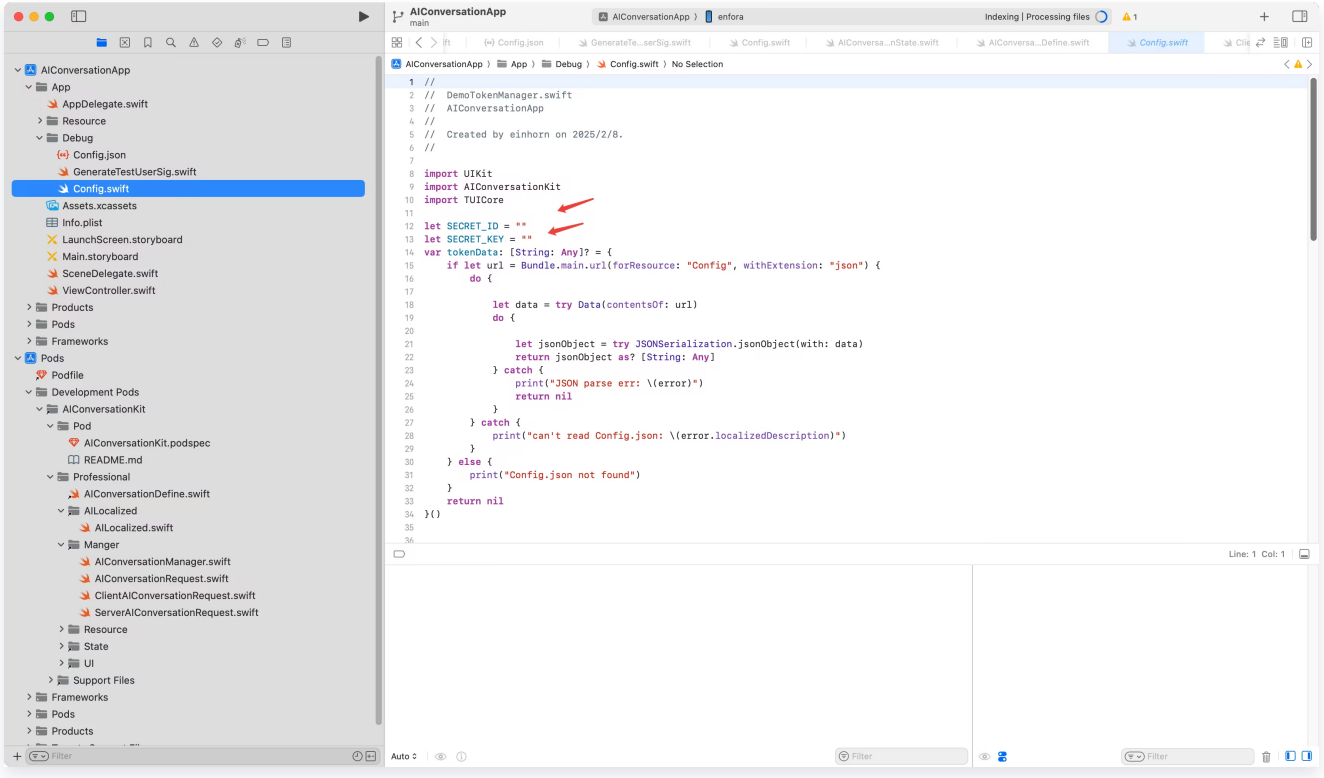



# 配置 Demo

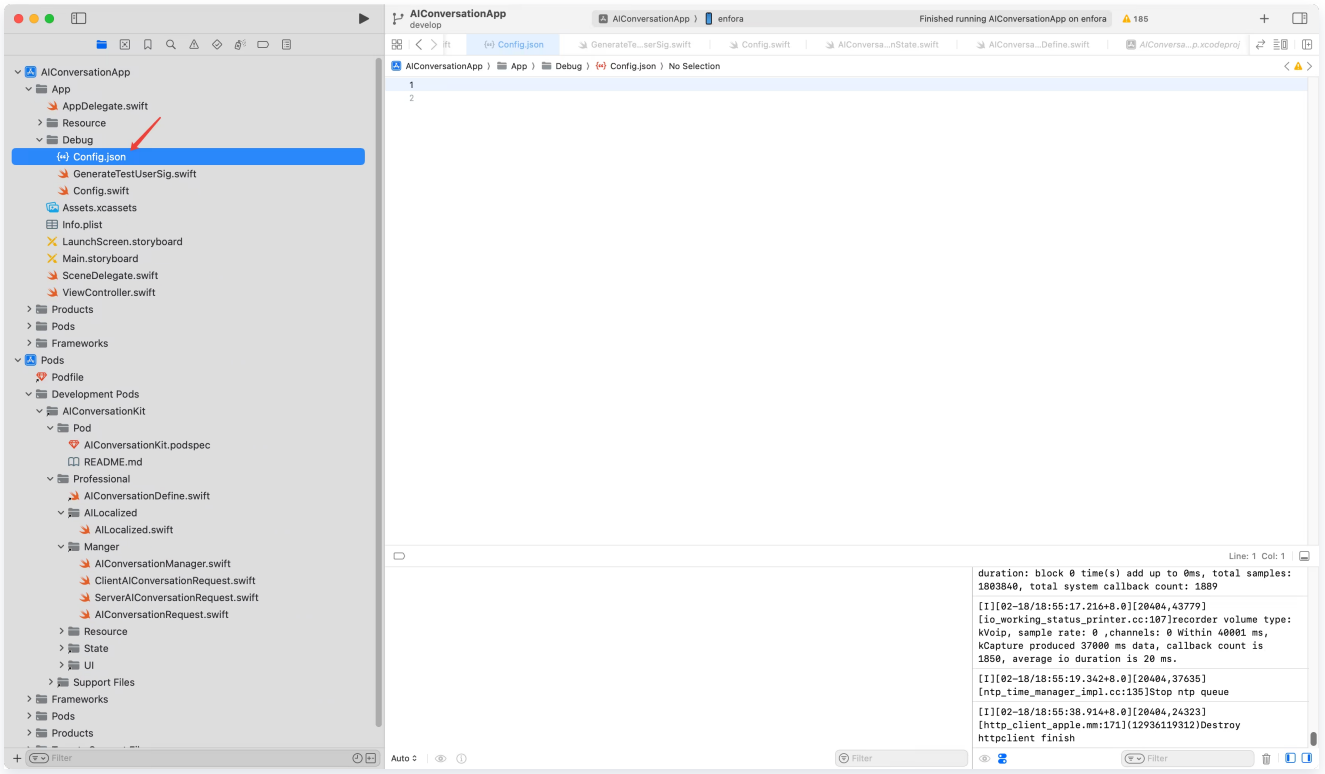
1. 请到 开通服务 创建应用，领取 7 天免费试用版本。
2. 请到 准备工作 配置 AI 对话后台参数，点击右下角的**快速跑通**，切换到 iOS，获取 SecretId、SecretKey 和 Config 参数。




3. 使用 Xcode 打开 AIConversationApp.xcworkspace 工程，在工程内找到 App/Debug/Config.swift：
   - 先将上一步中获取到的对应的 **SecretId**、**SecretKey** 填入其中箭头处：

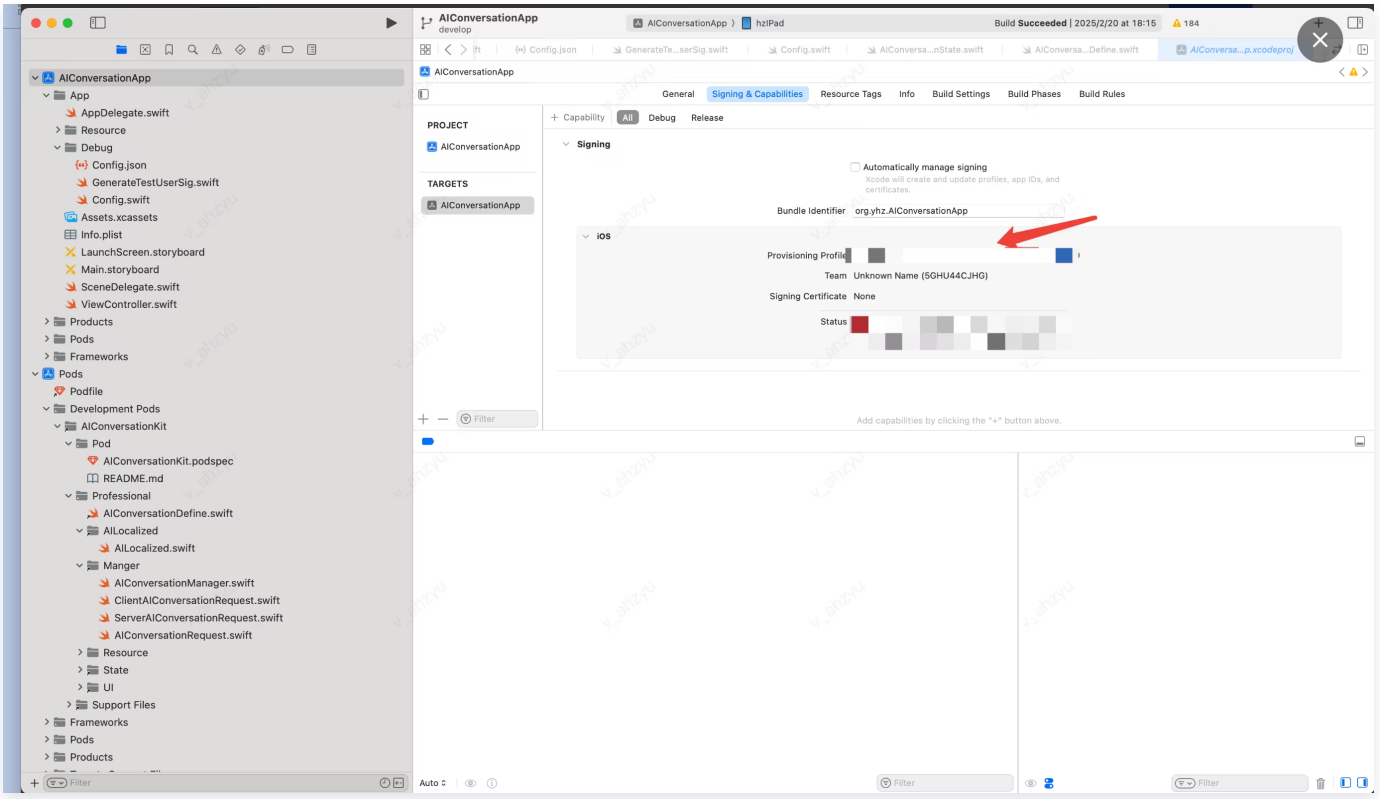


- 然后复制 config 信息到 App/Debug/Config.json。

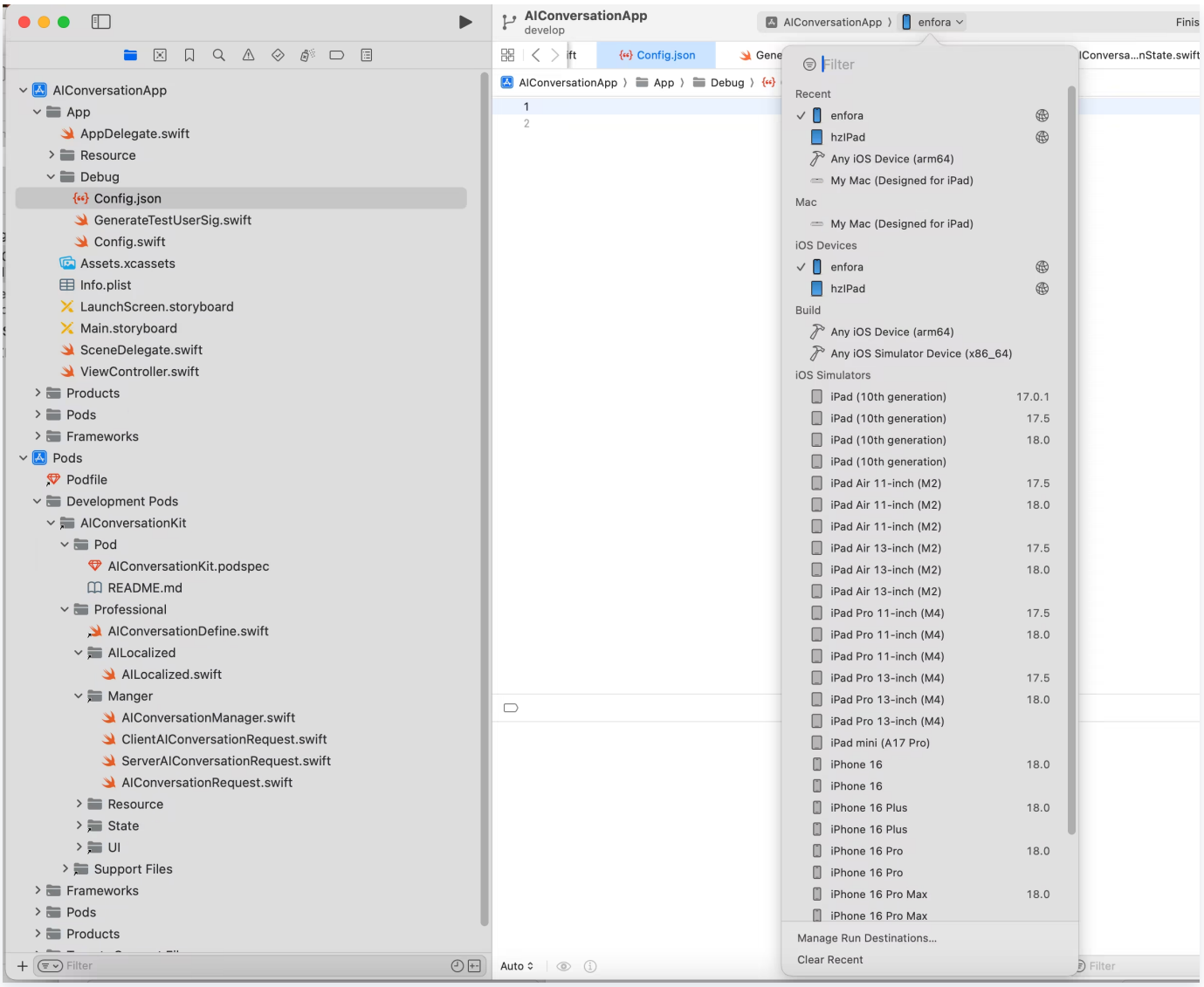



# 跑通 Demo

## 1. 配置证书：




## 2. 选择可以运行的设备。




3. 使用 `cmd + r` 运行 Demo。

# 开始您的第一次对话

开始和 AI 说话，例如让 AI 讲个笑话。


14:43
你好呀，今天过得怎么样？
给我讲个笑话
为什么数学书总是很忧郁？
因为它有太多“问题”要解决，而且永远找不到“答案”！ 😄
再讲个笑话
为什么程序员讨厌大自然？
因为那里有太多“虫子”，而且一出现就得“调试”很久！ 😂
正在听...

# 跑通 Demo（Web）

最近更新时间：2026-03-16 18:02:56

本文档主要介绍如何快速跑通 AIConversationKit 示例工程，体验高质量的 AI 对话。跟随本文档，您可以在20分钟内跑通 Demo，并最终体验一个包含基础功能 UI 界面的 AI 对话。

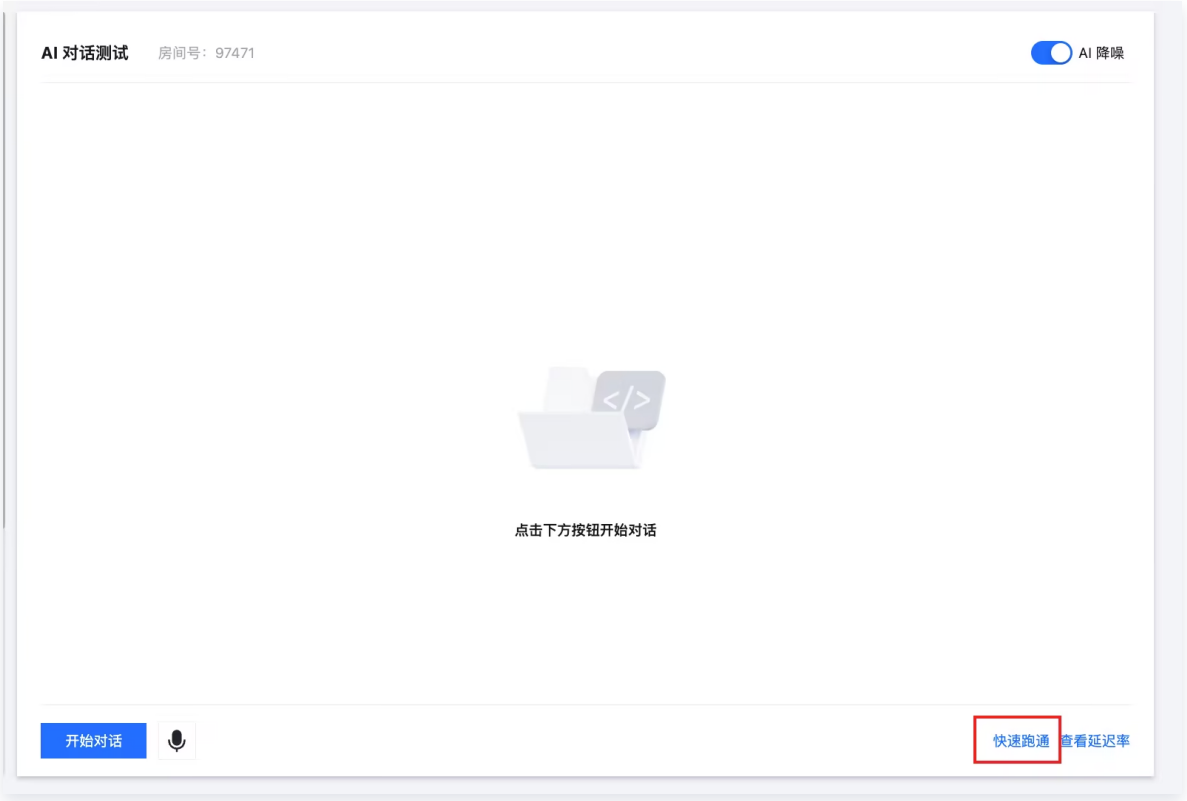



## 配置 Demo

1. 请先到 应用管理 创建应用，领取7天免费试用版本。




2. 使用 Playground 配置 AI 对话参数，点击右下角的**快速跑通**，选择 **web**，获取 云 API 的 SecretId 和 Secretkey 信息，按照顺序复制粘贴代码到本地。


AI 对话测试
房间号：97471
AI 降噪
点击下方按钮开始对话
开始对话
快速跑通
查看延迟率

web | Android | iOS

1 获取密钥参数

获取云API的SecretId和Secretkey信息

2 运行nodejs代码

```
/**
 * 腾讯云 TRTC 接口封装
 * 请先下载依赖：npm i express tencentcloud-sdk-nodejs-trtc
 * 启动：node index.js
*/
const express = require('express');
const tencentcloud = require("tencentcloud-sdk-nodejs-trtc");

const TrtcClient = tencentcloud.trtc.v20190722.Client;

const clientConfig = {
  credential: {
    secretId: "填写你的secretId",
    secretKey: "填写你的secretKey",
  },
  region: "ap-beijing",
  profile: {
    httpProfile: {
      endpoint: "trtc.tencentcloudapi.com",
    },
  },
};
```

3 前端代码

```
async function StartAIConversation(data) {
  return await fetch("http://localhost:3000/start-conversation", {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    body: data,
  }).then(res => res.json())
}

async function StopAIConversation(data) {
  return await fetch("http://localhost:3000/stop-conversation", {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    body: data,
  }).then(res => res.json())
}

async function startConversation() {
```

4 运行前端代码

# 跑通 Demo

1. 在运行服务端代码时，请先安装相关依赖，并替换对应的 secretId 和 secretKey。

```
npm i express tencentcloud-sdk-nodejs-trtc
```

2. 运行前端代码。
3. 开始您本地的 AI 对话。

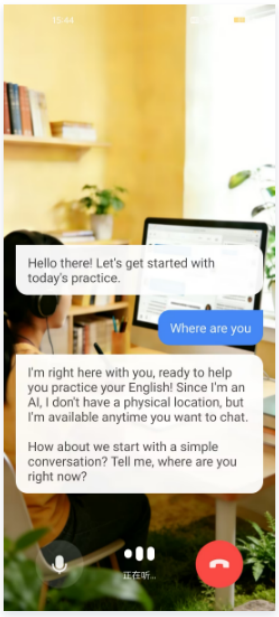


# 全功能集成
# 全功能集成（Android）

最近更新时间：2026-03-17 11:34:39

## 功能预览

通过集成 `AIConversationKit` 组件，您可为产品快速集成前沿的 AI 对话能力，实现流畅自然的智能交互体验，显著提升用户活跃度与满意度。

| 在线教育 | 智能客服 | 情感陪伴 |
| --- | --- | --- |
|  |  |  |



## 准备工作

### 开通服务

使用 `AIConversationKit` 前，请先参考 开通服务，领取体验版或开通付费版。

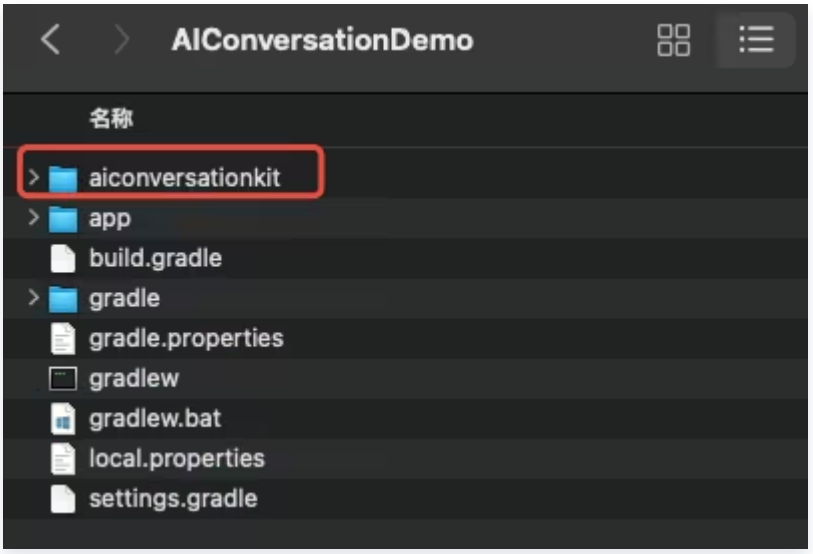
### 环境要求

- 系统版本：`Android` 5.0 (SDK API level 21)及以上
- 开发环境：Android Studio 3.5+（Gradle 3.5.4+）
- 设备要求：Android 5.0+ 的真机设备

## 代码集成

### 步骤 1：下载 AIConversationKit 组件

从 GitHub 克隆/下载代码，将 `Android/aiconversationkit` 目录复制到您的工程中（与 app 目录同级）。




## 步骤 2：工程配置

1. 在工程根目录的 `settings.gradle`（或 `settings.gradle.kts`）文件中添加以下代码，它的作用是将 `aiconversationkit` 组件导入到您当前的项目中。

**settings.gradle**

```
include ':aiconversationkit'
```

**settings.gradle.kts**

```
include (":aiconversationkit")
```

2. 在 app 目录的 `build.gradle`（或 `build.gradle.kts`）文件中添加以下代码，它的作用是声明当前 `app` 对新加入的 `aiconversationkit` 组件的依赖。

**build.gradle**

```
api project(':aiconversationkit')
```

**build.gradle.kts**

```
api(project(":aiconversationkit"))
```

3. 在 proguard-rules.pro 文件中添加以下代码，因为我们在 SDK 内部使用了Java 的反射特性，需要将 SDK 的部分类加入不混淆名单。

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

4. 为了防止编译时，`AndroidManifest` 合并过程中产生属性冲突，您需要在 `app/src/main/AndroidManifest.xml` 文件的 `<application>` 节点中，添加 `tools:replace="android:allowBackup"` 和 `android:allowBackup="false"`，用以覆盖组件内的设置。

```
// app/src/main/AndroidManifest.xml
<application
  android:name=".DemoApplication"
  android:allowBackup="false"
  android:icon="@drawable/app_ic_launcher"
  android:label="@string/app_name"
  android:largeHeap="true"
  android:theme="@style/AppTheme"
  tools:replace="android:allowBackup">
```

5. 配置 Kotlin 支持（可选）
   - 如果您的项目已是 Kotlin 工程：可跳过此步。
   - 如果您的项目尚未配置 Kotlin：请添加对应版本的 Kotlin Gradle 插件：

> ⚠ **注意：**
> 若需添加 Kotlin Gradle 插件，$kotlin_version 请填入具体版本号，并确保其与您项目的 Android Gradle 插件版本兼容（如 $kotlin_version 设置为 1.9.0，对应的 Android Gradle 插件版本是 8.6.0）。可以参见 Kotlin-Gradle 插件兼容性 。

```
// project root build.gradle
buildscript {
    dependencies {
        classpath "org.jetbrains.kotlin:kotlin-gradle-
plugin:$kotlin_version"
    }
}
```

## 完成登录

在您的项目中添加登录代码，这是使用 `AIConversationKit` 各项功能的关键前提：

**Kotlin**

```
val userId = "denny" // 请替换为您的 UserID
val sdkAppId = 1400000001 // 请替换为开通服务控制台的 SDKAppID
val sdkSecretKey = "xxxx" // 请替换为开通服务控制台的 SDKSecretKey
val userSig = GenerateTestUserSig.genTestUserSig(sdkAppId, userId,
sdkSecretKey)
TUILogin.login(this, sdkAppId, userId, userSig, object : TUICallback() {
      override fun onSuccess() {}

      override fun onError(errorCode: Int, errorMessage: String) {}
})
```

**Java**

```
String userId = "denny"; // 请替换为您的 UserID
int sdkAppId = 1400000001; // 请替换为开通服务控制台的 SDKAppID
String sdkSecretKey = "xxxx"; // 请替换为开通服务控制台的 SDKSecretKey
String userSig = GenerateTestUserSig.genTestUserSig(sdkAppId, userId,
sdkSecretKey);
TUILogin.login(this, sdkAppId, userId, userSig, new TUICallback() {
    @Override public void onSuccess() {}

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

**登录接口参数说明**

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| SDKAppID | Int | 从 TRTC 控制台 > 应用管理 获取。 |
| UserID | String | 当前用户的用户ID，仅包含英文字母、数字、连字符和下划线。 |
| userSig | String | 用于腾讯云鉴权的票据。请注意：<br>• 开发环境：您可以使用本地 `GenerateTestUserSig.genTestSig` 函数生成 UserSig 或者通过 UserSig 辅助工具 生成临时的 UserSig。 |

| | | |
|---|---|---|
| | | • 生产环境：为了防止密钥泄露，请务必采用服务端生成 UserSig 的方式。详细信息请参见 服务端生成 UserSig 。<br>更多信息请参见 如何计算及使用 UserSig 。 |

## 集成 AI 对话界面

只需要将 `AIConversationView` 添加到您的布局文件中即可。

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

    <com.trtc.uikit.aiconversationkit.AIConversationView
        android:id="@+id/app_ai_conversation_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>
</FrameLayout>
```

## 开始您的第一次 AI 对话

登录成功后，使用以下代码发起 AI 对话：

Kotlin

```
val config = AIConversationConfig()

val sdkAppId = 1400000001  // 1、替换 sdkAppId
val sdkSecretKey = "xxx"   // 2、替换 sdkSecretKey
val aiRobotId = "robot_" + TUILogin.getUserId()
val aiRobotSig = GenerateTestUserSig.genTestUserSig(sdkAppId, aiRobotId,
sdkSecretKey)
config.agentConfig.aiRobotId = aiRobotId
config.agentConfig.aiRobotSig = aiRobotSig

config.secretId = "xxx"  // 3、替换 secretId
config.secretKey = "xxx"  // 4、替换 secretKey

// 5、替换 llmConfig
```

```
config.llmConfig = "{\"LLMType\":\"openai\",\"Model\":\"hunyuan-turbo-
latest\",\"SystemPrompt\":\"您是一个私人助手
\",\"APIUrl\":\"https:xxx\",\"APIKey\":\"xxx\",\"History\":5,\"Streaming
\":true}"
// 6、替换 ttsConfig
config.ttsConfig = "
{\"TTSType\":\"tencent\",\"AppId\":\"xxx\",\"SecretId\":\"xxx\",\"Secret
Key\":\"xxx\",\"VoiceType\":\"502001\",\"Speed\":1.25,\"Volume\":5,\"Pri
maryLanguage\":1,\"FastVoiceType\":\"\"}"

val aiConversationView = findViewById<AIConversationView>
(R.id.app_ai_conversation_view)
aiConversationView.startAIConversation(config, null)
```

**Java**

```
AIConversationConfig config = new AIConversationConfig();

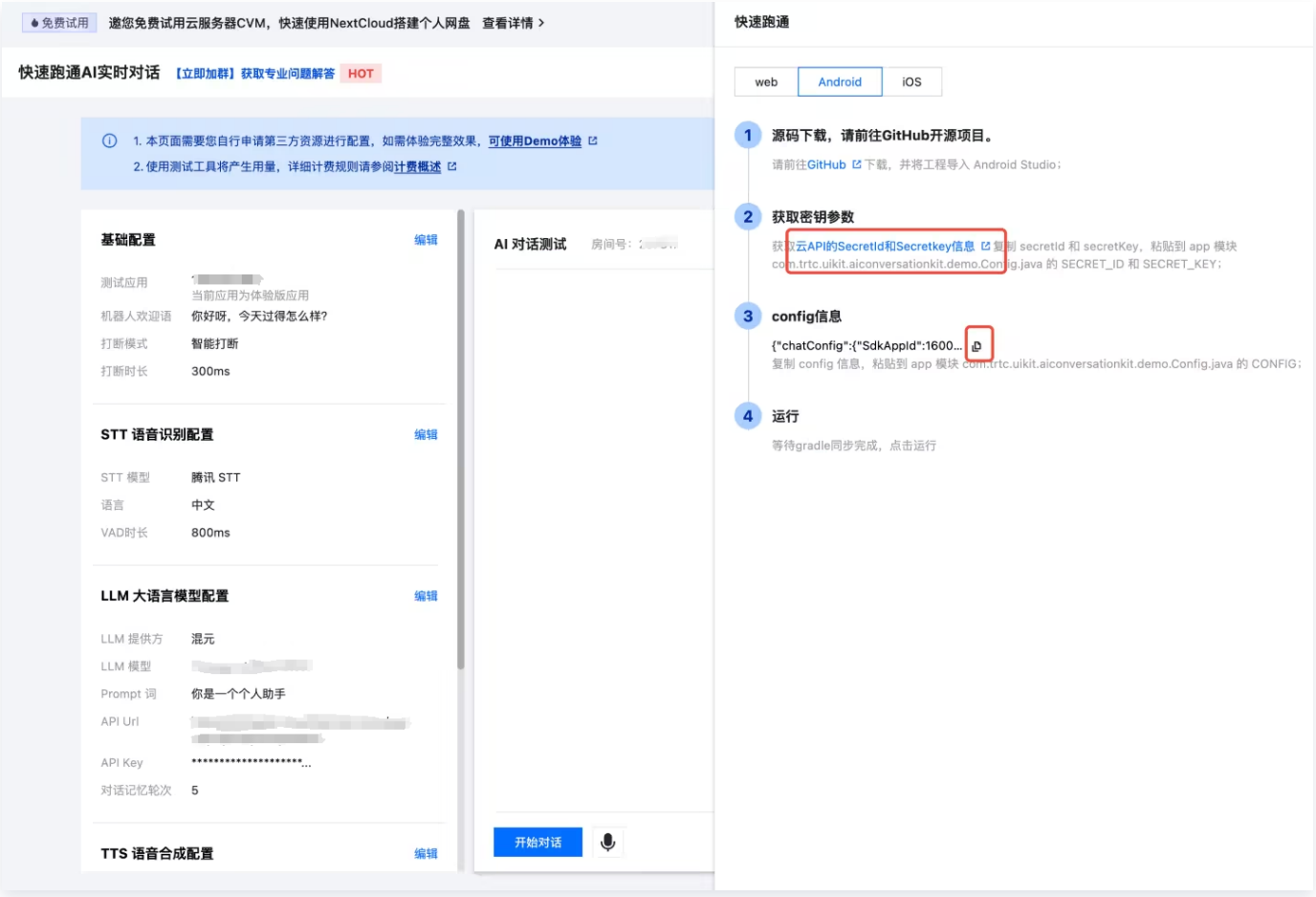
int sdkAppId = 1400000001;       // 1、替换 sdkAppId
String sdkSecretKey = "xxx";    // 2、替换 sdkSecretKey
String aiRobotId = "robot_" + TUILogin.getUserId();
String aiRobotSig = GenerateTestUserSig.genTestUserSig(sdkAppId,
aiRobotId, sdkSecretKey);
config.getAgentConfig().setAiRobotId(aiRobotId);
config.getAgentConfig().setAiRobotSig(aiRobotSig);

config.setSecretId("xxx");  // 3、替换 secretId
config.setSecretKey("xxx");  // 4、替换 secretKey

// 5、替换 llmConfig
config.setLlmConfig("{\"LLMType\":\"openai\",\"Model\":\"hunyuan-turbo-
latest\",\"SystemPrompt\":\"您是一个私人助手
\",\"APIUrl\":\"https:xxx\",\"APIKey\":\"xxx\",\"History\":5,\"Streaming
\":true}");
// 6、替换 ttsConfig
config.setTtsConfig("
{\"TTSType\":\"tencent\",\"AppId\":\"xxx\",\"SecretId\":\"xxx\",\"Secret
Key\":\"xxx\",\"VoiceType\":\"502001\",\"Speed\":1.25,\"Volume\":5,\"Pri
maryLanguage\":1,\"FastVoiceType\":\"\"}");
```

```
AIConversationView aiConversationView =
findViewById(R.id.app_ai_conversation_view);
aiConversationView.startAIConversation(config, null);
```

1. SDKAppId 和 SDKSecretKey 使用 完成登录 获取的数据即可。
2. 请到 准备工作 配置 AI 对话后台参数，包括基础配置、STT、LLM、TTS；然后点击右下角的**快速跑通**，切换到 Android，获取 SecretId、SecretKey 和 Config 参数。




3. 将云 API 的 SecretId 和 SecretKey 复制到 `config.secretId` 和 `config.secretKey` 中，SecretId 和 SecretKey 请妥善保存，避免泄露。
4. 将 Config 信息复制到 JSON 解析工具中，例如 JSONUtil，将 LLMConfig 对应的字符串值复制到 `config.llmConfig`，将 TTSConfig 对应的字符串值复制到 `config.ttsConfig` 中。

> **说明：**
>
> - **开发环境**：如果您正在本地开发调试阶段，可以采用上述方式快速集成 AI 对话。该方法中账号信息很容易被反编译逆向破解，一旦您的密钥泄露，攻击者就可以盗用您的腾讯云流量。
> - **生产环境**：如果您的项目要发布上线，请将上述账号信息保存到服务端中，避免流量被盗用；相关对话配置，也可以保存到服务端中，方便动态调整 AI 对话效果。

# 调节对话效果

为了在 AI 对话中获得最佳体验，清晰的声音采集至关重要。我们提供多种参数设置，帮助您在不同环境下都能获得清晰的语音交互效果。

## vadLevel 远场人声抑制

在多人交谈的环境中，其他人声可能会干扰 AI 对您声音的识别。通过调整人声抑制级别，我们可以有效过滤环境中的其他人声，让 AI 更准确地专注于您的语音；您可以通过设置 `config` 中的 `sttConfig.vadLevel` 参数来达到期望的效果。

| 级别 | 效果描述 | 使用场景 |
| --- | --- | --- |
| 0 | 关闭远场人声抑制功能 | 在嘈杂环境下 AI 容易被噪声打断，可能导致无法正常对话。 |
| 1 | 开启轻度抑制模式 | 适合相对安静的环境使用，例如家庭、图书馆。 |
| 2 | 推荐默认设置 | 适用于大多数日常场景（默认值）。 |
| 3 | 开启强力抑制模式 | 适合嘈杂环境使用，例如菜市场、车站。 |

**代码配置示例**

Kotlin

```
val config = AIConversationConfig()
config.sttConfig.vadLevel = 3

aiConversationView.startAIConversation(config, null)
```

Java

```
AIConversationConfig config = new AIConversationConfig();
config.getSttConfig().setVadLevel(2);

aiConversationView.startAIConversation(config, null);
```

## interruptMode 打断模式

根据您的使用习惯，可以设置不同的打断方式，在 AI 说话时进行打断，使对话交互更加自然流畅；您可以通过设置 `config` 中的 `agentConfig.interruptMode` 参数来达到期望的效果。

| 模式 | 功能描述 | 适用场景 |
| --- | --- | --- |
| 0 | 智能模式 | AI 正在说话时，可通过点击屏幕或说话打断。（默认值） |
| 1 | 手动打断 | AI 正在说话时，只能通过点击屏幕打断。 |

**代码配置示例**

**Kotlin**

```
val config = AIConversationConfig()
config.agentConfig.interruptMode = 1

aiConversationView.startAIConversation(config, null)
```

**Java**

```
AIConversationConfig config = new AIConversationConfig();
config.getAgentConfig().setInterruptMode(1);

aiConversationView.startAIConversation(config, null);
```

## language 语言识别

根据说话者使用的语言进行设置，AI 才能准确地将语音转换成对应的文字内容；您可以通过设置 `config` 中的 `sttConfig.language` 参数来达到期望的效果。

| 语言代码 | 语言名称 | 说明 |
| --- | --- | --- |
| "zh" | 中文 | 支持中文识别（默认值） |
| "en" | 英语 | 支持英语识别 |

更多支持的语言，请参考 STTConfig 官方文档。

**代码配置示例**

**Kotlin**

```
val config = AIConversationConfig()
config.sttConfig.language = "en"
```

```
aiConversationView.startAIConversation(config, null)
```

Java

```
AIConversationConfig config = new AIConversationConfig();
config.getSttConfig().setLanguage("en");

aiConversationView.startAIConversation(config, null);
```

# 定制您的 UI

## 更新背景图

将新的背景图保存到 `drawable-xxxhdpi` 文件夹中，通过 `setBackgroundImage` 接口设置您自定义的背景图。

Kotlin

```
aiConversationView.setBackgroundResource(R.drawable.app_bg_ai_conversati
on_full_screen)
```

Java

```
aiConversationView.setBackgroundImage(R.drawable.app_bg_ai_conversation_
full_screen);
```

**最终效果：**



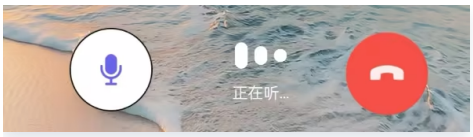

## 图标定制

AIConversationKit 用到的所有图标都存放于 `res/drawable-xxhdpi` 目录下，部分示例如下，您可以根据您的诉求来替换目录下的图标。

| 图标路径 | 图标样式 | 详细描述 |
| --- | --- | --- |
| conversation_ic_ai_state_interrupting |  | 打断 AI 说话的按钮。 |
| conversation_ic_mic_state_on |  | 本地 Mic 打开时的图标。 |
| conversation_ic_mic_state_off |  | 本地 Mic 被静音时的图标。 |
| conversation_ic_hang_up |  | 结束 AI 对话的图标。 |

例如，您需要替换 Mic 的图标，将新的图标直接覆盖原来的图标即可。

**最终效果：**




## 隐藏 UI

如果您不需要 麦克风、字幕、AI 交互 按钮，可以通过 `disableFeatures` 将它移除。例如您需要移除麦克风和 AI 交互，可以通过以下代码实现。

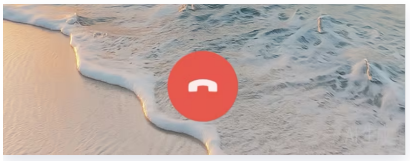
**Kotlin**

```kotlin
val features = listOf(Feature.MIC, Feature.AI_INTERACTION)
aiConversationView.disableFeatures(features)
```

**Java**

```java
List<Feature> features = List.of(Feature.MIC, Feature.AI_INTERACTION);
aiConversationView.disableFeatures(features);
```

**最终效果：**



# 常见问题

## 每次进房都需要调用登录吗?

不需要。通常您只需要完成一次 `TUILogin.login` 调用即可，建议将 `TUILogin.login` 和 `TUILogin.logout` 与自己的登录业务关联。

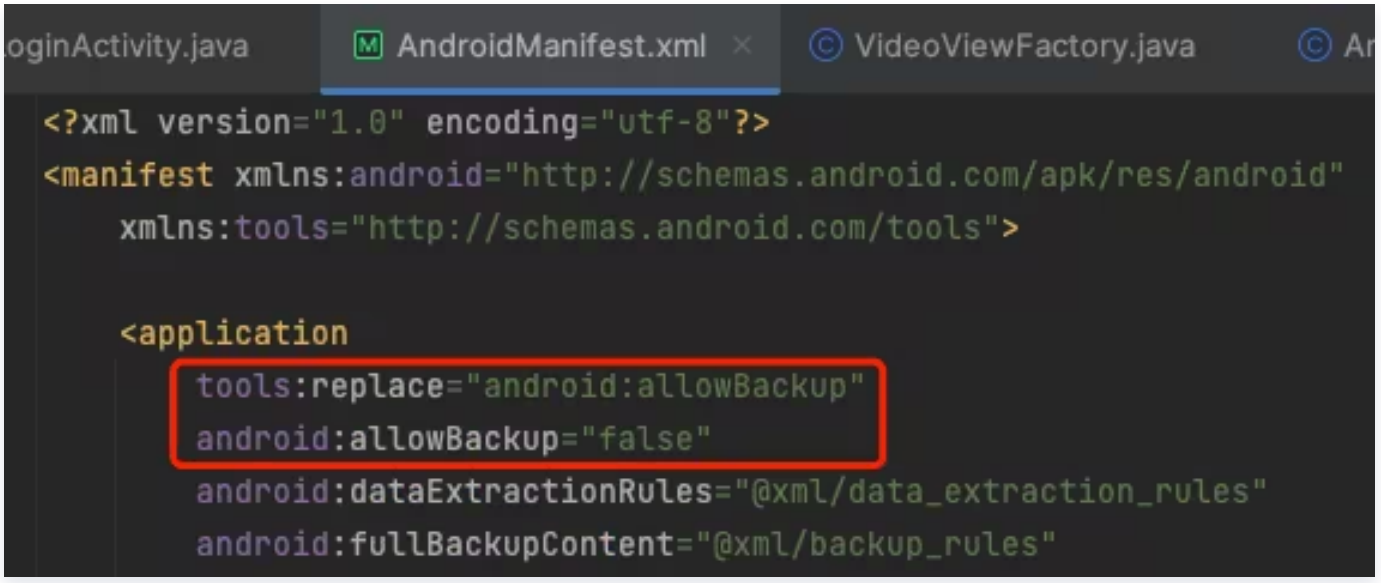
## 集成代码后产生如下图所示编译报错 allowBackup 异常，如何处理?

```
Manifest merger failed : Attribute application@allowBackup value=(false) from AndroidManifest.xml:7:9-36
    is also present at [com.github.yyued:SVGAPlayer-Android:2.6.1] AndroidManifest.xml:12:9-35 value=(true).
    Suggestion: add 'tools:replace="android:allowBackup"' to <application> element at AndroidManifest.xml:5:5-53:19 to override.
```

- **问题原因：**多个模块的 `AndroidManifest.xml` 中都配置了 `allowBackup` 属性，造成冲突。
- **解决方法：**

1. 在工程的 `AndroidManifest.xml` 文件中删除 `allowBackup` 属性或将该属性改为 `false`，表示关闭备份和恢复功能；

2. 在 `AndroidManifest.xml` 文件的 application 节点中添加 `tools:replace="android:allowBackup"`，表示覆盖其他模块的设置，使用您自己的设置；




## 集成代码后还需要添加麦克风等权限的声明吗？

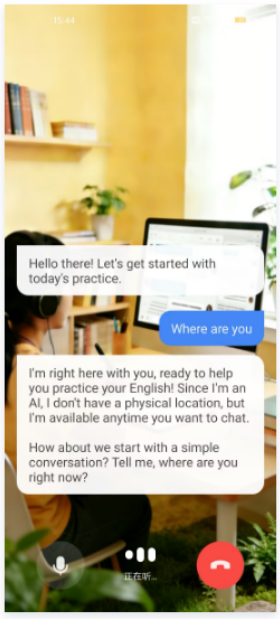
不需要，AIConversationKit 中已经内置了麦克风等权限的声明，在接入过程中您无需再关心这些权限的声明。

# 全功能集成（iOS）

最近更新时间：2026-03-17 11:34:47

## 功能预览

通过集成 `AIConversationKit` 组件，您可为产品快速集成前沿的 AI 对话能力，实现流畅自然的智能交互体验，显著提升用户活跃度与满意度。

| 在线教育 | 智能客服 | 情感陪伴 |
| --- | --- | --- |
|  |  |  |



## 准备工作

### 开通服务

使用 `AIConversationKit` 前，请先参考 开通服务，领取体验版或开通付费版。

### 环境要求

- **Xcode**：需使用 Xcode 15 或更高版本。
- **iOS 系统**：支持 iOS 13.0 或更高版本的设备。
- **CocoaPods 环境**：已安装 CocoaPods 环境。如果您尚未安装，请参见 CocoaPods 官网安装，或按以下步骤操作：
  - **使用 gem 安装 CocoaPods**：在终端中执行 `sudo gem install cocoapods` 命令进行安装。

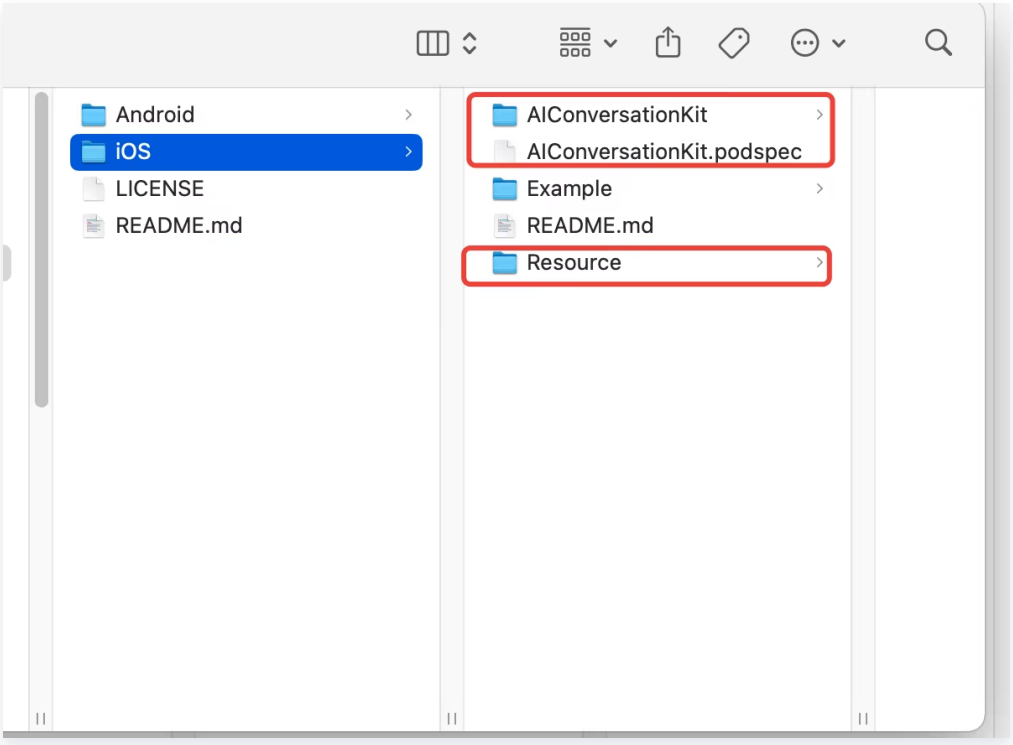
> **提示：**

> `sudo gem install cocoapods` 安装过程中可能需要输入电脑密码，按提示输入管理员密码即可。

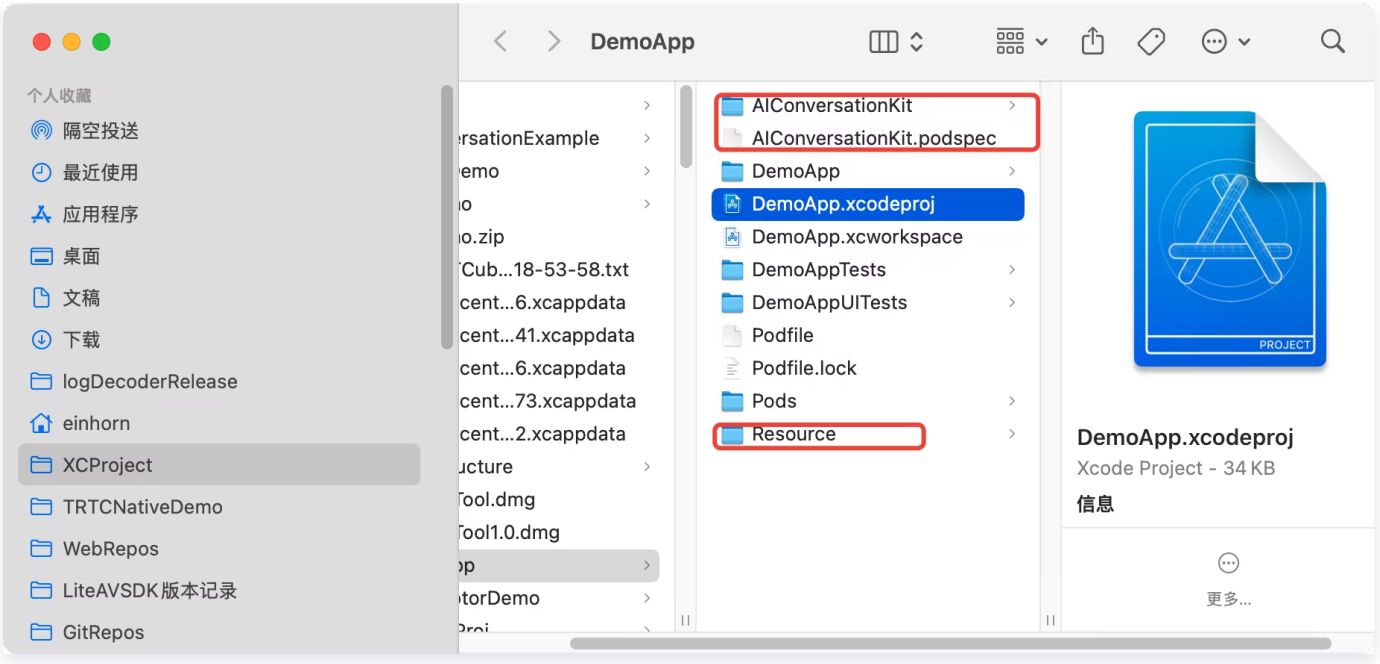
# 代码集成

## 步骤 1：获取 AIConversationKit 组件

1. 从 GitHub 下载 zip 文件，解压后，您会看到 AIConversationKit 目录、AIConversationKit.podspec 文件和 Resource 目录。

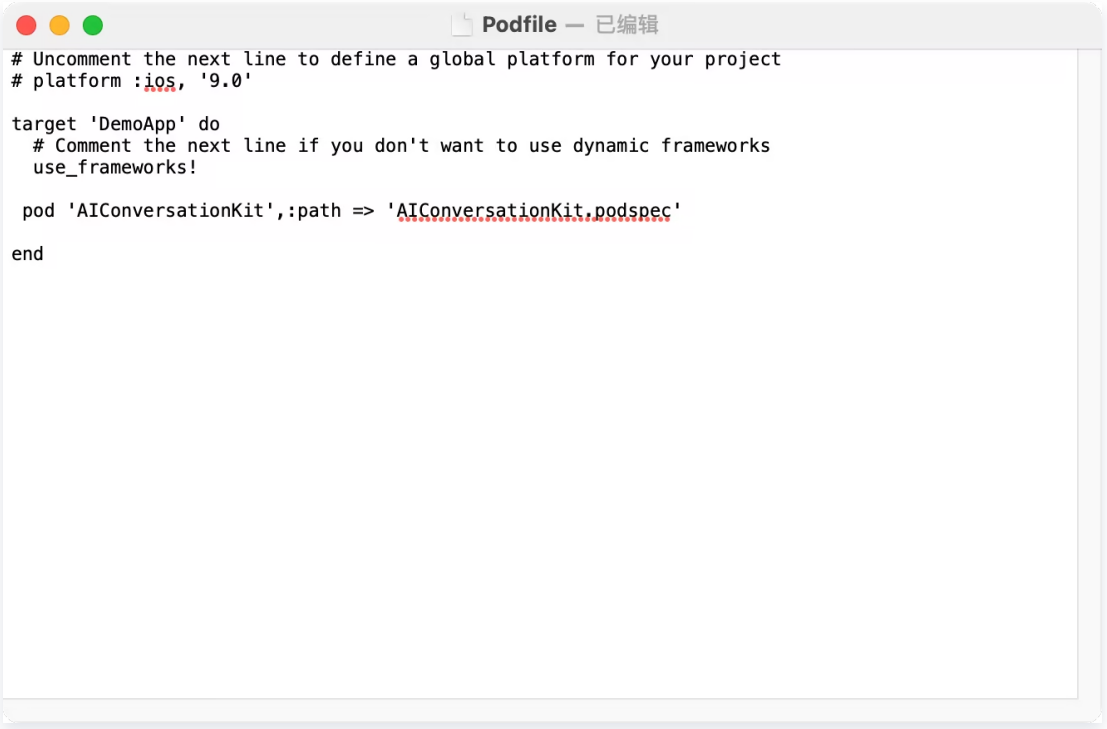



2. 复制上一步中解压得到的目录里的 AIConversationKit、podspec 和 Resource 到您的工程中。



3. 然后在 `.xcodeproj` 打开控制台，执行 `pod init` 。
4. 在生成的 Podfile 里这样写：

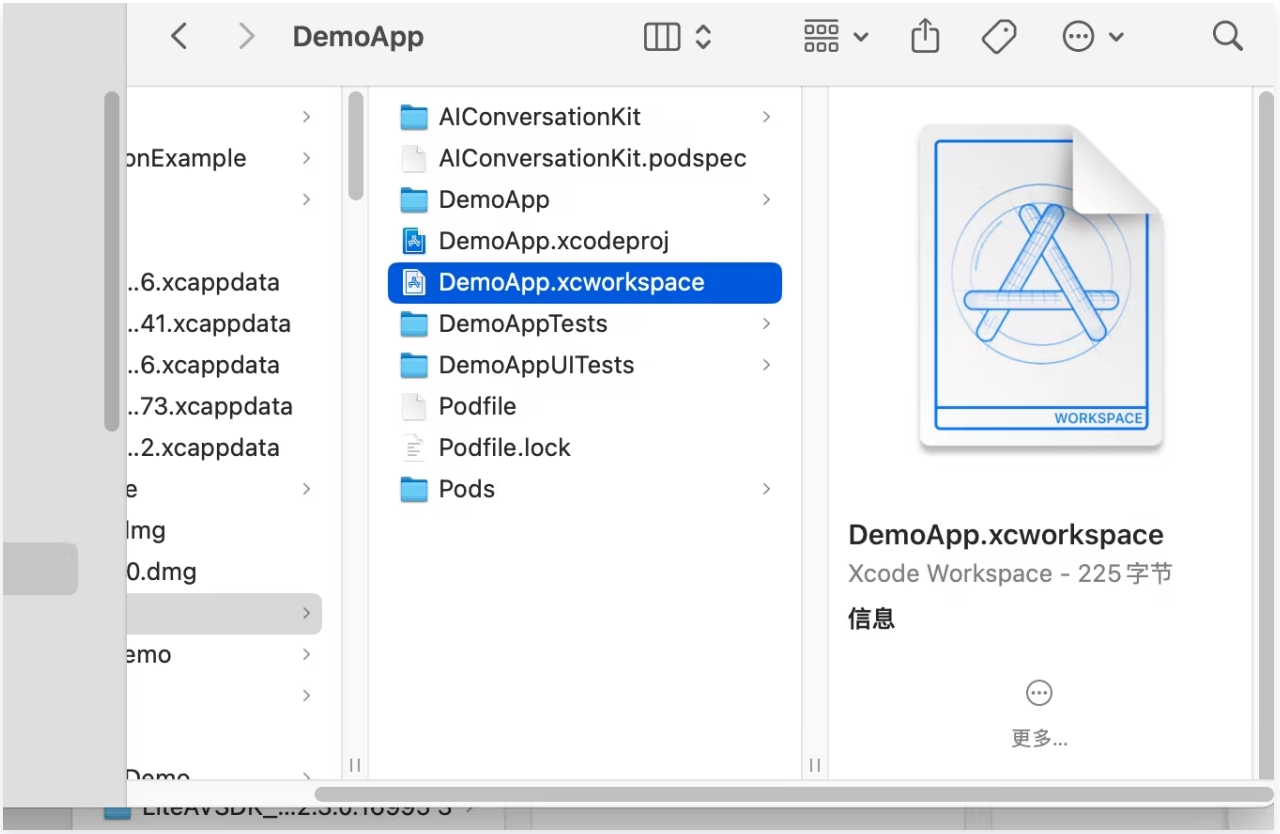
```
pod 'AIConversationKit',:path => 'AIConversationKit.podspec'
```



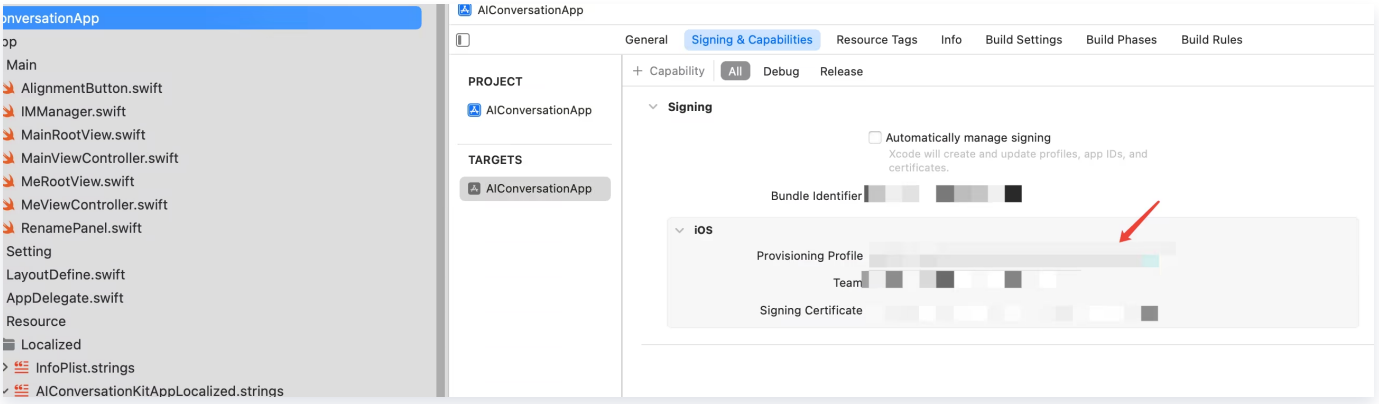

5. 执行 `pod install` 。




6. 打开 `.xcworkspace` ：



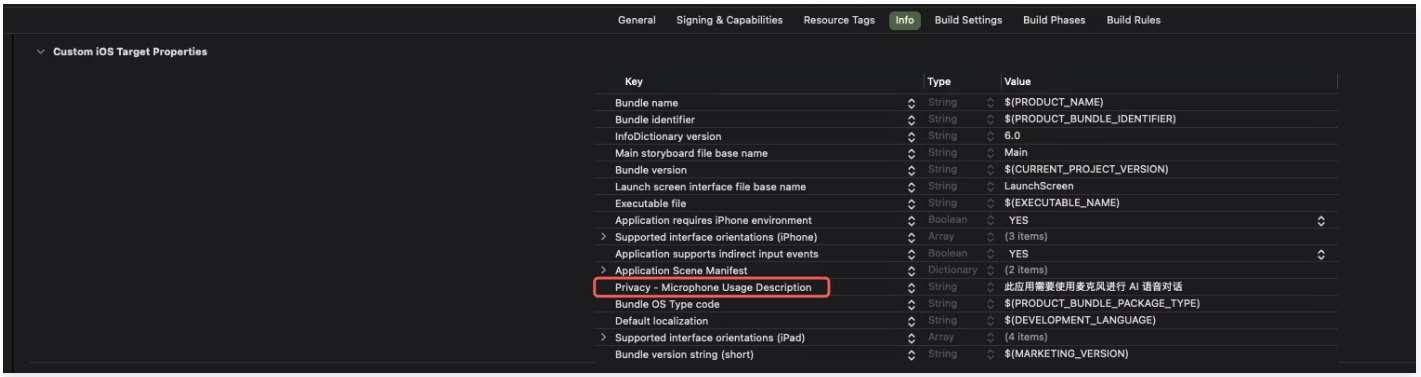

7. 配置证书信息：




## 步骤 2：工程配置

使用 AI 对话，需要授权麦克风使用权限。在 App 的 Info.plist 中添加以下麦克风声明，对应麦克风系统弹出授权对话框时的提示信息。

```
<key>NSMicrophoneUsageDescription</key>
<string>此应用需要使用麦克风进行 AI 语音对话</string>
```




## 完成登录

在您的项目中添加登录代码，这是使用 `AIConversationKit` 各项功能的关键前提：

swift

```
import TUICore
import AIConversationKit

// 示例代码在 didFinishLaunchingWithOptions 完成登录，推荐您在自己登录业务完成后再调用登录服务
func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
        let userId = "denny"       // 请替换为您的 UserID
        let sdkAppId = 1600000001  // 请替换为开通服务控制台的 SDKAppID
        let sdkSecretKey = "xxx"       // 请替换为开通服务控制台的 SDKSecretKey
        let userSig = GenerateTestUserSig.genTestUserSig(sdkAppId: sdkAppId, userId: userId, secrectkey: sdkSecretKey)
        TUILogin.login(Int32(sdkAppId), userID: userId, userSig:userSig) {
            print("login success")
        } fail: { code, message in
            print("login failed, code: \(code), error: \(message ?? "nil")")
        }
    return true
}
```

**登录接口参数说明**

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| SDKAppID | Int | 从 TRTC 控制台 > 应用管理 获取。 |
| UserID | String | 当前用户的唯一 ID，仅包含英文字母、数字、连字符和下划线。 |
| userSig | String | 用于腾讯云鉴权的票据。请注意：<br>• 开发环境：您可以采用本地 `GenerateTestUserSig.genTestSig` 函数生成 UserSig 或者通过 UserSig 辅助工具 生成临时的 UserSig。<br>• 生产环境：为了防止密钥泄露，请务必采用服务端生成 UserSig 的方式。详细信息请参见 服务端生成 UserSig 。<br>更多信息请参见 如何计算及使用 UserSig 。 |

## 集成 AI 对话界面

只需要将 `AIConversationView` 添加到您的 ViewController 中即可。

```
import UIKit
import AIConversationKit

class YourAIViewController: UIViewController {

    private let conversationView = AIConversationView()

    override func viewDidLoad() {
        super.viewDidLoad()

        view.addSubview(conversationView)
        conversationView.frame = view.bounds
        conversationView.autoresizingMask = [.flexibleWidth,
.flexibleHeight]
    }
}
```

## 开始您的第一次 AI 对话

登录成功后，使用以下代码发起 AI 对话：

```
import UIKit
```

```
import AIConversationKit
import TUICore

class YourAIViewController: UIViewController {

    // ... 其他代码 ...

    private func startAIConversation() {
        var config = AIConversationConfig()

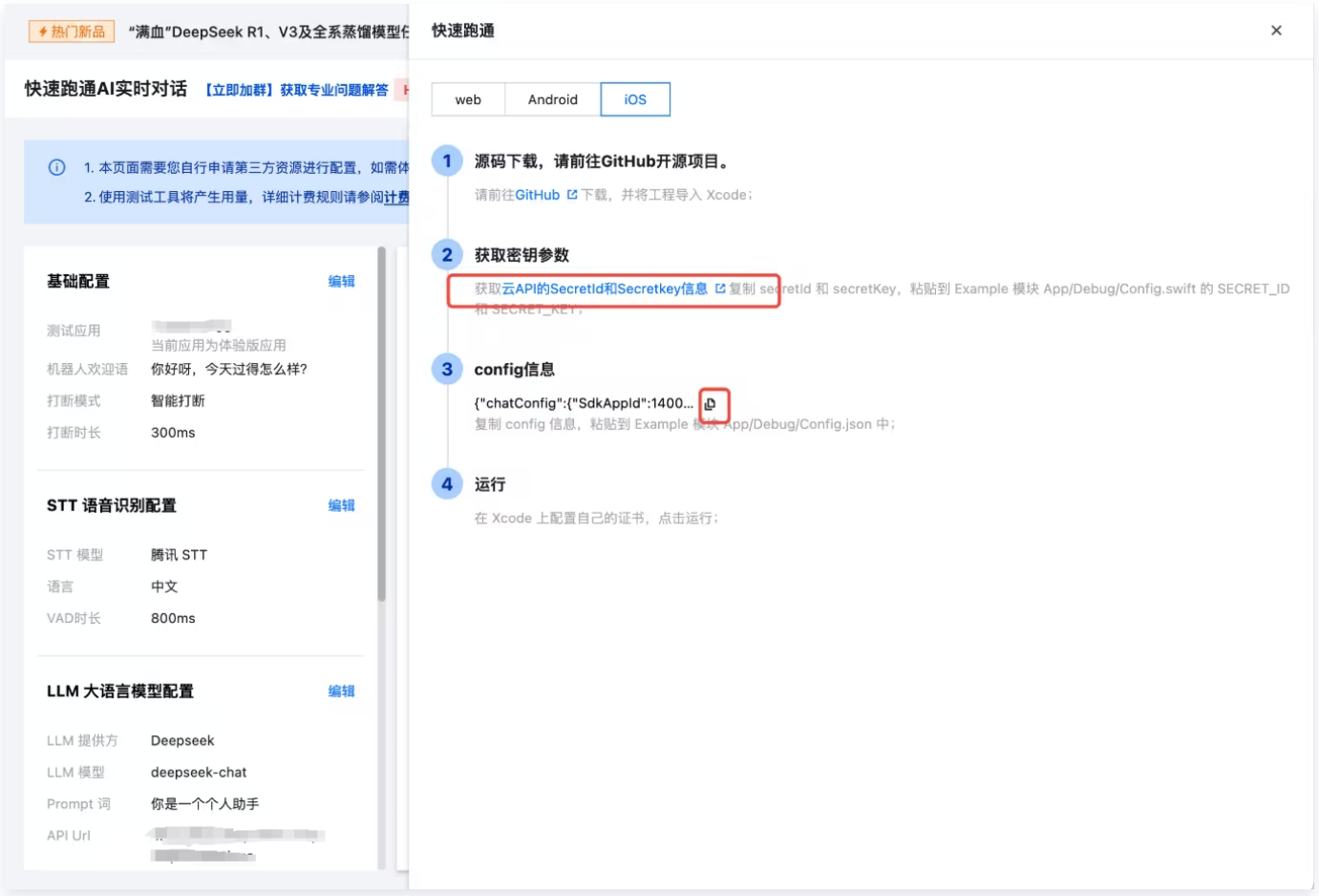
        let sdkAppId = 1600000001  // 1、替换 sdkAppId
        let sdkSecretKey = "xxx"   // 2、替换 sdkSecretKey
        let aiRobotId = "robot_\(TUILogin.getUserID() ?? "")"
        let aiRobotSig = GenerateTestUserSig.genTestUserSig(sdkAppId:
sdkAppId, userId: aiRobotId, secrectkey: sdkSecretKey)
        config.agentConfig?.aiRobotId = aiRobotId
        config.agentConfig?.aiRobotSig = aiRobotSig

        config.secretId = "xxx"           // 3、替换 secretId
        config.secretKey = "xxx"          // 4、替换 secretKey

        // 5、替换 llmConfig
        config.llmConfig = "{\"LLMType\":\"openai\",\"Model\":\"hunyuan-
turbo-latest\",\"SystemPrompt\":\"您是一个个人助手
\",\"APIUrl\":\"https://hunyuan.cloud.tencent.com/openai/v1/chat/complet
ions\",\"APIKey\":\"xxxx\",\"History\":5,\"Streaming\":true}"
        // 6、替换 ttsConfig
        config.ttsConfig = "
{\"TTSType\":\"tencent\",\"AppId\":\"xxx\",\"SecretId\":\"xxx\",\"Secret
Key\":\"xxx\",\"VoiceType\":\"502001\",\"Speed\":1.25,\"Volume\":5,\"Pri
maryLanguage\":1,\"FastVoiceType\":\"\"}"

        conversationView.startAIConversation(config: config) { code,
message in
            print("startAIConversation code: \(code), message: \
(message)")
        }
    }
}
```

1. sdkAppId 和 sdkSecretKey 使用 完成登录 获取的数据即可。
2. 请到 准备工作 配置 AI 对话后台参数，包括基础配置、STT、LLM、TTS；然后点击右下角的**快速跑通**，切换到 iOS，获取 SecretId、SecretKey 和 Config 参数。




3. 将云 API 的 SecretId 和 SecretKey 复制到 `config.secretId` 和 `config.secretKey` 中，SecretId 和 SecretKey 请妥善保存，避免泄露。
4. 将 Config 信息复制到 Json 解析工具中，如 JsonUtil，将 LLMConfig 对应的字符串值复制到 `config.llmConfig`，将 TTSConfig 对应的字符串值复制到 `config.ttsConfig` 中。

> **说明：**
> - **开发环境**：如果您正在本地开发调试阶段，可以采用上述方式快速集成 AI 对话。该方法中账号信息很容易被反编译逆向破解，一旦您的密钥泄露，攻击者就可以盗用您的腾讯云流量。
> - **生产环境**：如果您的项目要发布上线，请将上述账号信息保存到服务端中，避免流量被盗用；相关对话配置，也可以保存到服务端中，方便动态调整 AI 对话效果。

## 调节对话效果

为了在 AI 对话中获得最佳体验，清晰的声音采集至关重要。我们提供多种参数设置，帮助您在不同环境下都能获得清晰的语音交互效果。

## vadLevel 远场人声抑制

在多人交谈的环境中，其他人声可能会干扰 AI 对您声音的识别。通过调整人声抑制级别，我们可以有效过滤环境中的其他人声干扰，让 AI 更准确地专注于您的语音；您可以通过设置 `config` 中的 `sttConfig.vadLevel` 参数来达到期望的效果。

| 级别 | 效果描述 | 使用场景 |
| --- | --- | --- |
| 0 | 关闭远场人声抑制功能 | 在嘈杂环境下AI容易被噪声打断，可能导致无法正常对话。 |
| 1 | 开启轻度抑制模式 | 适合相对安静的环境使用，例如家庭、图书馆。 |
| 2 | 推荐默认设置 | 适用于大多数日常场景（默认值）。 |
| 3 | 开启强力抑制模式 | 适合嘈杂环境使用，如菜市场、车站。 |

### 代码配置示例

```
var config = AIConversationConfig()
config.sttConfig?.vadLevel = 3

conversationView.startAIConversation(config: config, completion: nil)
```

## interruptMode 打断模式

根据您的使用习惯，可以设置不同的打断方式，在 AI 说话时进行打断，使对话交互更加自然流畅；您可以通过设置 `config` 中的 `agentConfig.interruptMode` 参数来达到期望的效果。

| 模式 | 功能描述 | 适用场景 |
| --- | --- | --- |
| 0 | 智能模式 | AI 正在说话时，可通过点击屏幕或用户说话打断；（默认值） |
| 1 | 手动打断 | AI 正在说话时，只能通过点击屏幕打断； |

### 代码配置示例

```
var config = AIConversationConfig()
config.agentConfig?.interruptMode = 1

conversationView.startAIConversation(config: config, completion: nil)
```

## language 语言识别

根据说话者使用的语言进行设置，AI 才能准确地将语音转换成对应的文字内容；您可以通过设置 `config` 中的 `sttConfig.language` 参数来达到期望的效果。

| 语言代码 | 语言名称 | 说明 |
| --- | --- | --- |
| "zh" | 中文 | 支持普通话识别（默认值） |
| "en" | 英语 | 支持英语识别 |

更多支持的语言，请参考 STTConfig 官方文档。

**代码配置示例**

```
var config = AIConversationConfig()
config.sttConfig?.language = "en"

conversationView.startAIConversation(config: config, completion: nil)
```

# 定制您的 UI

## 更新背景图

将新的背景图保存到 `Assets.xcassets` 文件夹中，通过 `setBackgroundImage` 接口设置您自定义的背景图。

```
conversationView.setBackgroundImage(UIImage(named:
"ai_conversation_full_screen"))
```

**最终效果：**



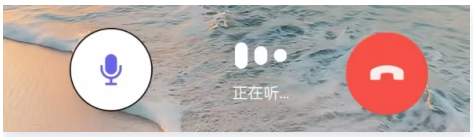

## 图标定制

AIConversationView 用到的所有图标都存放于 `Resource/AIConversationKit.xcassets` 目录下，部分示例如下，您可以根据您的诉求来替换目录下的图标。

| 图标路径 | 图标样式 | 详细描述 |
| --- | --- | --- |
| ai_pause |  | 打断 AI 说话的按钮。 |
| ai_audio_unmute |  | 本地 Mic 打开时的图标。 |
| ai_audio_mute |  | 本地 Mic 被静音时的图标。 |
| ai_hangup |  | 结束 AI 对话的图标。 |

例如，您需要替换 Mic 的图标，将新的图标直接覆盖原来的图标即可。

**最终效果：**

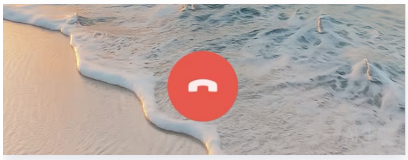



## 隐藏 UI

如果您不需要麦克风、字幕、AI 交互按钮，可以通过 `disableFeatures` 将它移除。例如您需要移除麦克风、AI 状态、打断 AI，只留下挂断按钮，可以通过以下代码实现。

```
conversationView.disableFeatures([.mic, .interruptSpeech,
.aiStatusAnimation])
```

**最终效果：**



# 常见问题

## pod install 执行后本地安装找不到 AIConversationKit 最新版本?

如果无法安装 TUILiveKit 最新版本，请按以下步骤操作：

1. 在 Podfile 所在目录下删除 `Podfile.lock` 和 `Pods`，您可以选择手动删除或终端执行以下命令

```
// cd 到 Podfile 所在目录下

rm -rf Pods/
rm Podfile.lock
```

2. 在 Podfile 所在目录下执行 `pod install --repo-update`

```
// cd 到 Podfile 所在目录下

pod install --repo-update
```

## 每次进房都需要调用登录吗?

不需要。通常您只需要完成一次 `TUILogin.login` 调用即可，我们建议您将 `TUILogin.login` 和 `TUILogin.logout` 与自己的登录业务关联。

## Podfile 文件有没有示例配置可以参考?

您可以参考 GitHub AIConversationKit Example 工程 `Podfile` 示例文件。

## 部署目标版本过低警告？

如果您遇到如下错误：

```
clang: error: SDK does not contain 'libarclite' at the path
'/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xcto
olchain/usr/lib/arc/libarclite_iphonesimulator.a'; try increasing the
minimum deployment target

部署目标版本过低警告：
TXLiteAVSDK_Professional: iOS 8.0 → 需要 12.0+
TXIMSDK_Plus_iOS_XCFramework: iOS 8.0 → 需要 12.0+
TUICore: iOS 9.0 → 需要 12.0+
```

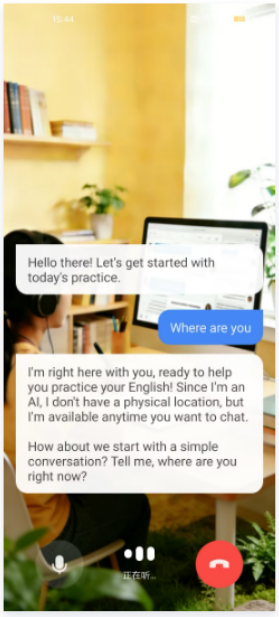
请参考 GitHub AIConversationKit Example 工程 `Podfile` 示例文件，添加以下代码：

```
post_install do |installer|
  installer.pods_project.targets.each do |target|
    target.build_configurations.each do |config|
      config.build_settings['CODE_SIGNING_ALLOWED'] = 'NO'
      config.build_settings['ENABLE_BITCODE'] = 'NO'
      config.build_settings['IPHONEOS_DEPLOYMENT_TARGET'] = 13.0
    end
  end
end
```

# 原子组件集成
# 原子组件集成（Android）

最近更新时间：2026-03-24 15:35:21

本篇文档旨在指导 Android 开发者如何使用核心组件 `AIConversationCoreView`，快速构建具备 AI 对话功能的 App。

| 在线教育 | 智能客服 | 情感陪伴 |
| --- | --- | --- |
|  |  |  |



## 核心功能

`AIConversationCoreView` 是一个专为 AI 对话设计的轻量级 View 组件，是您构建 AI 对话的核心，它封装了所有复杂的底层技术（例如开启 AI 对话、结束 AI 对话）。您可以将 `AIConversationCoreView` 作为 AI 对话画面的"画布"，专注于上层 UI 与交互的开发。

## 准备工作

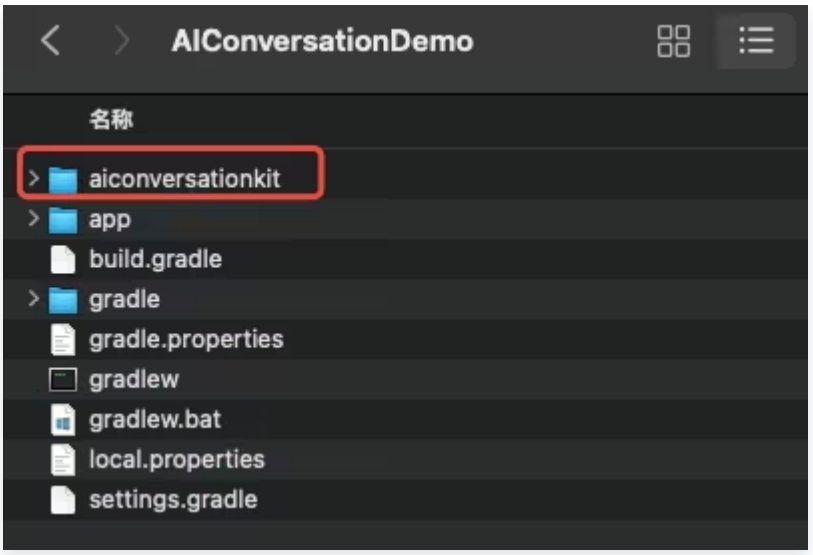
### 步骤 1：开通服务

使用 `AIConversationCoreView` 前，请先参考 开通服务，领取体验版或开通付费版。

### 步骤 2：代码集成

#### 获取 AIConversationKit 组件

从 GitHub 克隆/下载代码，将 `Android/aiconversationkit` 目录复制到您的工程中（与 app 目录同级）。




## 工程配置

1. 在工程根目录的 `settings.gradle`（或 `settings.gradle.kts`）文件中添加以下代码，它的作用是将 `aiconversationkit` 组件导入到您当前的项目中。

**settings.gradle**

```
include ':aiconversationkit'
```

**settings.gradle.kts**

```
include (":aiconversationkit")
```

2. 在 app 目录的 `build.gradle`（或 `build.gradle.kts`）文件中添加以下代码，它的作用是声明当前 `app` 对新加入的 `aiconversationkit` 组件的依赖。

**build.gradle**

```
api project(':aiconversationkit')
```

**build.gradle.kts**

```
api(project(":aiconversationkit"))
```

3. 在 proguard-rules.pro 文件中添加以下代码，因为我们在 SDK 内部使用了Java 的反射特性，需要将 SDK 的部分类加入不混淆名单。

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

4. 为了防止编译时，`AndroidManifest` 合并过程中产生属性冲突，您需要在 `app/src/main/AndroidManifest.xml` 文件的 `<application>` 节点中，添加 `tools:replace="android:allowBackup"` 和 `android:allowBackup="false"`，用以覆盖组件内的设置。

```
<!-- app/src/main/AndroidManifest.xml -->
<application
  android:name=".DemoApplication"
  android:allowBackup="false"
  android:icon="@drawable/app_ic_launcher"
  android:label="@string/app_name"
  android:largeHeap="true"
  android:theme="@style/AppTheme"
  tools:replace="android:allowBackup">
```

5. 配置 Kotlin 支持（可选）
   - 如果您的项目已是 Kotlin 工程：可跳过此步。
   - 如果您的项目尚未配置 Kotlin：请添加对应版本的 Kotlin Gradle 插件：

> ⚠ **注意：**
> 若需添加 Kotlin Gradle 插件，$kotlin_version 请填入具体版本号，并确保其与您项目的 Android Gradle 插件版本兼容（例如 $kotlin_version 设置为 1.9.0，对应的 Android Gradle 插件版本是 8.6.0）。可以参见 Kotlin-Gradle 插件兼容性 。

```
// project root build.gradle
buildscript {
    dependencies {
        classpath "org.jetbrains.kotlin:kotlin-gradle-
plugin:$kotlin_version"
    }
}
```

## 步骤 3：实现登录逻辑

在您的项目中添加登录代码，这是使用 `AIConversationCoreView` 各项功能的关键前提：

> ⚠ **重要：**
> 推荐在您 App 自身的用户账户登录成功后，再调用 TUILogin.login，以确保登录业务逻辑的清晰和一致。

```
val userId = "denny" // 请替换为您的 UserID
val sdkAppId = 1400000001 // 请替换为开通服务控制台的 SDKAppID
val sdkSecretKey = "xxxx" // 请替换为开通服务控制台的 SDKSecretKey
val userSig = GenerateTestUserSig.genTestUserSig(sdkAppId, userId,
sdkSecretKey)
TUILogin.login(this, sdkAppId, userId, userSig, object : TUICallback() {
      override fun onSuccess() {}

      override fun onError(errorCode: Int, errorMessage: String) {}
})
```

**登录接口参数说明**

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| SDKAppID | Int | 从 TRTC 控制台 > 应用管理 获取。 |
| UserID | String | 当前用户的用户ID，仅包含英文字母、数字、连字符和下划线。 |
| userSig | String | 用于腾讯云鉴权的票据。请注意：<br>开发环境：您可以采用本地 `GenerateTestUserSig.genTestUserSig` 函数生成 UserSig 或者通过 UserSig 辅助工具 生成临时的 UserSig。<br>生产环境：为了防止密钥泄露，请务必采用服务端生成 UserSig 的方式。详细信息请参见 服务端生成 UserSig 。<br>更多信息请参见 如何计算及使用 UserSig 。 |

# 搭建基础 AI 对话

## 步骤 1：布局 AIConversationCoreView

添加 `AIConversationCoreView` 到您的布局文件中。

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".YourAIActivity">

    <com.trtc.uikit.aiconversationkit.view.AIConversationCoreView
        android:id="@+id/ai_conversation_core_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

## 步骤 2：开启 AI 对话

通过调用 `startAIConversation` 接口开始 AI 对话，完整示例代码如下：

```
import android.os.Bundle
import android.util.Log
import androidx.appcompat.app.AppCompatActivity
import com.tencent.qcloud.tuicore.TUILogin
import com.trtc.uikit.aiconversationkit.debug.GenerateTestUserSig
import com.trtc.uikit.aiconversationkit.store.AIConversationConfig
import com.trtc.uikit.aiconversationkit.view.AIConversationCoreView
import io.trtc.tuikit.atomicxcore.api.CompletionHandler

// YourAIActivity 代表您的 AI 对话 Activity
class YourAIActivity : AppCompatActivity() {

    // 1. 将 AIConversationCoreView 作为您 Activity 的一个属性
    private lateinit var coreView: AIConversationCoreView

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.your_ai_activity_layout)

        coreView = findViewById<AIConversationCoreView>
(R.id.ai_conversation_core_view)
        startAIConversation()
    }
```

```
    private fun startAIConversation() {
        val config = AIConversationConfig()

        val sdkAppId = 1400000001  // 1、替换 sdkAppId
        val sdkSecretKey = "xxx"   // 2、替换 sdkSecretKey
        val aiRobotId = "robot_" + TUILogin.getUserId()
        val aiRobotSig = GenerateTestUserSig.genTestUserSig(sdkAppId,
aiRobotId, sdkSecretKey)
        config.agentConfig.aiRobotId = aiRobotId
        config.agentConfig.aiRobotSig = aiRobotSig

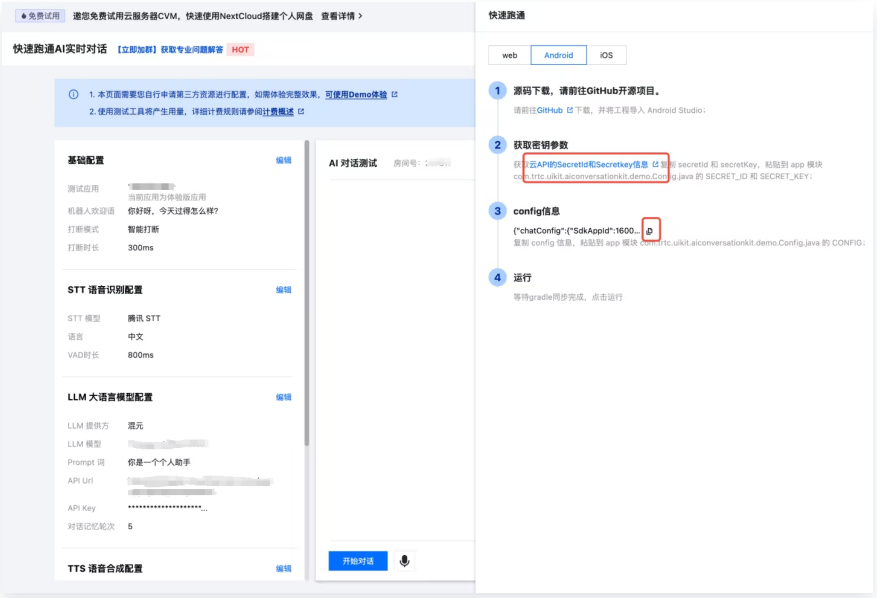
        config.secretId = "xxx"  // 3、替换 secretId
        config.secretKey = "xxx"  // 4、替换 secretKey


        // 5、替换 llmConfig
        config.llmConfig = "{\"LLMType\":\"openai\",\"Model\":\"hunyuan-
turbo-latest\",\"SystemPrompt\":\"您是一个私人助手
\",\"APIUrl\":\"https:xxx\",\"APIKey\":\"xxx\",\"History\":5,\"Streaming
\":true}"
        // 6、替换 ttsConfig
        config.ttsConfig = "
{\"TTSType\":\"tencent\",\"AppId\":\"xxx\",\"SecretId\":\"xxx\",\"Secret
Key\":\"xxx\",\"VoiceType\":\"502001\",\"Speed\":1.25,\"Volume\":5,\"Pri
maryLanguage\":1,\"FastVoiceType\":\"\"}"

        coreView.startAIConversation(config, object : CompletionHandler
{
            override fun onFailure(code: Int, desc: String) {
                Log.e("AIConversation", "startAIConversation failed
code: $code desc: $desc")
            }

            override fun onSuccess() {
                Log.d("AIConversation", "startAIConversation success")
            }
        })
    }
}
```

1. SDKAppId 和 SDKSecretKey 使用 完成登录 获取的数据即可。
2. 请到 准备工作 配置 AI 对话后台参数，包括基础配置、STT、LLM、TTS；然后点击右下角的**快速跑通**，切换到 Android，获取 SecretId、SecretKey 和 Config 参数。




3. 将云 API 的 SecretId 和 SecretKey 复制到 `config.secretId` 和 `config.secretKey` 中，SecretId 和 SecretKey 请妥善保存，避免泄露。
4. 将 Config 信息复制到 JSON 解析工具中，例如 JsonUtil，将 LLMConfig 对应的字符串值复制到 `config.llmConfig`，将 TTSConfig 对应的字符串值复制到 `config.ttsConfig` 中。

> **说明：**
> - **开发环境**：如果您正在本地开发调试阶段，可以采用上述方式快速集成 AI 对话。该方法中账号信息很容易被反编译逆向破解，一旦您的密钥泄露，攻击者就可以盗用您的腾讯云流量。
> - **生产环境**：如果您的项目要发布上线，请将上述账号信息保存到服务端中，避免流量被盗用；相关对话配置，也可以保存到服务端中，方便动态调整 AI 对话效果。

## 步骤 3：结束 AI 对话

通过调用 `stopAIConversation` 接口结束 AI 对话，完整示例代码如下：

```
import android.util.Log
import androidx.appcompat.app.AppCompatActivity
import com.trtc.uikit.aiconversationkit.view.AIConversationCoreView
import io.trtc.tuikit.atomicxcore.api.CompletionHandler

class YourAIActivity : AppCompatActivity() {

    // ... 其他代码 ...
```

```
    private fun stopAIConversation() {
        coreView.stopAIConversation(object : CompletionHandler {
            override fun onFailure(code: Int, desc: String) {
                Log.e("AIConversation", "stopAIConversation failed code:
$code desc: $desc")
            }

            override fun onSuccess() {
                Log.d("AIConversation", "stopAIConversation success")
            }
        })
    }

    // 确保在 Activity 销毁时也调用退出 AI 对话
    override fun onDestroy() {
        super.onDestroy()
        stopAIConversation()
        Log.d("AIConversation", "YourAIActivity onDestroy")
    }
}
```

## 步骤 4：显示 AI 对话字幕 (可选)

只需要将 `AIConversationSubtitleView` 添加到布局文件中，即可完成 AI 对话字幕的显示。

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    tools:context=".YourAIActivity">

    <com.trtc.uikit.aiconversationkit.view.AIConversationCoreView
        android:id="@+id/ai_conversation_core_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>
```

```
    <com.trtc.uikit.aiconversationkit.view.AIConversationSubtitleView
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_marginTop="150dp"
        android:layout_marginBottom="150dp"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toBottomOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

## 步骤 5：控制麦克风的开关（可选）

只需要将 `MicView` 添加到布局文件中，即可点击开关本地麦克风。

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    tools:context=".YourAIActivity">

    <com.trtc.uikit.aiconversationkit.view.AIConversationCoreView
        android:id="@+id/ai_conversation_core_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>

    <!-- 其它代码 -->

    <com.trtc.uikit.aiconversationkit.view.feature.MicView
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:layout_marginStart="32dp"
        android:layout_marginBottom="24dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintBottom_toBottomOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

版权所有：腾讯云计算（北京）有限责任公司

## 步骤 6：显示 AI 交互按钮（可选）

只需要将 `AIInteractionView` 添加到布局文件中，即可显示当前 AI 状态，并点击打断 AI 的说话。

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    tools:context=".YourAIActivity">

    <com.trtc.uikit.aiconversationkit.view.AIConversationCoreView
        android:id="@+id/ai_conversation_core_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>

    <!-- 其它代码 -->

    <com.trtc.uikit.aiconversationkit.view.feature.AIInteractionView
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:layout_marginBottom="24dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintBottom_toBottomOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

# 调节对话效果

为了在 AI 对话中获得最佳体验，清晰的声音采集至关重要。我们提供多种参数设置，帮助您在不同环境下都能获得清晰的语音交互效果。

## vadLevel 远场人声抑制

在多人交谈的环境中，其他人声可能会干扰 AI 对您声音的识别。通过调整人声抑制级别，我们可以有效过滤环境中的其他人声，让 AI 更准确地专注于您的语音；您可以通过设置 `config` 中的 `sttConfig.vadLevel` 参数来达到期望的效果。

| 级别 | 效果描述 | 使用场景 |
| --- | --- | --- |
| 0 | 关闭远场人声抑制功能 | 在嘈杂环境下 AI 容易被噪声打断，可能导致无法正常对话。 |
| 1 | 开启轻度抑制模式 | 适合相对安静的环境使用，例如家庭、图书馆。 |
| 2 | 推荐默认设置 | 适用于大多数日常场景（默认值）。 |
| 3 | 开启强力抑制模式 | 适合嘈杂环境使用，例如菜市场、车站。 |

**代码配置示例**

```
val config = AIConversationConfig()
config.sttConfig.vadLevel = 3

coreView.startAIConversation(config, null)
```

## interruptMode 打断模式

**根据您的使用习惯，可以设置不同的打断方式，在 AI 说话时进行打断，使对话交互更加自然流畅；您可以通过设置 `config` 中的 `agentConfig.interruptMode` 参数来达到期望的效果。**

| 模式 | 功能描述 | 适用场景 |
| --- | --- | --- |
| 0 | 智能模式 | AI 正在说话时，可通过点击屏幕或说话打断。（**默认值**） |
| 1 | 手动打断 | AI 正在说话时，只能通过点击屏幕打断。 |

**代码配置示例**

```
val config = AIConversationConfig()
config.agentConfig.interruptMode = 1

coreView.startAIConversation(config, null)
```

## language 语言识别

**根据说话者使用的语言进行设置，AI 才能准确地将语音转换成对应的文字内容；您可以通过设置 `config` 中的 `sttConfig.language` 参数来达到期望的效果。**

| 语言代码 | 语言名称 | 说明 |
| --- | --- | --- |
| "zh" | 中文 | 支持中文识别（默认值） |
| "en" | 英语 | 支持英语识别 |

更多支持的语言，请参考 STTConfig 官方文档。

**代码配置示例**

```
val config = AIConversationConfig()
config.sttConfig.language = "en"

coreView.startAIConversation(config, null)
```

# 常见问题

## 每次进房都需要调用登录吗?

不需要。通常您只需要完成一次 `TUILogin.login` 调用即可，建议将 `TUILogin.login` 和 `TUILogin.logout` 与自己的登录业务关联。

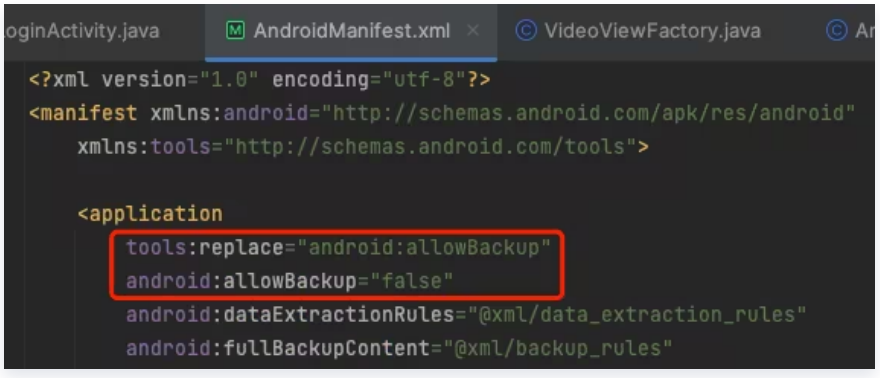
## 集成代码后产生如下图所示编译报错 allowBackup 异常，如何处理?

```
Manifest merger failed : Attribute application@allowBackup value=(false) from AndroidManifest.xml:7:9-36
    is also present at [com.github.yyued:SVGAPlayer-Android:2.6.1] AndroidManifest.xml:12:9-35 value=(true).
    Suggestion: add 'tools:replace="android:allowBackup"' to <application> element at AndroidManifest.xml:5:5-53:19 to override.
```

**问题原因：**多个模块的 `AndroidManifest.xml` 中都配置了 `allowBackup` 属性，造成冲突。

**解决方法：**

1. 在工程的 `AndroidManifest.xml` 文件中删除 `allowBackup` 属性或将该属性改为 `false`，表示关闭备份和恢复功能；
2. 在 `AndroidManifest.xml` 文件的 application 节点中添加 `tools:replace="android:allowBackup"`，表示覆盖其他模块的设置，使用您自己的设置；




## 集成代码后还需要添加麦克风等权限的声明吗?

不需要，AIConversationKit 中已经内置了麦克风等权限的声明，在接入过程中您无需再关心这些权限的声明。

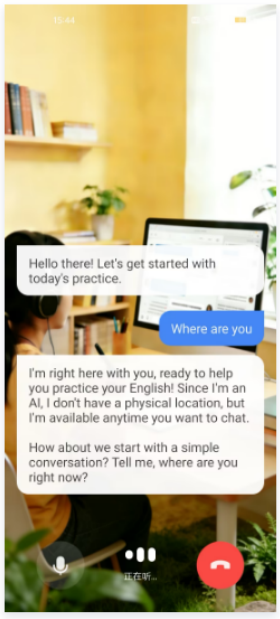
版权所有：腾讯云计算（北京）有限责任公司

# 原子组件集成（iOS）

最近更新时间：2026-03-24 15:35:21

## 功能预览

本篇文档旨在指导 iOS 开发者如何使用核心组件 `AIConversationCoreView`，快速构建具备 AI 对话功能的 app。

| 在线教育 | 智能客服 | 情感陪伴 |
| --- | --- | --- |
|  |  |  |



## 核心功能

`AIConversationCoreView` 是一个专为 AI 对话设计的轻量级 View 组件，是您构建 AI 对话的核心，它封装了所有复杂的底层技术（例如开启 AI 对话、结束 AI 对话）。您可以将 `AIConversationCoreView` 作为 AI 对话画面的"画布"，专注于上层 UI 与交互的开发。

## 准备工作

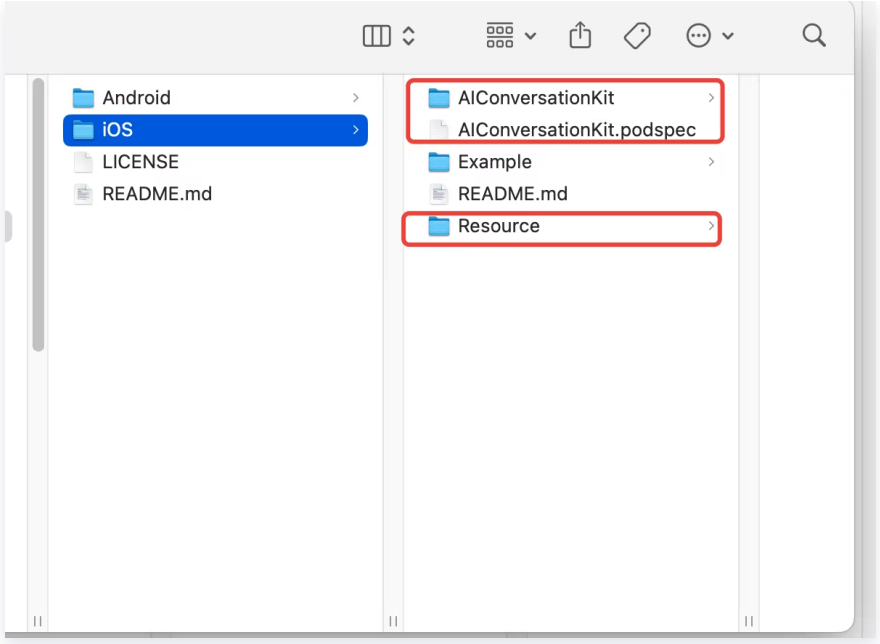
### 步骤 1：开通服务

使用 `AIConversationCoreView` 前，请先参考 开通服务，领取体验版或开通付费版。

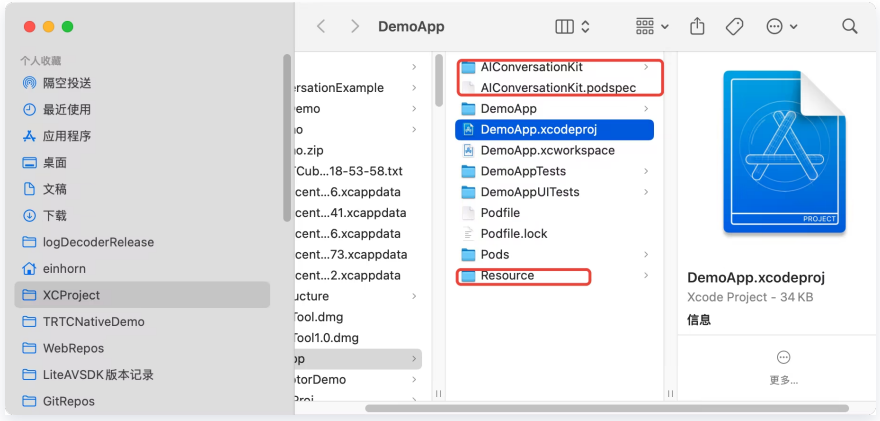
### 步骤 2：代码集成

#### 集成 AIConversationKit 组件

1. 从 GitHub 下载 zip 文件，解压后，您会看到 AIConversationKit 目录、AIConversationKit.podspec 文件和 Resource 目录。



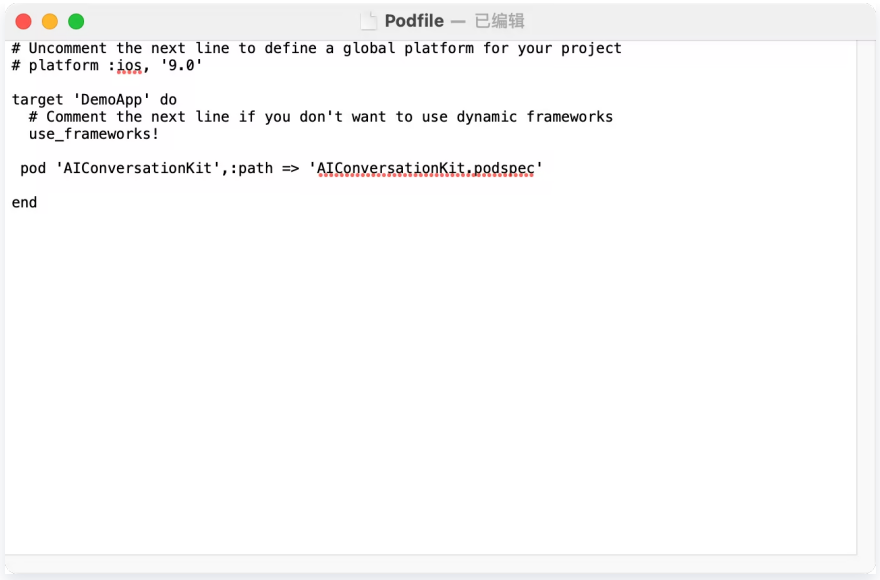

2. 复制上一步中解压得到的目录里的 AIConversationKit、podspec 和 Resource 到您的工程中。




3. 然后在 `.xcodeproj` 中打开控制台，执行 `pod init` 。

4. 在生成的 Podfile 里这样写：

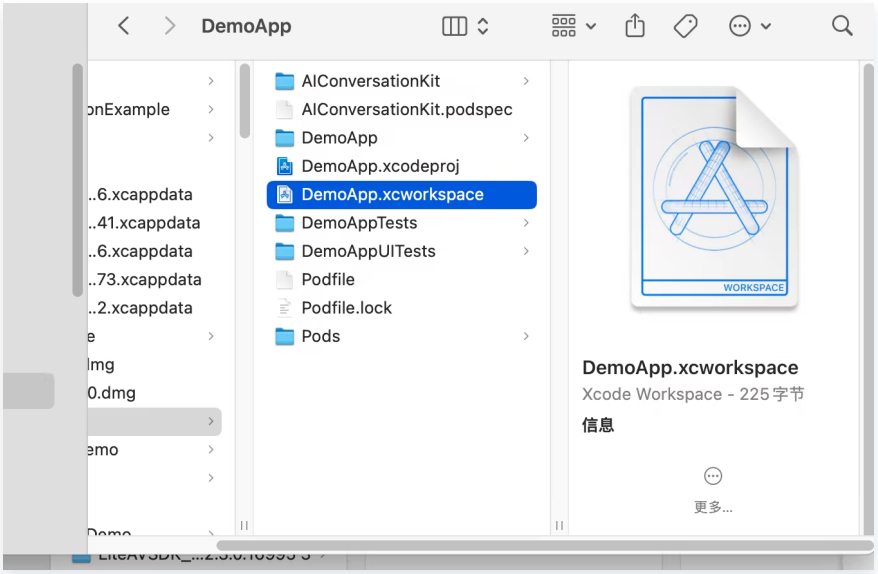
```
pod 'AIConversationKit', :path => 'AIConversationKit.podspec'
```



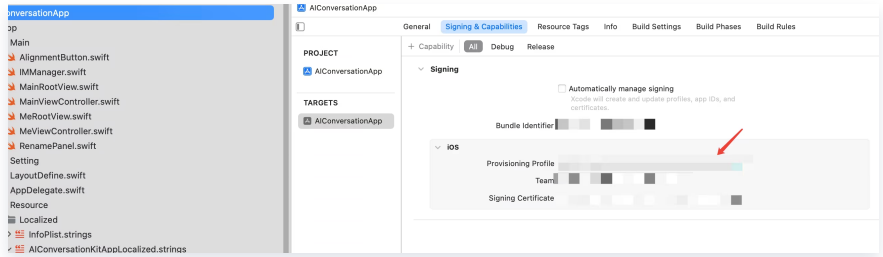

5. 执行 `pod install` 。




6. 打开 `.xcworkspace` ：



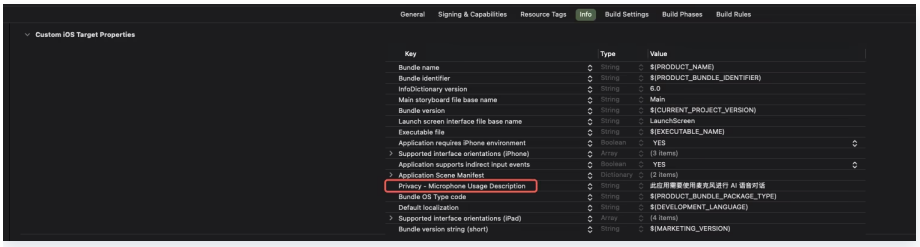
7. 配置证书信息：




## 工程配置

使用 AI 对话，需要授权麦克风使用权限。在 App 的 Info.plist 中添加以下麦克风声明，对应麦克风系统弹出授权对话框时的提示信息。

```
<key>NSMicrophoneUsageDescription</key>
<string>此应用需要使用麦克风进行 AI 语音对话</string>
```




## 步骤 3：实现登录逻辑

在您的项目中添加登录代码，这是使用 `AIConversationCoreView` 各项功能的关键前提：

> ⚠ **重要：**
> 推荐在您 App 自身的用户账户登录成功后，再调用 TUILogin.login，以确保登录业务逻辑的清晰和一致。

swift

```
import TUICore
import AIConversationKit

// 示例代码在 didFinishLaunchingWithOptions 完成登录，推荐您在自己登录业务完成后再调用登录服务
func application(_ application: UIApplication,
didFinishLaunchingWithOptions launchOptions:
[UIApplication.LaunchOptionsKey: Any]?) -> Bool {
        let userId = "denny"       // 请替换为您的 UserID
```

```
        let sdkAppId = 1600000001  // 请替换为开通服务控制台的 SDKAppID
        let sdkSecretKey = "xxx"       // 请替换为开通服务控制台的
SDKSecretKey
        let userSig = GenerateTestUserSig.genTestUserSig(sdkAppId:
sdkAppId, userId: userId, secretkey: sdkSecretKey)
        TUILogin.login(Int32(sdkAppId), userID: userId, userSig:userSig)
{
            print("login success")
        } fail: { code, message in
            print("login failed, code: \(code), error: \(message ??
"nil")")
        }
    return true
}
```

**登录接口参数说明**

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| SDKAppID | Int | 从 TRTC 控制台 > 应用管理 获取。 |
| UserID | String | 当前用户的唯一 ID，仅包含英文字母、数字、连字符和下划线。 |
| userSig | String | 用于腾讯云鉴权的票据。请注意：<br>开发环境：您可以采用本地 `GenerateTestUserSig.genTestUserSig` 函数生成 UserSig 或者通过 UserSig 辅助工具 生成临时的 UserSig。<br>生产环境：为了防止密钥泄露，请务必采用服务端生成 UserSig 的方式。详细信息请参见 服务端生成 UserSig 。<br>更多信息请参见 如何计算及使用 UserSig 。 |

# 搭建基础 AI 对话

## 步骤 1：添加 AIConversationCoreView

只需要将 `AIConversationCoreView` 添加到您的 ViewController 中即可。

```
import UIKit
import AIConversationKit

class YourAIViewController: UIViewController {
```

```
    private let conversationCoreView = AIConversationCoreView()

    override func viewDidLoad() {
        super.viewDidLoad()

        view.addSubview(conversationCoreView)
        conversationCoreView.frame = view.bounds
        conversationCoreView.autoresizingMask = [.flexibleWidth,
.flexibleHeight]
    }
}
```

## 步骤 2：开启 AI 对话

**通过调用 `startAIConversation` 接口开始 AI 对话，完整示例代码如下：**

```
import UIKit
import AIConversationKit
import TUICore

class YourAIViewController: UIViewController {

    private let conversationCoreView = AIConversationCoreView()

    override func viewDidLoad() {
        super.viewDidLoad()

        view.addSubview(conversationCoreView)
        conversationCoreView.frame = view.bounds
        conversationCoreView.autoresizingMask = [.flexibleWidth,
.flexibleHeight]

        startAIConversation()
    }

    private func startAIConversation() {
        var config = AIConversationConfig()

        let sdkAppId = 1600000001   // 1、替换 sdkAppId
        let sdkSecretKey = "xxx"    // 2、替换 sdkSecretKey
```

```
        let aiRobotId = "robot_\(TUILogin.getUserID() ?? "")"
        let aiRobotSig = GenerateTestUserSig.genTestUserSig(sdkAppId:
sdkAppId, userId: aiRobotId, secretkey: sdkSecretKey)
        config.agentConfig?.aiRobotId = aiRobotId
        config.agentConfig?.aiRobotSig = aiRobotSig

        config.secretId = "xxx"            // 3、替换 secretId
        config.secretKey = "xxx"           // 4、替换 secretKey

        // 5、替换 llmConfig
        config.llmConfig = "{\"LLMType\":\"openai\",\"Model\":\"hunyuan-
turbo-latest\",\"SystemPrompt\":\"您是一个个人助手
\",\"APIUrl\":\"https://hunyuan.cloud.tencent.com/openai/v1/chat/complet
ions\",\"APIKey\":\"xxxx\",\"History\":5,\"Streaming\":true}"
        // 6、替换 ttsConfig
        config.ttsConfig = "
{\"TTSType\":\"tencent\",\"AppId\":\"xxx\",\"SecretId\":\"xxx\",\"Secret
Key\":\"xxx\",\"VoiceType\":\"502001\",\"Speed\":1.25,\"Volume\":5,\"Pri
maryLanguage\":1,\"FastVoiceType\":\"\"}"

        conversationCoreView.startAIConversation(config: config) { code,
message in
            print("startAIConversation code: \(code), message: \
(message)")
        }
    }
}
```

1. sdkAppId 和 sdkSecretKey 使用 完成登录 后获取的数据即可。
2. 请到 准备工作 配置 AI 对话后台参数，包括基础配置、STT、LLM、TTS；然后点击右下角的**快速跑通**，切换到 iOS，获取 SecretId、SecretKey 和 Config 参数。




3. 将云 API 的 SecretId 和 SecretKey 复制到 `config.secretId` 和 `config.secretKey` 中，SecretId 和 SecretKey 请妥善保存，避免泄露。
4. 将 Config 信息复制到 JSON 解析工具中，例如 JSONUtil，将 LLMConfig 对应的字符串值复制到 `config.llmConfig`，将 TTSConfig 对应的字符串值复制到 `config.ttsConfig` 中。

> **说明：**
> - **开发环境**：如果您正在本地开发调试阶段，可以采用上述方式快速集成 AI 对话。该方法中账号信息很容易被反编译逆向破解，一旦您的密钥泄露，攻击者就可以盗用您的腾讯云流量。
> - **生产环境**：如果您的项目要发布上线，请将上述账号信息保存到服务端中，避免流量被盗用；相关对话配置，也可以保存到服务端中，方便动态调整 AI 对话效果。

## 步骤 3：结束 AI 对话

通过调用 `stopAIConversation` 接口结束 AI 对话，示例代码如下：

```
import UIKit
import AIConversationKit
import TUICore

class YourAIViewController: UIViewController {

    // ... 其他代码 ...

    private func stopAIConversation() {
        conversationCoreView.stopAIConversation() { code, message in
            print("stopAIConversation code: \(code), message: \
(message)")
```

```
        }
    }
}
```

## 步骤 4：显示 AI 对话字幕 (可选)

只需要将 `AIConversationSubtitleView` 添加到您的 ViewController 中，即可完成 AI 对话字幕的显示。

```
import UIKit
import AIConversationKit

class YourAIViewController: UIViewController {

    private let conversationCoreView = AIConversationCoreView()
    private let subtitleView = AIConversationSubtitleView()

    override func viewDidLoad() {
        super.viewDidLoad()

        view.addSubview(conversationCoreView)
        conversationCoreView.frame = view.bounds
        conversationCoreView.autoresizingMask = [.flexibleWidth,
.flexibleHeight]

        // 添加字幕视图
        view.addSubview(subtitleView)
        subtitleView.translatesAutoresizingMaskIntoConstraints = false
        NSLayoutConstraint.activate([
            subtitleView.leadingAnchor.constraint(equalTo:
view.leadingAnchor),
            subtitleView.trailingAnchor.constraint(equalTo:
view.trailingAnchor),
            subtitleView.topAnchor.constraint(equalTo: view.topAnchor,
constant: 120),
            subtitleView.bottomAnchor.constraint(equalTo:
view.bottomAnchor, constant: -120)
        ])
    }
}
```

## 步骤 5：控制麦克风开关（可选）

只需要将 `MicView` 添加到您的 ViewController 中，即可点击开关本地麦克风。

```
import UIKit
import AIConversationKit

class YourAIViewController: UIViewController {

    private let conversationCoreView = AIConversationCoreView()
    private let micView = MicView()

    override func viewDidLoad() {
        super.viewDidLoad()

        view.addSubview(conversationCoreView)
        conversationCoreView.frame = view.bounds
        conversationCoreView.autoresizingMask = [.flexibleWidth,
.flexibleHeight]

        // 添加 MicView
        view.addSubview(micView)
        micView.translatesAutoresizingMaskIntoConstraints = false
        NSLayoutConstraint.activate([
            micView.widthAnchor.constraint(equalToConstant: 120),
            micView.heightAnchor.constraint(equalToConstant: 120),
            micView.leadingAnchor.constraint(equalTo:
view.leadingAnchor, constant: 48),
            micView.bottomAnchor.constraint(equalTo:
view.safeAreaLayoutGuide.bottomAnchor, constant: -32)
        ])
    }
}
```

## 步骤 6：显示 AI 状态（可选）

只需要将 `AIStatusView` 添加到您的 ViewController 中，即可显示当前 AI 状态。

```
import UIKit
import AIConversationKit
```

```
class YourAIViewController: UIViewController {

    private let conversationCoreView = AIConversationCoreView()
    private let aiStatusView = AIStatusView()

    override func viewDidLoad() {
        super.viewDidLoad()

        view.addSubview(conversationCoreView)
        conversationCoreView.frame = view.bounds
        conversationCoreView.autoresizingMask = [.flexibleWidth,
.flexibleHeight]

        // 添加 AIStatusView
        view.addSubview(aiStatusView)
        aiStatusView.translatesAutoresizingMaskIntoConstraints = false
        NSLayoutConstraint.activate([
            aiStatusView.widthAnchor.constraint(equalToConstant: 180),
            aiStatusView.heightAnchor.constraint(equalToConstant: 130),
            aiStatusView.centerXAnchor.constraint(equalTo:
view.centerXAnchor),
            aiStatusView.bottomAnchor.constraint(equalTo:
view.safeAreaLayoutGuide.bottomAnchor, constant: -48)
        ])
    }
}
```

## 调节对话效果

为了在 AI 对话中获得最佳体验，清晰的声音采集至关重要。我们提供多种参数设置，帮助您在不同环境下都能获得清晰的语音交互效果。

## vadLevel 远场人声抑制

在多人交谈的环境中，其他人声可能会干扰 AI 对您声音的识别。通过调整人声抑制级别，我们可以有效过滤环境中的其他人声干扰，让 AI 更准确地专注于您的语音；您可以通过设置 config 中的 sttConfig.vadLevel 参数来达到期望的效果。

| 级别 | 效果描述 | 使用场景 |
| --- | --- | --- |

| | | |
|---|---|---|
| 0 | 关闭远场人声抑制功能 | 在嘈杂环境下AI容易被噪声打断，可能导致无法正常对话。 |
| 1 | 开启轻度抑制模式 | 适合相对安静的环境使用，例如家庭、图书馆。 |
| 2 | 推荐默认设置 | 适用于大多数日常场景（默认值）。 |
| 3 | 开启强力抑制模式 | 适合嘈杂环境使用，例如菜市场、车站。 |

**代码配置示例**

```
var config = AIConversationConfig()
config.sttConfig?.vadLevel = 3

conversationCoreView.startAIConversation(config: config, completion:
nil)
```

## interruptMode 打断模式

根据您的使用习惯，可以设置不同的打断方式，在 AI 说话时进行打断，使对话交互更加自然流畅；您可以通过设置 config 中的 agentConfig.interruptMode 参数来达到期望的效果。

| 模式 | 功能描述 | 适用场景 |
|---|---|---|
| 0 | 智能模式 | AI 正在说话时，可通过点击屏幕或用户说话打断；（默认值） |
| 1 | 手动打断 | AI 正在说话时，只能通过点击屏幕打断； |

**代码配置示例**

```
var config = AIConversationConfig()
config.agentConfig?.interruptMode = 1

conversationCoreView.startAIConversation(config: config, completion:
nil)
```

## language 语言识别

根据说话者使用的语言进行设置，AI 才能准确地将语音转换成对应的文字内容；您可以通过设置 config 中的 sttConfig.language 参数来达到期望的效果。

| 语言代码 | 语言名称 | 说明 |
|---|---|---|

| | | |
|---|---|---|
| "zh" | 中文 | 支持普通话识别（默认值） |
| "en" | 英语 | 支持英语识别 |

更多支持的语言，请参考 STTConfig 官方文档。

**代码配置示例**

```
var config = AIConversationConfig()
config.sttConfig?.language = "en"

conversationCoreView.startAIConversation(config: config, completion:
nil)
```

# 常见问题

## pod install 执行后本地安装找不到 AIConversationKit 最新版本?

如果无法安装 TUILiveKit 最新版本，请按以下步骤操作：

1. 在 Podfile 所在目录下删除 `Podfile.lock` 和 `Pods`，您可以选择手动删除或终端执行以下命令

```
// cd 到 Podfile 所在目录下

rm -rf Pods/
rm Podfile.lock
```

2. 在 Podfile 所在目录下执行 `pod install --repo-update`

```
// cd 到 Podfile 所在目录下

pod install --repo-update
```

## 每次进房都需要调用登录吗?

不需要。通常您只需要完成一次 `TUILogin.login` 调用即可，我们建议您将 `TUILogin.login` 和 `TUILogin.logout` 与自己的登录业务关联。

## Podfile 文件有没有示例配置可以参考?

您可以参考 GitHub AIConversationKit Example 工程 `Podfile` 示例文件。

## 部署目标版本过低警告？

**如果您遇到如下错误：**

```
clang: error: SDK does not contain 'libarclite' at the path
'/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xcto
olchain/usr/lib/arc/libarclite_iphonesimulator.a'; try increasing the
minimum deployment target
```

**部署目标版本过低警告：**

```
TXLiteAVSDK_Professional: iOS 8.0 → 需要 12.0+
TXIMSDK_Plus_iOS_XCFramework: iOS 8.0 → 需要 12.0+
TUICore: iOS 9.0 → 需要 12.0+
```

**请参考 GitHub AIConversationKit Example 工程 `Podfile` 示例文件，添加以下代码：**

```
post_install do |installer|
  installer.pods_project.targets.each do |target|
    target.build_configurations.each do |config|
      config.build_settings['CODE_SIGNING_ALLOWED'] = 'NO'
      config.build_settings['ENABLE_BITCODE'] = 'NO'
      config.build_settings['IPHONEOS_DEPLOYMENT_TARGET'] = 13.0
    end
  end
end
```

# 更多接入方案
# TRTC SDK 接入

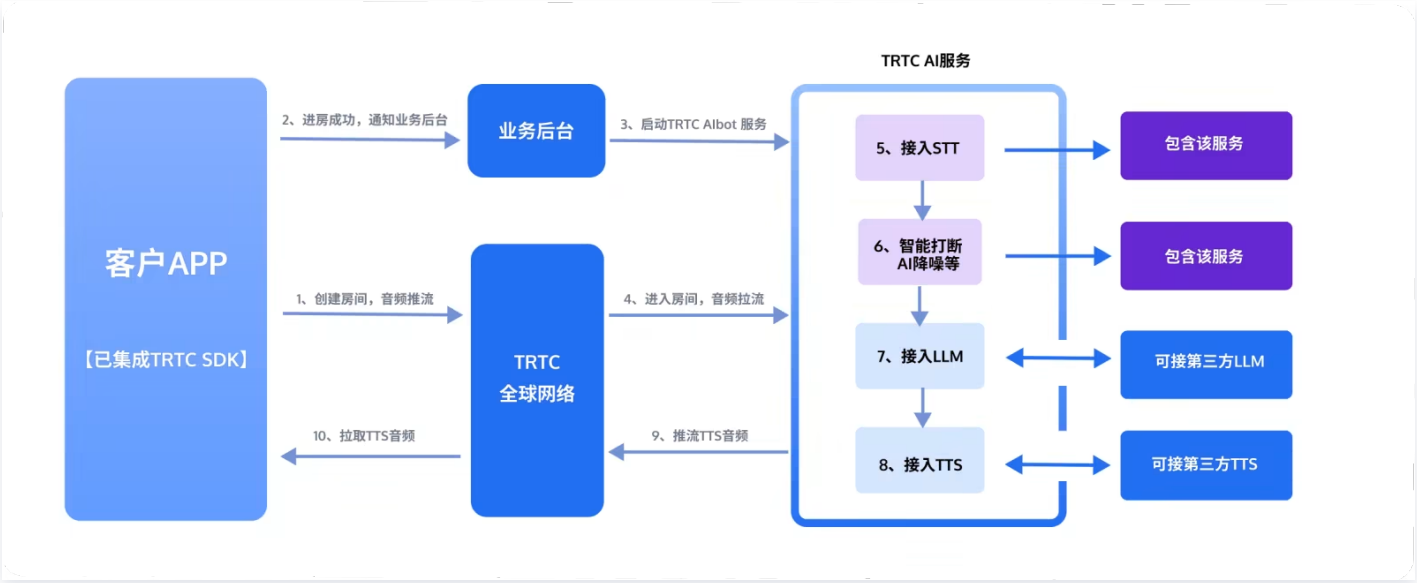
最近更新时间：2026-07-13 16:43:42

本文将介绍如何基于 TRTC SDK 实现 AI 实时对话的解决方案。

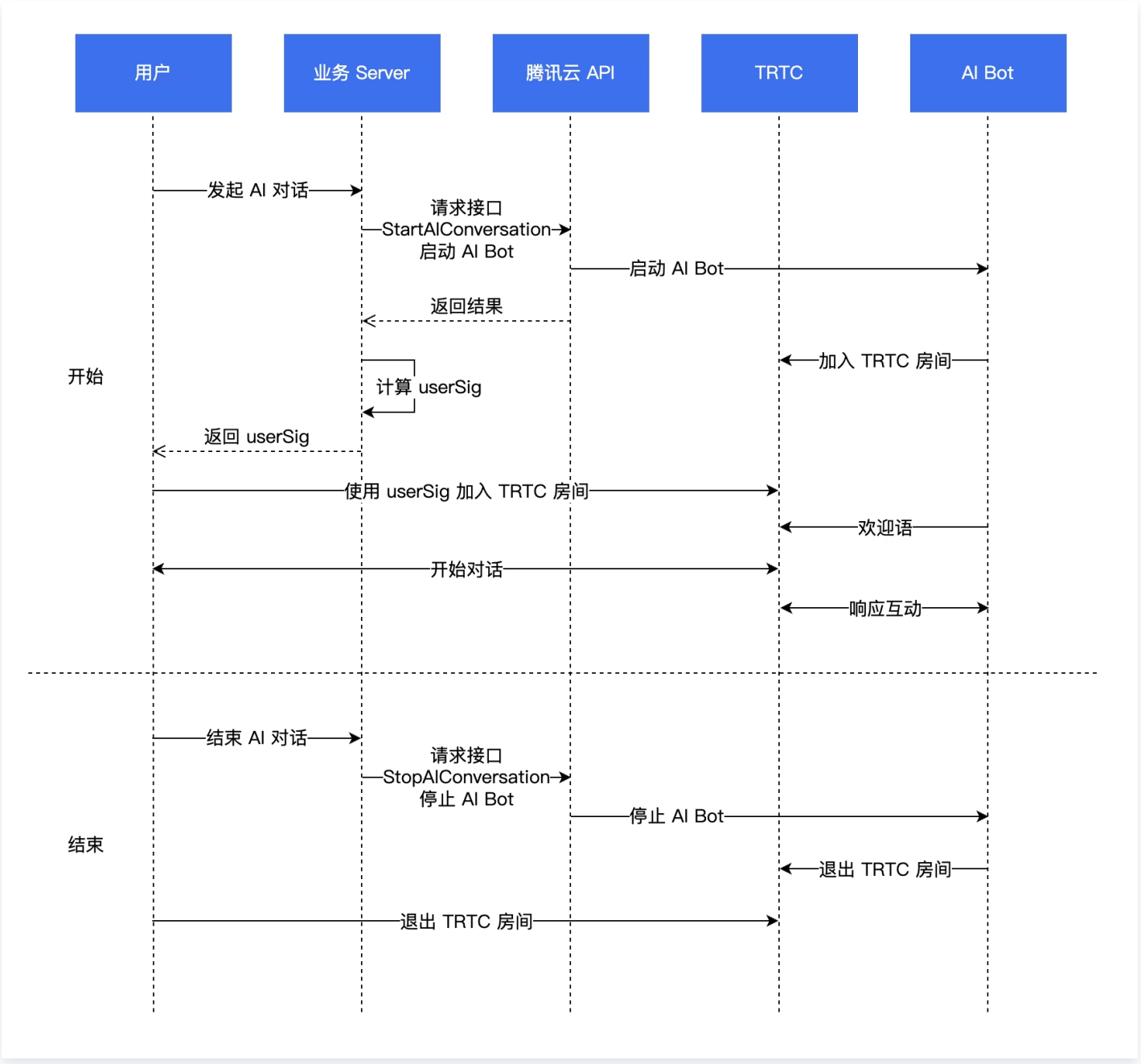
## 方案介绍

方案基于 TRTC SDK 调用 TRTC 服务，通过调用 AI 实时对话的接口，可以实现极低延迟的 AI 实时对话服务。本方案为您提供了非常灵活的集成方案，您可以根据业务的实际需求接入第三方的 LLM 和 TTS，实现高效的业务实践效果。在整体方案中，我们针对语音实时降噪、AI 智能打断、上下文管理都有较多的技术优化，不断提升用户体验。

## 方案架构图




## 业务流程图




# 集成指引

## 前提条件

> ⚠ **注意：**
> AI 实时对话调用会产生使用费用，具体详情请参见 AI 实时对话计费说明 。

1. 创建 TRTC 应用 。
2. 开通 语音转文字 服务。
3. 创建腾讯云 TTS（可选，支持第三方）。
4. 创建 LLM 应用：可以自选适合的大模型厂商进行注册。

## 步骤1：集成 TRTC SDK:

### 1. 导入 TRTC SDK 到项目中

- iOS 无 UI 集成指引
- Android 无 UI 集成指引
- Web&H5 无 UI 集成指引
- 小程序 无 UI 集成指引
- Flutter 无 UI 集成指引

### 2. 进入 TRTC 房间

- iOS 进入 TRTC 房间
- Android 进入 TRTC 房间
- Web&H5 进入TRTC房间
- 小程序 进入TRTC房间
- Flutter 进入TRTC房间

### 3. 发布音频流

Android&iOS&Flutter

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。虽然这个参数的名字叫做 quality，但并不是说质量越高越好，不同的业务场景有最适合的参数选择（这个参数更准确的含义是 scene）。
**AI 对话场景下推荐使用 SPEECH 模式**，该模式下的 SDK 音频模块会专注于提炼语音信号，尽最大限度的过滤周围的环境噪音，同时该模式下的音频数据也会获得较好的差质量网络的抵抗能力，因此该模式特别适合于“视频通话”和“在线会议”等侧重于语音沟通的场景。

Android

```
// 开启麦克风采集，并设置当前场景为：语音模式（高噪声抑制能力、强弱网络抗性）
mCloud.startLocalAudio(TRTCCloudDef.TRTC_AUDIO_QUALITY_SPEECH );
```

iOS&Mac

```
self.trtcCloud = [TRTCCloud sharedInstance];
// 开启麦克风采集，并设置当前场景为：语音模式（高噪声抑制能力、强弱网络抗性）
```

```
[self.trtcCloud startLocalAudio:TRTCAudioQualitySpeech];
```

**Flutter**

```
// 开启麦克风采集，并设置当前场景为：语音模式（高噪声抑制能力、强弱网络抗性）
trtcCloud.startLocalAudio(TRTCAudioQuality.speech);
```

### Web&H5

使用 trtc.startLocalAudio() 方法开启麦克风，并发布到房间。

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

### 小程序

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({
          sdkAppID: 1400xxxxx, // 您的腾讯云账号
          userID: 'trtc-user', //当前进房用户的userID
          userSig: 'xxxxxxx', // 您服务端生成的userSig
          roomID: 1234, // 您进房的房间号，
          enableMic: true, // 进房默认开启音频上行
        }),
    }, () => {
        this.TRTC.getPusherInstance().start() // 开始进行推流
    })
},
```

**说明：**

> AI 实时对话场景对音频采集端的降噪能力有较高要求，为了获得更好的体验，建议 打开 AI 降噪功能 。此外，我们还有专门针对 AI 实时对话场景训练的降噪模型，欢迎通过商务或 提交工单 联系我们。

## 步骤2：发起 AI 对话

### 开始 AI 对话：StartAIConversation

通过业务后台调用 开始 AI 对话任务 接口，来发起 AI 实时对话，调用成功后，AI 机器人会进入 TRTC 房间。

> ⚠ **注意：**
> - `RoomId` 需要和客户端进房的 `RoomId` 保持一致，并且房间号的类型（数字房间号、字符串房间号）也必须相同（即机器人和用户需要在同一个房间）。
> - `TargetUserId` 需要和客户端进房使用的 `UserId` 一致。
> - `LLMConfig` 和 `TTSConfig` 均为 JSON 字符串，需要正确配置才能成功发起 AI 实时对话。

**目前支持的 `LLMConfig` 和 `TTSConfig` 配置说明：**

- 大模型配置（LLMConfig）
- 文字转语音配置（TTSConfig）

> ⓘ **说明：**
> - 如果以上步骤都操作正确，现在已经可以和 AI 进行对话啦！
> - 目前 AI 对话服务并发限制100，如有更高并发需求，请提交工单 联系我们。

## 步骤3：接收 AI 对话字幕及 AI 状态

通过 TRTC SDK 接收自定义消息功能，在客户端上监听回调来接收实时字幕与 AI 状态等数据。**cmdID 固定是 1。**

### 接收实时字幕

消息格式

```
{
  "type": 10000, // 10000表示是下发的实时字幕
  "sender": "user_a", // 说话人的userid
  "receiver": [], // 接收者userid列表，该消息实际是在房间内广播
  "payload": {
     "text":"", // 语音识别出的文本
     "start_time":"00:00:01", // 这句话的开始时间
     "end_time":"00:00:02", // 这句话的结束时间
     "roundid": "xxxxx", // 唯一标识一轮对话
```

```
    "end": true // 如果为true，代表这是一句完整的话
  }
}
```

### 接收机器人状态

#### 消息格式

```
{
  "type": 10001, // 机器人的状态
  "sender": "user_a", // 发送者userid，这里是机器人的id
  "receiver": [], // 接受者userid列表，该消息实际是在房间内广播
  "payload": {
    "roundid": "xxx", // 唯一标识一轮对话
    "timestamp": 123,
    "state": 1,       //  1 聆听中  2 思考中  3 说话中  4 被打断  5 已说完
  }
}
```

### 示例代码

#### Android

```
@Override
public void onRecvCustomCmdMsg(String userId, int cmdID, int seq, byte[]
message) {
    String data = new String(message, StandardCharsets.UTF_8);
    try {
        JSONObject jsonData = new JSONObject(data);
        Log.i(TAG, String.format("receive custom msg from %s cmdId: %d
seq: %d data: %s", userId, cmdID, seq, data));
    } catch (JSONException e) {
        Log.e(TAG, "onRecvCustomCmdMsg err");
        throw new RuntimeException(e);
    }
}
```

### iOS

```
func onRecvCustomCmdMsgUserId(_ userId: String, cmdID: Int, seq: UInt32,
message: Data) {
    if cmdID == 1 {
        do {
            if let jsonObject = try JSONSerialization.jsonObject(with:
message, options: []) as? [String: Any] {
                print("Dictionary: \(jsonObject)")
                // handleMessage(jsonObject)
            } else {
                print("The data is not a dictionary.")
            }
        } catch {
            print("Error parsing JSON: \(error)")
        }
    }
}
```

### Web&H5

```
trtcClient.on(TRTC.EVENT.CUSTOM_MESSAGE, (event) => {
    let data = new TextDecoder().decode(event.data);
    let jsonData = JSON.parse(data);
    console.log(`receive custom msg from ${event.userId} cmdId:
${event.cmdId} seq: ${event.seq} data: ${data}`);

    if (jsonData.type == 10000 && jsonData.payload.end == false) {
        // 字幕中间状态
    } else if (jsonData.type == 10000 && jsonData.payload.end == true) {
       //  一句话说完了
    }
});
```

**说明：**

我们有更多 AI 对话客户端上的回调，具体可参见：AI 对话状态回调 、AI 对话字幕回调 、AI 对话指标回调 、AI 对话错误回调 。

## 步骤4：发送自定义消息

统一通过端上发送 TRTC 自定义消息，cmdID 固定是2。
参见 发送和接收消息 。

- 可以通过发送自定义的文本，跳过 ASR 过程，直接跟 AI Service 进行文字沟通。

```
  {
  "type": 20000, // 端上发送自定义文本消息
  "sender": "user_a", // 发送者userid， 服务端会check该userid是否有效
  "receiver": ["user_bot"], // 接受者userid列表，只需要填写机器人userid，服
务端会check该userid是否有效
  "payload": {
    "id": "uuid", // 消息id，可以使用uuid，排查问题使用
    "message": "xxx", // 消息内容
    "timestamp": 123 // 时间戳，排查问题使用
  }
}
```

- 可以通过发送打断信令来进行打断。

```
{
  "type": 20001, // 端上发送打断信令
  "sender": "user_a", // 发送者userid， 服务端会check该userid是否有效
  "receiver": ["user_bot"], // 接受者userid列表，只需要填写机器人userid，服
务端会check该userid是否有效
  "payload": {
    "id": "uuid", // 消息id，可以使用uuid，排查问题使用
    "timestamp": 123 // 时间戳，排查问题使用
  }
}
```

## 示例代码

### Android

```
public void sendInterruptCode() {
    try {
        int cmdID = 0x2;
```

```
        long time = System.currentTimeMillis();
        String timeStamp = String.valueOf(time/1000);
        JSONObject payLoadContent = new JSONObject();
        payLoadContent.put("timestamp", timeStamp);
        payLoadContent.put("id",
String.valueOf(GenerateTestUserSig.SDKAPPID) + "_" + mRoomId);

        String[] receivers = new String[]{robotUserId};

        JSONObject interruptContent = new JSONObject();
        interruptContent.put("type",
AICustomMsgType.AICustomMsgType_Send_Interrupt_CMD);
        interruptContent.put("sender", mUserId);
        interruptContent.put("receiver", new JSONArray(receivers));
        interruptContent.put("payload", payLoadContent);

        String interruptString = interruptContent.toString();
        byte[] data = interruptString.getBytes("UTF-8");

        Log.i(TAG, "sendInterruptCode :" + interruptString);

        mTRTCCloud.sendCustomCmdMsg(cmdID, data, true, true);
    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    } catch (JSONException e) {
        throw new RuntimeException(e);
    }
}
```

## iOS

```
@objc func interruptAi() {
    print("interruptAi")
    let cmdId = 0x2
    let timestamp = Int(Date().timeIntervalSince1970 * 1000)
    let payload = [
        "id": userId + "_\(roomId)" + "_\(timestamp)", // 消息id，可以使用
uuid，排查问题使用
```

```
        "timestamp": timestamp // 时间戳，排查问题使用
    ] as [String : Any]
    let dict = [
        "type": 20001,
        "sender": userId,
        "receiver": [botId],
        "payload": payload
    ] as [String : Any]
    do {
        let jsonData = try JSONSerialization.data(withJSONObject: dict,
options: [])
        self.trtcCloud.sendCustomCmdMsg(cmdId, data: jsonData, reliable:
true, ordered: true)
    } catch {
        print("Error serializing dictionary to JSON: \(error)")
    }
}
```

### Web&H5

```
const message = {
  "type": 20001,
  "sender": "user_a",
  "receiver": ["user_bot"],
  "payload": {
    "id": "uuid",
    "timestamp": 123
  }
};

trtc.sendCustomMessage({
  cmdId: 2,
  data: new TextEncoder().encode(JSON.stringify(message)).buffer
});
```

> ⚠ **注意：**
> 目前小程序端暂不支持接收和发送自定义消息，如果您想在小程序端实现接收字幕或发送消息等功能，需要使用 IM 提供的即时通信来实现，可通过商务或 提交工单 联系我们开通 IM 信令通道。

## 步骤5：停止 AI 对话，退出 TRTC 房间

1. 服务端停止 AI 对话任务。

   通过业务后台调用 停止 AI 对话 接口，停止该对话任务。
2. 客户端退出 TRTC 房间，建议参见 退出房间 。

## 步骤6：其他功能

### 1. 其他服务端接口

- **查询 AI 对话任务状态：DescribeAIConversation**

  可以查询 AI 对话任务状态。有 4 个值：

  1.1 `Idle` 表示任务未开始。

  1.2 `Preparing` 表示任务准备中。

  1.3 `InProgress` 表示任务正在运行。

  1.4 `Stopped` 表示任务已停止，正在清理资源中。
- **更新 AI 对话启动参数：UpdateAIConversation**

  可以在对话过程中，动态更新 TTS 的音色。
- **控制 AI 对话任务：ControlAIConversation**

  当您想让机器人主动播报文本的时候，可以使用该接口。

### 2. 开启服务端回调

可参见 AI 对话服务端回调

> ⚠ **注意：**
> - 回调地址在 TRTC 控制台设置，AI 实时对话回调。
> - 可配合 TRTC 房间与媒体回调 使用，丰富功能。

### 3. 其余高级功能介绍

| 功能 | 操作指引 |
| --- | --- |
| 智能打断 | 智能打断逻辑 |
| 实现上下文管理 | 如何实现上下文管理 |
| 调用 function call | 点击查看示例 |

版权所有：腾讯云计算（北京）有限责任公司

# 高级设置
# 大模型配置

最近更新时间：2025-09-09 14:42:21

本文主要介绍如何在 开始AI对话任务 StartAIConversation 接口中配置 `LLMConfig` 参数。

## OpenAI 标准协议

**说明：**
支持任何符合 OpenAI 标准协议的 LLM 模型，`LLMType` 填写 `openai`，其他参数根据实际情况填写即可。

### 参数说明

| 名称 | 类型 | 是否必填 | 描述 |
| --- | --- | --- | --- |
| LLMType | String | 是 | 大模型类型，只要是符合 OpenAI API 协议的大模型，都填写 `openai`。 |
| Model | String | 是 | 具体的模型名称，例如 gpt-4o、deepseek-chat。 |
| APIKey | String | 是 | 大模型的 APIKey。 |
| APIUrl | String | 是 | 大模型的 APIUrl。 |
| Streaming | Boolean | 否 | 是否流式，默认为 `false`，建议填 `true`。 |
| SystemPrompt | String | 否 | 系统提示词。 |
| Timeout | Float | 否 | 超时时间，取值范围 [1~50]，默认为 3 秒（单位：秒）。 |
| History | Integer | 否 | 设置 LLM 的上下文轮次，默认值：0（不提供上下文管理），最大值：50（提供最近 50 轮的上下文管理）。 |

| | | | |
|---|---|---|---|
| MaxTokens | Integer | 否 | 输出文本的最大 token 限制。 |
| Temperature | Float | 否 | 采样温度。 |
| TopP | Float | 否 | 采样的选择范围，控制输出 token 的多样性。 |
| ExtraBody | Object | 否 | 额外透传给大模型的参数，例如关闭思考：<br>`{ "enable_thinking": false }` |
| UserMessages | Object[] | 否 | 用户提示词，提供把更多上下文信息提供给大模型，帮助大模型更好地理解任务需求。 |
| MetaInfo | Object | 否 | 自定义参数，会放在请求的 body 中透传给大模型。 |

## 配置示例

### OpenAI 标准规范

```
"LLMConfig": {
    "LLMType": "openai",
    "Model": "gpt-4o",
    "APIKey": "api-key",
    "APIUrl": "https://api.openai.com/v1/chat/completions",
    "Streaming": true,
    "SystemPrompt": "你是一个个人助手",
    "Timeout": 3.0,
    "History": 5,
    "MetaInfo": {},
    "MaxTokens": 4096,
    "Temperature": 0.8,
    "TopP": 0.8,
    "UserMessages": [
      {
        "Role": "user",
```

```
        "Content": "content"
      },
      {
        "Role": "assistant",
        "Content": "content"
      }
    ],
    "ExtraBody": {
          "enable_thinking":false
    }
  }
```

## MiniMax

```
"LLMConfig":{
    "LLMType": "openai",
    "Model": "abab6.5s-chat",
    "Streaming": true,
    "SystemPrompt": "你是一个个人助手",
    "APIKey": "eyJhbGcixxxx",
    "APIUrl": "https://api.minimax.chat/v1/text/chatcompletion_v2",
    "History": 5,
    "MetaInfo": {}
 }
```

## 混元

```
"LLMConfig":{
      "LLMType": "openai",
      "Model": "hunyuan-standard", // hunyuan-turbo、hunyuan-standard
      "APIKey": "hunyuan-apikey",
      "APIUrl":
"https://hunyuan.cloud.tencent.com/openai/v1/chat/completions",
      "Streaming": true,
      "History": 10,
      "MetaInfo": {}
}
```

腾讯云智能体开发平台

> **说明：**
> 用于对接腾讯云智能体开发平台，具体可参考 官方文档 。

## 参数说明

| 名称 | 类型 | 是否必填 | 描述 |
| --- | --- | --- | --- |
| LLMType | String | 是 | 大模型类型，使用腾讯云智能体开发平台时，请填写 `tencent_lke` 。 |
| APIKey | String | 是 | 对应 LKE 产品应用发布得到的 `appkey` 。 |
| APIUrl | String | 否 | 可以不传，默认使用 `https://wss.lke.cloud.tencent.com/v1/qbot/chat/sse` 。 |
| Timeout | Float | 否 | 超时时间，取值范围 [1~50]， 默认为 3 秒（单位：秒）。 |
| FileInfos | Object[] | 否 | 文件信息，对应 `file_infos` 。 |
| VisitorLabels | Object[] | 否 | 知识标签（用于知识库中知识的检索过滤），对应 `visitor_labels` 。 |
| StreamingThrottle | Int | 否 | 流式回复频率控制，对应 `streaming_throttle` 。 |
| CustomVariables | Object | 否 | 自定义API参数的值，对应 `custom_variables` 。 |
| SystemRole | String | 否 | 角色指令（提示词），对应 `system_role` 。 |
| SessionId | String | 否 | 会话 ID，用于标识一个会话，对应 `session_id` 。 |
| VisitorBizId | String | 否 | 访客 ID，对应 `visitor_biz_id` 。 |

版权所有：腾讯云计算（北京）有限责任公司

> **说明：**
> 扩展参数的使用请详细阅读对话端接口文档（HTTP SSE）。

## 配置示例

```
"LLMConfig": {
    "LLMType": "tencent_lke",
    "APIUrl": "https://wss.lke.cloud.tencent.com/v1/qbot/chat/sse",
    "APIKey": "xxxxx",
    "Timeout": 10.0,
    "FileInfos": [],
    "VisitorLabels":[],
    "StreamingThrottle":5,
    "CustomVariables":{},
    "SystemRole":"",
    "SessionId":"",
    "VisitorBizId":""
}
```

### Dify

> **说明：**
> 用于对接 `Dify`，具体可参考官方文档。

## 参数说明

| 名称 | 类型 | 是否必填 | 描述 |
| --- | --- | --- | --- |
| LLMType | String | 是 | 大模型类型，使用 Dify 时，请填写 `dify`。 |
| APIKey | String | 是 | Dify 的 API 密钥。 |
| APIUrl | String | 否 | 选填，默认使用 `https://api.dify.ai/v1/chat-messages`。 |

| | | | |
|---|---|---|---|
| Timeout | Float | 否 | 超时时间，取值范围 [1~50]， 默认为 3 秒（单位：秒）。 |
| Streaming | Boolean | 否 | 是否流式，默认为 `false` ，建议填 `true` 。 |
| Inputs | Object | 否 | Dify LLM 请求 body 中的 `inputs` 参数，不填默认为{}。 |
| User | String | 否 | Dify LLM 请求 body 中的 `user` 参数，不填则默认填充为此次 `taskid` 。 |
| ConversationId | String | 否 | 建议不填，AI 对话默认支持 `conversation_id` ，如果不填会自动创建 `conversation_id` 。 |

## 配置示例

```
"LLMConfig": {
    "LLMType": "dify", // 必填，llm类型
    "APIKey": "your-secret-key", // 必填，dify的api密钥
    "APIUrl": "https://api.dify.ai/v1/chat-messages",  // 选填，默认"https://api.dify.ai/v1/chat-messages"
    "Timeout": 10.0,
    "Streaming": true, // 选填，是否流式，默认为false，建议填true
    "Inputs": {}, // 选填，dify llm请求body中的inputs参数，不填默认为{}
    "User": "your-user-id" // 选填，dify llm请求body中的user参数，不填则默认填充为此次taskid
    "ConversationId": "" //  选填，建议不填，AI对话默认支持conversation_id，如果不填会自动创建 conversation_id
}
```

### Coze

> **说明：**
> 用于对接 `Coze` ，具体可参考 官方文档（国内）、官方文档（国际）。

## 参数说明

| 名称 | 类型 | 是否必填 | 描述 |
| --- | --- | --- | --- |
| LLMType | String | 是 | 大模型类型，使用 Coze 时，请填写 `coze` 。 |
| APIKey | String | 是 | Coze 的 `secret token` 。 |
| BotId | String | 是 | Coze 的 `botid` 。 |
| APIUrl | String | 否 | 选填，默认使用 `https://api.coze.cn/v3/chat` ；如果是国外平台账号，请使用 `https://api.coze.com/v3/chat` 。 |
| Timeout | Float | 否 | 超时时间，取值范围 [1~50]， 默认为 3 秒（单位：秒）。 |
| Streaming | Boolean | 否 | 固定填 `true` ，Coze 暂时不支持非流式。 |
| UserId | String | 否 | Coze LLM 请求 body 中的 `user_id` 参数，不填则默认填充为此次 `taskid` 。 |

### 配置示例

```
"LLMConfig": {
    "LLMType": "coze", // 必填，llm类型
    "APIKey": "your-secret-key", // 必填，coze的secret token
    "Streaming": true, // 固定填true，coze暂时不支持非流式
    "Timeout": 10.0,
    "BotId": "botid", // 必填，coze的botid
    "UserId": "your-user-id" // 选填，coze llm请求body中的user_id参数，不填则默认填充为此次taskid
    "APIUrl": "https://api.coze.cn/v3/chat" // 选填，默认为https://api.coze.cn/v3/chat，如果是国外平台账号，则填写https://api.coze.com/v3/chat
}
```

## 大模型的请求

此外我们会在 HTTP header 中增加多个参数来辅助用户支持更复杂的逻辑：

```
X-Task-Id: <task_id_value> // 此任务的 id,
X-Request-Id: <request_id> // 此次请求的id， 重试会携带相同的requestId
X-Sdk-App-Id:  SdkAppId
X-User-Id: UserId
X-Room-Id: RoomId
X-Room-Id-Type: "0" // "0"表示数字房间号 "1"表示字符串房间号
```

版权所有：腾讯云计算（北京）有限责任公司

# 文字转语音配置

最近更新时间：2026-07-15 19:36:57

本文主要介绍如何配置 开始 AI 对话任务 StartAIConversation 接口中的 `TTSConfig` 参数。
您可以选择快速配置 TRTC 的 TTS 服务或者 自主接入第三方服务 。

## 使用 TRTC TTS 配置

若您在 AI 实时对话中选择使用 TRTC 实时语音合成服务，可参考下方配置快速接入。套餐权限开通和计费规则请参考 AI 智能语音计费 。
目前仅支持配置默认 TTS 版本，即 AI 智能语音 的语音合成 Flash 版。

```
{
  "TTSType": "flow",  // 【必填】固定为此值
  "VoiceId": "v-female-R2s4N9qJ", // 【必填】精品音色 ID /克隆音色 ID。不同音色ID 库参考下方音色列表
  "Model": "flow_02_turbo", // 【必填】当前默认的 TTS 模型版本（对应 Flash 版本）
  "Speed": 1.0,     //【可选】调节语速 范围 [0.5-2.0],默认 1.0; 取值越大，语速越快
  "Volume": 1.0,    // 【可选】调节音量 (0, 1] 默认值 1.0; 取值越大，音量越高
  "Pitch": 0,    // 【可选】调节语调 [-12,12],默认值为 0,其中 0 为原音色输出。
  "Language": "zh" //【可选】建议填写，目前支持填写中文：zh 英文：en 粤语方言：yue; 参数参考：(ISO 639-1)
}
```

**说明：**

- 目前 TRTC 的 TTS 服务支持一批中英精品音色，如果您有其他音色或者更多语种需求，欢迎通过商务联系我们。
- 若您需要在 AI 实时对话中自主接入 TTS 服务，请点击跳转至 自主接入 TTS 。

### 精品音色库

TRTC 语音合成精品音色库 Voice ID 请参考下方列表，您可以根据需求偏好选择音色进行配置填写。

| 音色名称 | 音色 ID | 支持语言 | 语言 ID |
| --- | --- | --- | --- |
| 威严男霸总 | v-male-Bk7vD3xP | 中文 | zh |
| 温柔姐姐 | v-female-R2s4N9qJ | 中文 | zh |

| | | | |
|---|---|---|---|
| 傲娇学姐 | v-female-m1KpW7zE | 中文 | zh |
| 夹子女生 | v-female-U8aT2yLf | 中文 | zh |
| 闲聊男声 | v-male-s5NqE0rZ | 中文 | zh |
| 自然男声 | v-male-W1tH9jVc | 中文 | zh |
| 客服小美（女） | female-kefu-xiaomei | 中文 | zh |
| 客服小心（女） | female-kefu-xiaoxin | 中文 | zh |
| 客服小悦（女） | female-kefu-xiaoyue | 中文 | zh |
| 客服小徐（男） | male-kefu-xiaoxu | 中文 | zh |
| 清晰女旁白 | v-female-p9Xy7Q1L | 美式英语 | en |
| 理性女讲解 | v-female-Z3x9LmQ2 | 美式英语 | en |
| 严谨男讲师 | v-male-A4b9KqP2 | 美式英语 | en |
| 权威男解读 | v-male-r7K2pQ9L | 美式英语 | en |
| 沉着男评审 | v-male-Q6p8ZxL3 | 美式英语 | en |
| 静心女教练 | v-female-T3s8BqL9 | 美式英语 | en |
| 温和男顾问 | v-male-P6q7LzD8 | 美式英语 | en |
| 内敛女播音 | v-female-M7k2PxL9 | 美式英语 | en |
| 淡然女配音 | v-female-S5n9QxJ4 | 美式英语 | en |
| 沉稳女配音 | v-female-T8m4WxP7 | 美式英语 | en |
| 深沉男评析 | v-male-D6p3KxN8 | 美式英语 | en |
| 温情女主持 | v-female-A9b3KfL2 | 美式英语 | en |
| 真挚女创作 | v-female-A7h2MxQ5 | 美式英语 | en |
| 温和男创作 | v-male-G4n7RxM3 | 美式英语 | en |
| 暖心男顾问 | v-male-H3p9LxK7 | 美式英语 | en |
| 真诚男主播 | v-male-R6n2MxT9 | 美式英语 | en |
| 自信女演员 | v-female-C8k4NxL6 | 美式英语 | en |

版权所有：腾讯云计算（北京）有限责任公司

| | | | |
|---|---|---|---|
| 阳光男演讲 | v-male-L7m5QxP4 | 美式英语 | en |
| 理性男评论 | v-male-N4k8TxR7 | 美式英语 | en |
| 理智女旁白 | v-female-B7k5WxN4 | 美式英语 | en |
| 雅致女解说 | v-female-k3P8sL0Q | 中文-粤语方言 | yue |
| 沉稳男解说 | v-male-L4s7PqZ9 | 中文-粤语方言 | yue |

## 用户自主接入 TTS

若您选择自主接入 TTS，则需要您自行准备相关 TTS 服务账号或 API Key，不同服务商的配置方式可参考下面章节。

### Tencent TTS

```
{
    "TTSType": "tencent", // String TTS 类型
    "AppId": 您的应用ID, // Integer 必填
    "SecretId": "您的密钥ID", // String 必填
    "SecretKey":  "您的密钥Key", // String 必填
    "VoiceType": 101001, // Integer 必填，音色 ID，包括标准音色与精品音色，精品音色拟真度更高，价格不同于标准音色，请参见语音合成计费概述。完整的音色 ID 列表请参见语音合成音色列表。
    "Speed": 1.25, // Float 非必填，语速，范围：[-2，6]，分别对应不同语速： -2：代表0.6倍 -1：代表0.8倍 0：代表1.0倍（默认） 1：代表1.2倍 2：代表1.5倍 6：代表2.5倍 如果需要更细化的语速，可以保留小数点后 2 位，例如0.5/1.25/2.81等。 参数值与实际语速转换，可参考 语速转换
    "Volume": 5, // Float 非必填，音量大小，范围：[0，10]，分别对应11个等级的音量，默认值为0，代表正常音量。
    "PrimaryLanguage": 1, // Integer 可选 主要语言 1-中文（默认） 2-英文 3-日文
    "FastVoiceType": "xxxx", // String 可选参数，快速声音复刻的参数
    "EmotionCategory":"angry", // String 非必填，控制合成音频的情感，仅支持多情感音色使用。取值：neutral（中性）、sad（悲伤）...
    "EmotionIntensity":150 // Integer 非必填，控制合成音频情感程度，取值范围为[50,200]，默认为 100；只有 EmotionCategory 不为空时生效。
}
```

参见：语音合成-音色列表

## MiniMax TTS

```
{
    "TTSType": "minimax",  // String TTS 类型,
    "Model": "speech-2.6-turbo",  // String 必填，请求的模型版本。
    "APIUrl": "https://api.minimax.chat/v1/t2a_v2",  // String 必填，接口地址
    "APIKey": "eyxxxx",  // String 必填
    "GroupId": "1810000000000000",  // String 必填，用户所属的组
    "VoiceType":"female-tianmei-jingpin",  // String 必填，请求的音色编号，对应 voice_id
    "Speed": 1.0,  // Float 非必填，生成声音的语速，范围[0.5,2]，默认值为1.0
    "Volume": 1.0,  // Float 非必填，生成声音的音量，范围（0,10]，默认值为1.0
    "Pitch": 0,  // Integer 非必填，生成声音的语调，范围[-12,12]，默认值为0
    "Emotion": "",  // String 非必填，控制合成语音的情绪，该参数仅对 speech-2.6-hd, speech-2.6-turbo, speech-02-hd, speech-02-turbo, speech-01-turbo,speech-01-hd 生效
    "LanguageBoost": "",  // String 非必填，增强对指定的小语种和方言的识别能力，设置后可以提升在指定小语种/方言场景下的语音表现。如果不明确小语种类型，则可以选择 "auto"，模型将自主判断小语种类型。
    "PronunciationDict": { "tone": [] }  // 非必填，替换需要特殊标注的文字、符号及对应的注音。
}
```

- 有关 MiniMax 更多模型型号和 API 参数详情可查看：MiniMax > 同步语音合成 。
- API 速率限制参见：MiniMax > 速率限制 。

## Azure TTS

```
{
    "TTSType": "azure", // 必填：String；  TTS类型
    "SubscriptionKey": "xxxxxxxx", // 必填：String；  用户订阅的 Key
    "Region": "chinanorth3",  // 必填：String 订阅的地区
    "VoiceName": "zh-CN-XiaoxiaoNeural", // 必填：String 音色名必填
    "Language": "zh-CN", // 必填：String 合成的语言
    "Rate": 1 // 选填：float 语速  0.5～2 默认为 1
}
```

参见：使用 SSML 来自定义语音和声音

## Cartesia TTS

```
{
    "TTSType": "cartesia",  // 必填: String;  TTS 类型
    "Model": "sonic-multilingual", //必填 模型型号
    "APIKey": "eyxxxx", //必填: 获取 API 密钥
    "VoiceId": "eda5bbff-1ff1-4886-8ef1-4e69a77640a0"  //必填 声音 id
https://play.cartesia.ai/
}
```

参见：Cartesia TTS

## ElevenLabs TTS

```
{
    "TTSType": "elevenlabs",  // 必填: String TTS 类型,
    "Model": "eleven_turbo_v2_5", //必填，模型类型
    "APIKey": "eyxxxx",
    "VoiceId": "eda5bbff-1ff1-4886-8ef1-4e69a77640a0"  //声音类型
https://elevenlabs.io/docs/api-reference/get-voices
}
```

参见：ElevenLabs TTS

## 自定义 TTS

```
{
  "TTSType": "custom", // String 必填
  "APIKey": "ApiKey", // String 必填，用来鉴权
  "APIUrl": "http://0.0.0.0:8080/stream-audio", // String，必填，对应服务商
的 TTS API URL
  "AudioFormat": "wav", // String, 非必填，期望输出的音频格式，如mp3，
ogg_opus，pcm，wav，默认为 wav，目前只支持pcm和wav，
  "SampleRate": 16000,  // Integer，非必填，音频采样率，默认为16000(16k)，推荐
值为16000
  "AudioChannel": 1,    // Integer，非必填，音频通道数，取值: 1 或 2  默认为1
}
```

具体协议规范请参见 自定义 TTS 协议 。

# 智能打断逻辑

最近更新时间：2025-07-16 15:56:02

## 智能打断概述

智能打断是实时语音对话中至关重要的功能，直接影响用户体验。TRTC AI 对话凭借其丰富的打断方案，为客户提供了灵活而强大的解决方案，满足各种场景需求。我们的智能打断功能主要分为自动打断和手动打断两种模式，能够有效提升对话的流畅性和自然度。

## 自动打断

自动打断是指系统在检测到特定条件时，根据用户的音频输入自动触发，支持两种主要规则：基于音频时长的打断和基于语义的打断。

1. **基于音频时长的打断**
   - 用户可动态配置打断时长，默认为500ms。
   - 可调范围：300ms（更灵敏）到5000ms（避免误打断）。
   - 建议：
     - 对于客服对话等高交互场景，设置较短时长（如300ms 或者 500ms）以提高响应速度。
     - 对于演讲或授课场景，设置较长时长（如1000ms）以避免误打断。
2. **基于语义的打断**

   基于语音活动检测（VAD）技术，当服务端检测到用户输入了一句语义完整的话时，会自动触发打断。
3. **自动打断与手动打断的配合**

   在自动打断模式下，仍可使用端上 SDK 发送自定义打断信令，实现特定的手动打断控制，提供更精细的交互体验。

## 手动打断

手动打断是指由用户主动发起，通过端上 SDK 发送自定义消息实现，为用户提供更直接的控制方式。通常依赖于用户界面上的按钮或快捷键。

1. **打断信令说明**

   手动打断使用如下 JSON 格式的自定义消息： 统一通过端上发送 TRTC 自定义消息，**cmdID 固定是2**。

```
{
  "type": 20001, // 固定type
  "sender": "user_a", // 发送者userid,服务端会验证有效性
  "receiver": ["user_bot"], // 接收者userid列表,填写机器人userid
  "payload": {
    "id": "uuid", // 消息id,用于问题排查
    "timestamp": 123 // 时间戳,用于问题排查
```

```
    }
}
```

2. **使用方法**

   通过 Web SDK 的 `sendCustomCmdMessage` 或 Native SDK 的 `sendCustomCmdMsg` 方法发送上述自定义消息即可实现手动打断。

## 应用场景示例

1. 客服对话：使用较短的自动打断时长，配合手动打断，实现快速响应和精准控制。
2. 在线教育：采用较长的自动打断时长，避免误打断学生发言，教师可使用手动打断进行必要干预。
3. 多人会议：结合自动打断和手动打断，灵活管理发言顺序和时长。

# 如何实现上下文管理

最近更新时间：2024-11-08 18:05:32

大模型的上下文管理非常重要，它可以让大模型根据聊天历史进行更精准的内容回答。TRTC AI 提供了基本的上下文管理能力，同时也支持开发者实现自己的丰富上下文管理方案。

## 基本的上下文管理

TRTC AI 提供了基本的上下文管理功能。在 LLMConfig 参数中，我们引入了 History 参数来控制上下文管理：
History：

- 设置 LLM 的上下文轮次默认值0（不提供上下文管理）。
- 最大值：50（提供最近50轮的上下文管理）。

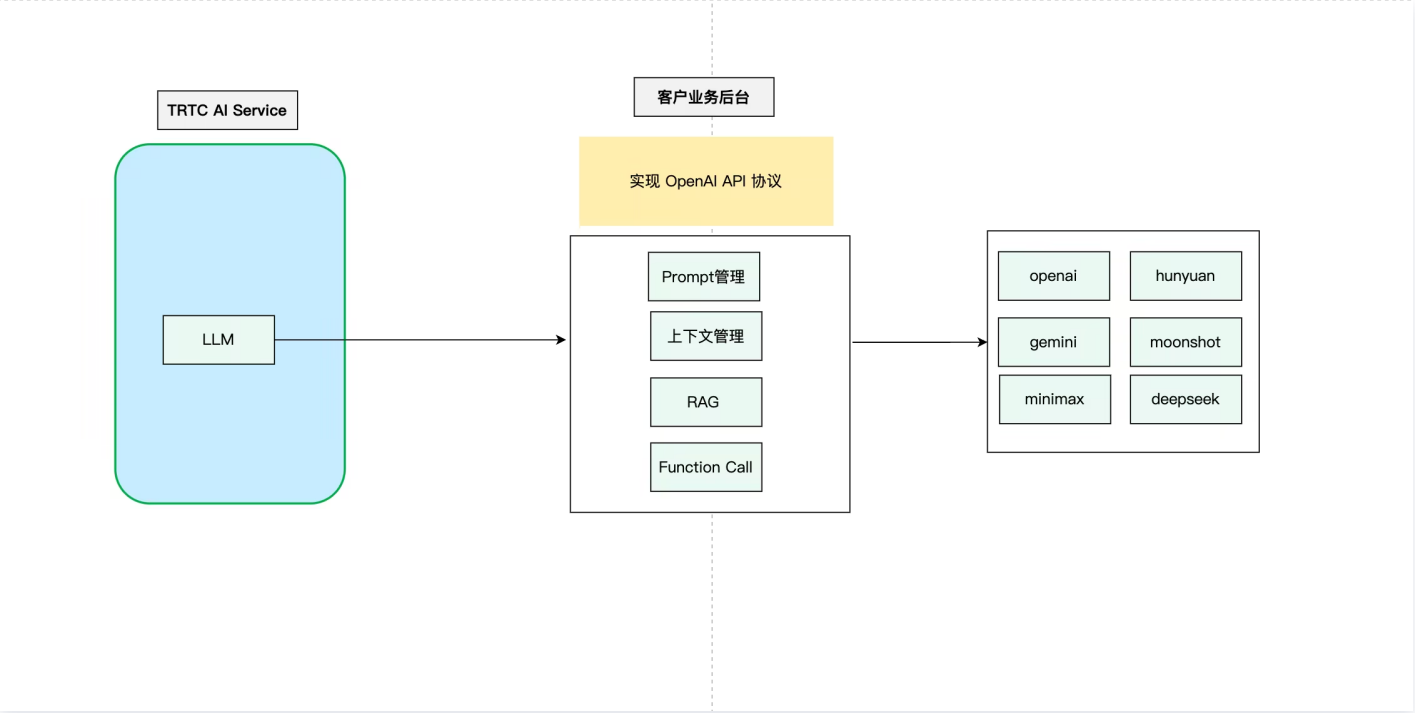
相关配置示例如下：

```
"LLMConfig": {
            "LLMType": "openai",
            "Model":"gpt-4o",
            "APIKey":"api-key",
            "APIUrl":"https://api.openai.com/chat/completions",
            "Streaming": true,
            "SystemPrompt": "你是一个个人助手",
            "Timeout": 3.0,
            "History": 5     //  最大支持 50 轮对话，默认为 0
    }
```

## 自定义上下文管理

TRTC AI 对话服务支持标准的 OpenAI 规范，这使得开发者能够在自己的业务中实现定制化的上下文管理。实现流程如下：

版权所有：腾讯云计算（北京）有限责任公司




这个流程图展示了自定义上下文管理的基本步骤。开发者可以根据自己的具体需求对这个流程进行调整和优化。

## 实现示例

开发者可以在自己的业务后台实现与 OpenAI API 兼容的大模型接口，并将封装了上下文逻辑的大模型请求发送给第三方大模型。以下是一个简化的示例代码：

```
import time
from fastapi import FastAPI, HTTPException
from fastapi.middleware.cors import CORSMiddleware
from pydantic import BaseModel
from typing import List, Optional
from langchain_core.messages import HumanMessage, SystemMessage
from langchain_openai import ChatOpenAI


app = FastAPI(debug=True)

# 添加 CORS 中间件
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
```

```
)


class Message(BaseModel):
    role: str
    content: str


class ChatRequest(BaseModel):
    model: str
    messages: List[Message]
    temperature: Optional[float] = 0.7


class ChatResponse(BaseModel):
    id: str
    object: str
    created: int
    model: str
    choices: List[dict]
    usage: dict


@app.post("/v1/chat/completions")
async def chat_completions(request: ChatRequest):
    try:
        # 将请求消息转换为 LangChain 消息格式
        langchain_messages = []
        for msg in request.messages:
            if msg.role == "system":

langchain_messages.append(SystemMessage(content=msg.content))
            elif msg.role == "user":

langchain_messages.append(HumanMessage(content=msg.content))


        # add more historys

        # 使用 LangChain 的 ChatOpenAI 模型
```

```
        chat = ChatOpenAI(temperature=request.temperature,
                          model_name=request.model)
        response = chat(langchain_messages)
        print(response)

        # 构造符合 OpenAI API 格式的响应
        return ChatResponse(
            id="chatcmpl-" + "".join([str(ord(c))
                                      for c in response.content[:8]]),
            object="chat.completion",
            created=int(time.time()),
            model=request.model,
            choices=[{
                "index": 0,
                "message": {
                    "role": "assistant",
                    "content": response.content
                },
                "finish_reason": "stop"
            }],
            usage={
                "prompt_tokens": -1,  # LangChain 不提供这些信息,所以我们使用
占位值
                "completion_tokens": -1,
                "total_tokens": -1
            }
        )
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

详情请点击 查看示例 。

# AI 对话 SDK 端回调
# AI 对话状态回调

最近更新时间：2025-10-15 21:24:42

TRTC AI 对话提供了丰富的状态回调功能。这些状态回调通过 TRTC 的自定义消息发送，使得在客户端上可以方便地实现丰富的状态切换，例如"聆听中"、"思考中"、“说话中”等状态。
通过 TRTC SDK 接收自定义消息功能，在客户端上监听回调来接收实AI 状态的数据。**cmdID 固定是1。**

## 功能特点

1. 实时性：状态变化可以实时传递给所有参与者。
2. 灵活性：使用自定义消息格式，便于集成和扩展。
3. 多状态支持：包括聆听、思考、说话和被打断等多种状态。

## 消息格式

状态回调消息使用 JSON 格式，具体字段如下：

| 字段 | 类型 | 描述 |
| --- | --- | --- |
| type | Number | 消息类型，10001 表示机器人状态 |
| sender | String | 发送者的 userid，这里是机器人的 id |
| receiver | Array | 接收者 userid 列表，该消息实际在房间内广播 |
| payload | Object | 消息负载，包含状态详细信息 |

payload 对象包含以下字段：

| 字段 | 类型 | 描述 |
| --- | --- | --- |
| roundid | String | 唯一标识一轮对话的 ID |
| timestamp | Number | 时间戳，表示状态变化的具体时间 |
| state | Number | 机器人当前状态代码 |

状态代码说明：

| 状态代码 | 描述 |
| --- | --- |
| 1 | 聆听中 |

| | |
|---|---|
| 2 | 思考中 |
| 3 | 说话中 |
| 4 | 被打断 |
| 5 | 已说完 |

## 示例消息

```
{
  "type": 10001,
  "sender": "ai_assistant_001",
  "receiver": [],
  "payload": {
    "roundid": "conversation_789012",
    "timestamp": 1629384755,
    "state": 2
  }
}
```

## 应用场景

1. UI 反馈：根据不同状态显示相应的界面元素，如“正在聆听”的动画。
2. 交互控制：在 AI 思考或说话时，可以禁用某些用户输入以避免中断。
3. 调试与分析：利用状态信息进行会话分析，优化用户体验。

版权所有：腾讯云计算（北京）有限责任公司

# AI 对话字幕回调

最近更新时间：2025-01-08 10:03:32

TRTC AI 对话提供了显示实时字幕的能力。实时字幕通过 TRTC 的自定义消息发送，可以实现与音频对话毫秒级别的同步。
通过 TRTC SDK 接收自定义消息功能，在客户端上监听回调来接收实AI 字幕的数据。**cmdID 固定是1。**

## 功能特点

1. 实时性：字幕与音频对话同步，延迟在毫秒级别。
2. 灵活性：使用自定义消息格式，便于集成和扩展。

## 消息格式

实时字幕消息使用 JSON 格式，具体字段如下：

| 字段 | 类型 | 描述 |
| --- | --- | --- |
| type | Number | 消息类型，10000表示实时字幕 |
| sender | String | 说话人的 userid |
| receiver | Array | 接收者 userid 列表，该消息实际在房间内广播 |
| payload | Object | 消息负载，包含字幕详细信息 |

payload 对象包含以下字段：

| 字段 | 类型 | 描述 |
| --- | --- | --- |
| text | String | 语音识别出的原文本 |
| start_time | String | 这句话的开始时间，格式为 "HH:MM:SS" |
| end_time | String | 这句话的结束时间，格式为 "HH:MM:SS" |
| roundid | String | 唯一标识一轮对话的 ID |
| end | Boolean | 如果为 true，代表这是一句完整的话 |

## 示例消息

```
{
  "type": 10000,
```

```
  "sender": "user_a",
  "receiver": [],
  "payload": {
    "text": "你好，很高兴认识你。",
    "start_time": "00:00:01",
    "end_time": "00:00:03",
    "roundid": "conversation_123456",
    "end": true
  }
}
```

## 实现注意事项

1. 消息处理：接收方需要正确解析 JSON 消息，并根据 type 字段识别实时字幕消息。
2. 时间同步：使用 start_time 和 end_time 确保字幕与音频正确对齐。
3. 对话分段：使用 end 字段判断一句话是否结束，可用于界面更新或存储完整对话。

## Web SDK 自定义消息解析如下

```
    trtcClient.on(TRTC.EVENT.CUSTOM_MESSAGE, (event) => {
        let data = new TextDecoder().decode(event.data);
        let jsonData = JSON.parse(data);
        console.log(`receive custom msg from ${event.userId} cmdId:
${event.cmdId} seq: ${event.seq} data: ${data}`);

        if (jsonData.type == 10000 && jsonData.payload.end == false) {
            // 字幕中间状态
        } else if (jsonData.type == 10000 && jsonData.payload.end ==
true) {
            //  一句话说完了
        }
    });
```

# AI 对话指标回调

最近更新时间：2025-03-14 11:39:12

TRTC AI 对话提供了丰富的指标回调功能。这些指标回调通过 TRTC 的自定义消息发送，使得在客户端上可以方便监控对话运行情况，如 LLM、TTS 调用耗时以及性能数据通过 TRTC 自定义消息实时推送到终端，监控 AI 对话的耗时与质量等。
通过 TRTC SDK 接收自定义消息功能，在客户端上监听回调来接收实AI 对话指标的数据。**cmdID 固定是1。**

| 字段 | 类型 | 描述 |
| --- | --- | --- |
| type | Number | 消息类型，10020 表示AI服务的调用回调 |
| sender | String | 发送者的 userid，这里是机器人的 id |
| payload | Object | 消息负载，包含指标详细信息 |

payload 对象包含以下字段：

| 字段 | 类型 | 描述 |
| --- | --- | --- |
| metric | String | 调用指标名称如下：<br>• asr_latency<br>• llm_network_latency<br>• llm_first_token<br>• tts_network_latency<br>• tts_first_frame_latency<br>• tts_discontinuity<br>• interruption |
| value | Number | 调用指标 |
| tag | Object | 指标关联的 tag |

tag 对象包含以下字段：

| 字段 | 类型 | 描述 |
| --- | --- | --- |
| roundid | String | 对话轮次 ID |

指标名称说明：

| 状态代码 | 描述 |
| --- | --- |

| | |
|---|---|
| asr_latency | asr 延迟。注意：指标包含启动 AI 对话时 VadSilenceTime 所设置的时间 |
| llm_network_latency | llm 请求的网络耗时 |
| llm_first_token | llm 首 token 耗时，指标包含网络耗时 |
| tts_network_latency | tts 请求的网络耗时 |
| tts_first_frame_latency | tts 首帧耗时，指标包含网络耗时 |
| tts_discontinuity | tts 未连续的次数，代表 tts 流式请求播放完成之后，下一个请求还没有返回结果，通常是 tts 延迟比较高导致 |
| interruption | 表示此轮对话被打断 |

版权所有：腾讯云计算（北京）有限责任公司

# AI 对话错误回调

最近更新时间：2025-03-14 11:39:12

TRTC AI 对话提供了错误回调功能。这些错误回调通过 TRTC 的自定义消息发送，使得在客户端上可以方便监控对话运行情况，如 LLM、TTS 调用失败时会将数据通过 TRTC 自定义消息实时推送到终端，监控 AI 对话的质量与成功率等。
通过 TRTC SDK 接收自定义消息功能，在客户端上监听回调来接收实AI 对话错误回调的数据。**cmdID 固定是 1。**

| 字段 | 类型 | 描述 |
| --- | --- | --- |
| type | Number | 消息类型，10030 表示 AI 服务的错误回调 |
| sender | String | 发送者的 userid，这里是机器人的 id |
| payload | Object | 消息负载，包含详细信息 |

**payload 对象包含以下字段：**

| 字段 | 类型 | 描述 |
| --- | --- | --- |
| metric | String | 调用指标名称如下<br>• asr_error<br>• llm_error<br>• tts_error |
| tag | Object | 指标关联的 tag |

**tag 对象包含以下字段：**

| 字段 | 类型 | 描述 |
| --- | --- | --- |
| roundid | String | 对话轮次 ID |
| code | Number | 调用错误码 |
| message | String | 错误信息的详细描述 |

**指标名称说明：**

| 状态代码 | 描述 |
| --- | --- |
| asr_error | asr 识别出现错误 |

版权所有：腾讯云计算（北京）有限责任公司

| | |
|---|---|
| llm_error | llm 请求出现错误 |
| tts_error | tts 请求出现错误 |

**错误码列表**

| 服务类别 | 错误码 | 错误描述 |
|---|---|---|
| ASR | 30100 | 请求超时 |
| | 30102 | 内部错误 |
| LLM | 30200 | 请求 LLM 超时 |
| | 30201 | LLM 请求被频率限制 |
| | 30202 | LLM 服务返回失败 |
| TTS | 30300 | 请求 TTS 服务超时 |
| | 30301 | TTS 请求被频率限制 |
| | 30302 | TTS 服务返回失败 |

**说明：**
详细的错误码，例如第三方 LLM、TTS 的错误码会包装在 `tag.message` 中，便于快速定位问题以及提示用户出错。

# AI 对话服务端回调

最近更新时间：2025-09-25 17:39:51

本文用于介绍 AI 服务（ AI 实时对话 ）相关的云 API 接口产生的事件，以 HTTP/HTTPS 请求的形式通知到您的服务器。您可以向腾讯云提供相关的配置信息来开通该服务。您也可以结合TRTC的 房间与媒体回调 使用，实现更多自定义逻辑。

## 配置信息

实时音视频 TRTC 控制台支持自助配置回调信息，配置完成后即可接收事件回调通知。详细操作指引请参见 回调配置 。

> ⚠ **注意：**
> 您需要提前准备以下信息：
> - **必要项**：接收回调通知的 HTTP/HTTPS 服务器地址。
> - **可选项**：计算签名的 密钥 key，由您自定义一个最大32个字符的 key，以大小写字母及数字组成。

## 超时重试

事件回调服务器在发送消息通知后，5秒内没有收到您的服务器的响应，即认为通知失败。首次通知失败后会立即重试，后续失败会以10秒的间隔继续重试，直到消息存续时间超过1分钟，不再重试。

## 事件回调消息格式

事件回调消息以 HTTP/HTTPS POST 请求发送给您的服务器，其中：

- **字符编码格式**：UTF-8。
- **请求**：body 格式为 JSON。
- **应答**：HTTP STATUS CODE = 200，服务端忽略应答包具体内容，为了协议友好，建议客户应答内容携带 JSON： {"code":0}。
- **包体示例**：下述为“启动AI对话任务成功”事件的包体示例。

```
{
    "EventGroupId":  9,
    "EventType": 901,
    "CallbackTs": 1687770730166,
    "EventInfo": {
        "EventMsTs": 1622186275757,
        "TaskId": "hKPD2Q7kBVzu-
6ezFiqmcEBJQCykqbZrS9OOTE46uYlb4NvQDIaEXlpOlLXFtGBiado5oP0zfLDZs",
        "RoomId": "1234",
```

```
        "RoomIdType": 0,
        "Payload": {
            "Status": 0
        }
    }
}
```

# 参数说明

## 回调消息参数

- 事件回调消息的 header 中包含以下字段：

| 字段名 | 值 |
| --- | --- |
| Content-Type | application/json |
| Sign | 签名值 |
| SdkAppId | sdk application id |

- 事件回调消息的 body 中包含以下字段：

| 字段名 | 类型 | 含义 |
| --- | --- | --- |
| EventGroupId | Number | 事件组 ID，混流转推事件固定为4 |
| EventType | Number | 回调通知的事件类型 |
| CallbackMsTs | Number | 事件回调服务器向您的服务器发出回调请求的 Unix 时间戳，单位为毫秒 |
| EventInfo | JSON Object | 事件信息 |

## 事件组 ID

| 字段名 | 值 | 含义 |
| --- | --- | --- |
| EVENT_GROUP_AI_SERVICE | 9 | AI 服务事件组 |

## 事件类型

| 字段名 | 值 | 含义 |
| --- | --- | --- |
| EVENT_TYPE_AI_SERVICE_START | 901 | AI 任务开始状态回调 |
| EVENT_TYPE_AI_SERVICE_STOP | 902 | AI 任务结束状态回调 |
| EVENT_TYPE_AI_SERVICE_MSG | 903 | 回调ASR识别的完整一句话或 LLM 返回的完整内容 |
| EVENT_TYPE_AI_START_OF_SPEECH | 904 | ASR 识别一句话的开始回调 |
| EVENT_TYPE_AI_SPEAKING_FINISHED | 905 | 在一轮对话中，AI 说完话的回调 |
| EVENT_TYPE_AI_METRIC_MESSAGE | 906 | AI 服务中调用 LLM/TTS 的指标回调 |
| EVENT_TYPE_AI_ERROR_METRIC_CALLBACK | 908 | AI 服务的调用指标错误回调 |
| EVENT_TYPE_AI_SESSION_STATUS_CALLBACK | 909 | AI 会话准备就绪回调，表示音视频通道已建立，可以进行对话。 |
| EVENT_TYPE_AI_ROUND_FINISHED | 910 | 一轮对话结束的回调 |

## 事件类型为（901）时事件信息的定义：

| 字段名 | 类型 | 含义 |
| --- | --- | --- |
| EventMsTs | String | 事件发生的 Unix 时间戳，单位为毫秒 |
| TaskId | String | AI 任务 ID |
| RoomId | String | TRTC 的房间 ID |
| RoomIdType | Integer | • 0：表示数字房间号<br>• 1：表示字符串房间号 |
| Payload.Status | Number | • 0：启动 AI 任务成功<br>• 1：启动 AI 任务失败 |

```
{
```

```
    "EventGroupId": 9,
    "EventType": 901,
    "CallbackTs": 1687770730166,
    "EventInfo": {
        "EventMsTs": 1622186275757,
        "TaskId": "xx",
        "RoomId": "1234",
        "RoomIdType": 0,
        "Payload": {
            "Status": 0
        }
    }
}
```

## 事件类型为（902）时事件信息的定义：

| 字段名 | 类型 | 含义 |
| --- | --- | --- |
| EventMsTs | String | 事件发生的 Unix 时间戳，单位为毫秒 |
| TaskId | String | AI 任务 ID |
| RoomId | String | TRTC 的房间 ID |
| RoomIdType | Integer | • 0：表示数字房间号<br>• 1：表示字符串房间号 |
| Payload.LeaveCode | Integer | • 0：正常调用停止接口后任务退出<br>• 1：业务自己踢掉转录机器人后任务退出<br>• 2：业务自己解散房间后任务退出<br>• 3：TRTC 服务端踢掉机器人<br>• 4：TRTC 服务端解散房间<br>• 98：内部异常错误，建议业务进行重试<br>• 99：代表房间内除了转录机器人没有其他用户流，超过指定时间退出 |

```
{
    "EventGroupId": 9,
    "EventType": 902,
    "CallbackTs": 1687770730166,
    "EventInfo": {
```

```
        "EventMsTs": 1622186275757,
        "TaskId": "xx",
        "RoomId": "1234",
        "RoomIdType": 0,
        "Payload": {
            "LeaveCode": 0
        }
    }
}
```

## 事件类型为（903）时事件信息的定义：

| 字段名 | 类型 | 含义 |
| --- | --- | --- |
| EventMsTs | String | 事件发生的 Unix 时间戳，单位为毫秒 |
| TaskId | String | AI 任务 ID |
| RoomId | String | TRTC 的房间 ID |
| RoomIdType | Integer | • 0：表示数字房间号<br>• 1：表示字符串房间号 |
| Payload | JSON Object | 为 JSON 对象，与客户端自定义消息回调格式一致<br>{<br>"UserId":"",<br>"Text":"",<br>"StartTimeMs":1234,<br>"EndTimeMs":1269,<br>"RoundId":"xxxxxx" // 一轮对话的唯一id<br>} |

```
{
    "EventGroupId":  9,
    "EventType":     903,
    "CallbackTs":    1687770730166,
    "EventInfo": {
        "EventMsTs": 1622186275757,
       "TaskId": "xx",
        "RoomId": "1234",
        "RoomIdType": 0,
        "Payload": {
```

```
            "UserId":"",
            "Text":"",
            "StartTimeMs":1234,
            "EndTimeMs":1269,
            "RoundId":"xxxxxx"
        }
    }
}
```

## 事件类型为（904）时事件信息的定义：

| 字段名 | 类型 | 含义 |
| --- | --- | --- |
| EventMsTs | String | 事件发生的 Unix 时间戳，单位为毫秒 |
| TaskId | String | AI 任务 ID |
| RoomId | String | TRTC 的房间 ID |
| RoomIdType | Integer | • 0：表示数字房间号<br>• 1：表示字符串房间号 |
| Payload | JSON Object | 为 JSON 对象<br>{<br>"UserId": "xxx",<br>"RoundId": "xxxxx" // 一轮对话的唯一id<br>} |

```
{
    "EventGroupId":  9,
    "EventType":     904,
    "CallbackTs":    1687770730166,
    "EventInfo": {
        "EventMsTs": 1622186275757,
        "TaskId": "xx",
        "RoomId": "1234",
        "RoomIdType": 0,
        "Payload": {
           "UserId": "xxx",
           "RoundId": "xxxxx"
        }
```

```
    }
}
```

## 事件类型为（905）时事件信息的定义：

| 字段名 | 类型 | 含义 |
| --- | --- | --- |
| EventMsTs | String | 事件发生的 Unix 时间戳，单位为毫秒 |
| TaskId | String | AI 任务 ID |
| RoomId | String | TRTC 的房间 ID |
| RoomIdType | Integer | • 0：表示数字房间号<br>• 1：表示字符串房间号 |
| Payload | JSON Object | 为 JSON 对象<br>{<br>"UserId": "UserId"，// AI机器人的UserId<br>"RoundId": "RoundId"，// 当前对话的RoundId<br>"Text": "Text" // 在本轮对话中，AI机器人已经说的文本<br>} |

```
{
    "EventGroupId":  9,
    "EventType":     905,
    "CallbackTs":    1687770730166,
    "EventInfo": {
        "EventMsTs": 1622186275757,
        "TaskId": "xx",
        "RoomId": "1234",
        "RoomIdType": 0,
        "Payload": {
            "UserId": "UserId",
            "RoundId": "RoundId",
            "Text": "Text"
        }
    }
}
```

## 事件类型为（906）时事件信息的定义：

| 字段名 | 类型 | 含义 |
| --- | --- | --- |
| EventMsTs | String | 事件发生的 Unix 时间戳，单位为毫秒 |
| TaskId | String | AI 任务 ID |
| RoomId | String | TRTC 的房间 ID |
| RoomIdType | Integer | • 0：表示数字房间号<br>• 1：表示字符串房间号 |
| Payload | JSON Object | 为 JSON 对象<br>{<br>"Metric": "llm_network_latency"， // 指标名<br>"Value": 218，// 指标<br>"Tag": {<br>"RoundId": "070c4908-105"，// 对话轮次id<br>}<br>}<br><br>调用指标名称如下：<br>• asr_latency<br>• llm_network_latency<br>• llm_first_token<br>• tts_network_latency<br>• tts_first_frame_latency<br>• tts_discontinuity<br>• interruption |

```
{
    "EventGroupId":  9,
    "EventType":     906,
    "CallbackTs":    1687770730166,
    "EventInfo": {
        "EventMsTs": 1622186275757,
        "TaskId": "xx",
        "RoomId": "1234",
        "RoomIdType": 0,
        "Payload": {
            "Metric": "llm_first_token",
            "Value": 218,
            "Tag": {
```

```
                "RoundId": "070c4908-1057-4ced-a949-356bf11848bc"
            }
        }
    }
}
```

## 事件类型为（908）时事件信息的定义：

| 字段名 | 类型 | 含义 |
| --- | --- | --- |
| EventMsTs | String | 事件发生的 Unix 时间戳，单位为毫秒 |
| TaskId | String | AI 任务 ID |
| RoomId | String | TRTC 的房间 ID |
| RoomIdType | Integer | • 0：表示数字房间号<br>• 1：表示字符串房间号 |
| Payload | JSON Object | 为 JSON 对象<br>{<br>"Metric": "llm_error"， // 指标名<br>"Tag": {<br>"RoundId": "070c4908-1057-4ced-a949-356bf11848bc",<br>"Code": 0, // 服务错误码<br>"Message": "" // 错误信息的详细描述<br>}<br>}<br><br>错误指标名称如下：<br>• asr_error<br>• llm_error<br>• tts_error |

```
{
    "EventGroupId":  9,
    "EventType":     908,
    "CallbackTs":    1687770730166,
    "EventInfo": {
        "EventMsTs": 1622186275757,
        "TaskId": "xx",
```

```
        "RoomId": "1234",
        "RoomIdType": 0,
        "Payload": {
            "Metric": "llm_error",
            "Tag":  {
                "RoundId": "070c4908-1057-4ced-a949-356bf11848bc",
                "Code": 0,
                "Message": ""
            }
        }
    }
}
```

## 事件类型为（909）时事件信息的定义：

| 字段名 | 类型 | 含义 |
| --- | --- | --- |
| EventMsTs | String | 事件发生的 Unix 时间戳，单位为毫秒 |
| TaskId | String | AI 任务 ID |
| RoomId | String | TRTC 的房间 ID |
| RoomIdType | Integer | • 0：表示数字房间号<br>• 1：表示字符串房间号 |
| Payload | JSON Object | 为 JSON 对象<br>{<br>"Status": "session_ready" // 表示音视频通道已建立，可以进行对话。<br>} |

```
{
    "EventGroupId":  9,
    "EventType":     909,
    "CallbackTs":    1687770730166,
    "EventInfo": {
        "EventMsTs": 1622186275757,
        "TaskId": "xx",
        "RoomId": "1234",
        "RoomIdType": 0,
```

```
        "Payload": {
            "Status": "session_ready"
        }
    }
}
```

## 事件类型为（910）时事件信息的定义：

| 字段名 | 类型 | 含义 |
| --- | --- | --- |
| EventMsTs | String | 事件发生的 Unix 时间戳，单位为毫秒 |
| TaskId | String | AI 任务 ID |
| RoomId | String | TRTC 的房间 ID |
| RoomIdType | Integer | • 0：表示数字房间号<br>• 1：表示字符串房间号 |
| Payload | JSON Object | 为 JSON 对象<br>{<br>"RoundId": "RoundId"，// 本轮对话的RoundId<br>} |

```
{
    "EventGroupId":  9,
    "EventType":     910,
    "CallbackTs":    1687770730166,
    "EventInfo": {
        "EventMsTs": 1622186275757,
        "TaskId": "xx",
        "RoomId": "1234",
        "RoomIdType": 0,
        "Payload": {
            "RoundId": "RoundId"
        }
    }
}
```

## 计算签名

签名由 HMAC SHA256 加密算法计算得出，您的事件回调接收服务器收到回调消息后，通过同样的方式计算出签名，相同则说明是腾讯云的实时音视频的事件回调，没有被伪造。签名的计算如下所示：

```
//签名 Sign 计算公式中 key 为计算签名 Sign 用的加密密钥。
Sign = base64(hmacsha256(key, body))
```

⚠ **注意：**

body 为您收到回调请求的原始包体，不要做任何转化，示例如下：

```
body="{\n\t\"Ebody="
{\"EventGroupId\":7,\"EventType\":701,\"CallbackMsTs\":17019379000
12,\"EventInfo\":
{\"EventMsTs\":1701937900012,\"TaskId\":\"WMdqEeEgj2ksqnyUsuXC+qLk
VypGmwjrgh1JC6ZefVP+rvsidDnZsAw8uWgX0XRGvdSVfAMunise2kcZaefdgHvx3-
M2v6fmTjRNgg..\",\"Status\":0}}"ventGroupId\":\t1,\n\t\"EventType\
":\t103,\n\t\"CallbackTs\":\t1615554923704,\n\t\"EventInfo\":\t{\n
\t\t\"RoomId\":\t12345,\n\t\t\"EventTs\":\t1608441737,\n\t\t\"User
Id\":\t\"test\",\n\t\t\"UniqueId\":\t1615554922656,\n\t\t\"Role\":
\t20,\n\t\t\"Reason\":\t1\n\t}\n}"
```

## 签名校验示例

Java

```
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;
//# 功能：第三方回调sign校验
//# 参数：
//#   key：控制台配置的密钥key
//#   body：腾讯云回调返回的body体
//#   sign：腾讯云回调返回的签名值sign
//# 返回值：
//#   Status：OK 表示校验通过，FAIL 表示校验失败，具体原因参考Info
//#   Info：成功/失败信息

public class checkSign {
```

```
    public static String getResultSign(String key, String body) throws
Exception {
        Mac hmacSha256 = Mac.getInstance("HmacSHA256");
        SecretKeySpec secret_key = new SecretKeySpec(key.getBytes(),
"HmacSHA256");
        hmacSha256.init(secret_key);
        return
Base64.getEncoder().encodeToString(hmacSha256.doFinal(body.getBytes()));
    }
    public static void main(String[] args) throws Exception {
        String key = "123654";
        String body = "{\n" + "\t\"EventGroupId\":\t2,\n" +
"\t\"EventType\":\t204,\n" + "\t\"CallbackTs\":\t1664209748188,\n" +
"\t\"EventInfo\":\t{\n" + "\t\t\"RoomId\":\t8489,\n" +
"\t\t\"EventTs\":\t1664209748,\n" +
"\t\t\"EventMsTs\":\t1664209748180,\n" +
"\t\t\"UserId\":\t\"user_85034614\",\n" + "\t\t\"Reason\":\t0\n" +
"\t}\n" + "}";
        String Sign = "kkoFeO3Oh2ZHnjtg8tEAQhtXK16/KI05W3BQff8IvGA=";
        String resultSign = getResultSign(key, body);

        if (resultSign.equals(Sign)) {
            System.out.println("{'Status': 'OK', 'Info': '校验通过'}");
        } else {
            System.out.println("{'Status': 'FAIL', 'Info': '校验失败'}");
        }
    }
}
```

**Python**

```
# -*- coding: utf8 -*-
import hmac
import base64
from hashlib import sha256

# 功能：第三方回调sign校验
# 参数：
#   key：控制台配置的密钥key
```

```
#    body：腾讯云回调返回的body体
#    sign：腾讯云回调返回的签名值sign
# 返回值：
#    Status：OK 表示校验通过，FAIL 表示校验失败，具体原因参考Info
#    Info：成功/失败信息

def checkSign(key, body, sign):
    temp_dict = {}
    computSign = base64.b64encode(hmac.new(key.encode('utf-8'),
body.encode('utf-8'), digestmod=sha256).digest()).decode('utf-8')
    print(computSign)
    if computSign == sign:
        temp_dict['Status'] = 'OK'
        temp_dict['Info'] = '校验通过'
        return temp_dict
    else:
        temp_dict['Status'] = 'FAIL'
        temp_dict['Info'] = '校验失败'
        return temp_dict

if __name__ == '__main__':
    key = '123654'
    body = "{\n" + "\t\"EventGroupId\":\t2,\n" +
"\t\"EventType\":\t204,\n" + "\t\"CallbackTs\":\t1664209748188,\n" +
"\t\"EventInfo\":\t{\n" + "\t\t\"RoomId\":\t8489,\n" +
"\t\t\"EventTs\":\t1664209748,\n" +
"\t\t\"EventMsTs\":\t1664209748180,\n" +
"\t\t\"UserId\":\t\"user_85034614\",\n" + "\t\t\"Reason\":\t0\n" +
"\t}\n" + "}"
    sign = 'kkoFeO3Oh2ZHnjtg8tEAQhtXK16/KI05W3BQff8IvGA='
    result = checkSign(key, body, sign)
    print(result)
```

## PHP

```
<?php

class TlsEventSig {
```

```
    private $key = false;
    private $body = false;

    public function __construct( $key, $body ) {
        $this->key = $key;
        $this->body = $body;
    }

    private function __hmacsha256() {
        $hash = hash_hmac( 'sha256', $this->body, $this->key, true );
        return base64_encode( $hash);
    }

    public function genEventSig() {
        return $this->__hmacsha256();
    }
}

$key="789";
$data="
{\n\t\"EventGroupId\":\t1,\n\t\"EventType\":\t101,\n\t\"CallbackTs\":\t1
608086882372,\n\t\"EventInfo\":\t{\n\t\t\"RoomId\":\t20222,\n\t\t\"Event
Ts\":\t1608086882,\n\t\t\"UserId\":\t\"222222_phone\"\n\t}\n}";

$api = new  TlsEventSig($key, $data);
echo $api->genEventSig();
```

## Golang

```
package main
import "fmt"
import (
    "crypto/hmac"
    "crypto/sha256"
    "encoding/base64"
)

func main () {
```

```
    var data = "
{\n\t\"EventGroupId\":\t1,\n\t\"EventType\":\t101,\n\t\"CallbackTs\":\t1
608086882372,\n\t\"EventInfo\":\t{\n\t\t\"RoomId\":\t20222,\n\t\t\"Event
Ts\":\t1608086882,\n\t\t\"UserId\":\t\"222222_phone\"\n\t}\n}"
    var key = "789"

    //JSRUN引擎2.0，支持多达30种语言在线运行，全仿真在线交互输入输出。
    fmt.Println(hmacsha256(data,key))
}

func hmacsha256(data string, key string) string {
    h := hmac.New(sha256.New, []byte(key))
    h.Write([]byte(data))
    return base64.StdEncoding.EncodeToString(h.Sum(nil))
}
```

# 自定义 TTS 协议

最近更新时间：2024-11-08 18:05:32

## 调用方式

POST：http://xxxxxxxxxxxx/api/v1/tts/stream

## 示例

## HTTP 请求 Header

```
Content-Type: application/json;charset=UTF-8
X-Task-Id: task_id_value
X-Rquest-Id: request_id
X-Sdk-App-Id: SdkAppId
X-User-Id: UserId
X-Room-Id: RoomId
X-Room-Id-Type: "0"
Authorization: Bearer "API-KEY"
```

## 请求体示例

```
{
  "Text": "你好，世界！这是流式 TTS API 的测试。",
  "Format": "wav",
  "SampleRate": 16000,
  "Channel": 1
}
```

# HTTP 请求 Header

| 字段 | 描述 |
| --- | --- |
| Content-Type | application/json |
| charset | UTF-8 |
| X-Task-Id | 对话任务的 Id |
| X-Request-Id | 此次请求的 Id，重试会携带相同的 RequestId |

| | |
|---|---|
| X-Sdk-App-Id | SDK的AppId |
| X-User-Id | 用户Id |
| X-Room-Id | Romm Id |
| X-Room-Id-Type | Romm Id的类型，0 - 数字房间号，1 - 表示字符串房间号 |
| Authorization | 鉴权，格式为Bearer "API-KEY" |

## 请求体

| 参数名称 | 必填 | 类型 | 描述 |
|---|---|---|---|
| Text | 是 | String | 语音文本 |
| Format | 否 | String | 期望输出的音频格式，如mp3， ogg_opus，pcm，wav，默认为wav，一期只支持pcm和wav |
| SampleRate | 否 | Integer | 音频采样率，默认为16000(16k)，推荐值为16000 |
| Channel | 否 | Integer | 音频通道，取值：1 或 2 默认 1 |

## 响应

需要根据 Content-Type 的值来确定合成是否成功。

- **如果成功，正常返回为二进制语音，不同的音频格式的 Content-Type 如下，需要在 HTTP 响应的 Header 设置 Transfer-Encoding: chunked。**

> ⚠ **注意：**
> Response Header 中必须包含 Content-Type 和 Transfer-Encoding，我们服务会强制检查。
> 比如：
> {
> **Transfer-Encoding: chunked;**
> **Content-Type: audio/L16**
> }

| 音频格式 | Content-Type |
|---|---|
| mp3 | audio/mpeg |

| | |
|---|---|
| ogg_opus | audio/ogg |
| pcm | audio/L16 |
| wav | audio/wav |

- **如果失败**，则会返回 json 结果，具体 header 信息为：Content-type: application/json。响应为：

```
{
  "error": {
    "code": "ERROR_CODE",
    "message": "A description of the error"
  }
}
```

# 使用语义断句

最近更新时间：2025-07-03 10:06:11

## 概述

对话语义断句是一种高级断句功能，通过结合**传统声学信号**和**上下文语义分析**来实现更精准的对话断句识别，有效解决传统方法中的限制，显著提升对话体验质量。

## 实现原理

传统的 VAD (Voice Activity Detection) 方案主要依靠静音时长来进行断句判断。然而，这种方法在处理复杂对话场景时存在诸多不足。
语义断句技术在传统 VAD 的基础上，增加了上下文语义分析能力：

- 结合声学信号特征（例如停顿、语调变化）。
- 分析语句的语义完整性和上下文关联。
- 智能识别真正的句子边界（识别不完整时等待，识别到完整时尽快进行后续处理）。

这种融合方法能够更准确地判断用户发言的实际结束点，有效缓解 AI 在对话中出现的抢话行为，使交互更自然流畅。

## 启用方式

启用语义断句功能非常简单，只需在启动对话任务时进行适当配置：

```
// 在调用 StartAIConversation 接口时
AgentConfig.TurnDetectionMode = 3
```

**参数说明：TurnDetectionMode = 3：启用语义断句模式**

## 参数配置指南

## 基本设置

**TurnDetectionMode = 3**：启用语义断句模式，系统将根据语义内容智能判断用户语音的自然断点。

## 灵敏度调节（可选）

当启用语义断句模式后，您可通过以下参数调整断句的灵敏度：

```
// 在调用 StartAIConversation 接口时
AgentConfig.TurnDetection.SemanticEagerness = "low" | "medium" | "high"
| "auto"
```

版权所有：腾讯云计算（北京）有限责任公司

## 参数说明

| 参数 | 说明 |
| --- | --- |
| auto | **默认**设置，与 medium 级别相同，提供平衡的响应时机。 |
| low | 保守模式，给予用户充分表达时间，等待更长的停顿才判定为语音结束。 |
| medium | 均衡模式，在用户表达与系统响应间取得平衡。 |
| high | 积极模式，更快速地对音频进行分块处理，实现更频繁的交互响应。 |

> **说明：**
> 无论选择哪种灵敏度，系统最终都会将语音内容分段并传送给大模型进行处理和回复。

## 应用场景推荐

| 模式 | 参数 | 适用场景 |
| --- | --- | --- |
| 积极模式（高灵敏度） | high | • **客服交互**：需要对客户问题快速响应，减少等待时间。<br>• **信息查询**：用户提问简短明确，期望立即获得答案。<br>• **导航指引**：用户需要实时指导，如路线导航或操作引导。<br>• **紧急服务**：时间敏感场景，需要迅速捕捉关键信息。<br>• **销售对话**：快节奏的商业互动，需要及时跟进客户意向。 |
| 均衡模式（中等灵敏度） | medium、auto | • **日常对话**：一般性交流，既不需要特别快速也不需要特别耐心。<br>• **教学辅助**：标准教学环境中的问答互动。<br>• **产品咨询**：用户需要了解产品信息，节奏适中。<br>• **技术支持**：解决常规技术问题。<br>• **健康咨询**：患者描述症状与医疗 AI 的互动。 |
| 保守模式（低灵敏度） | low | • **深度面试**：需要给予应聘者充分思考和表达的时间。<br>• **心理咨询**：用户可能需要时间组织语言表达情绪或问题。<br>• **教育评估**：学生回答复杂问题，需要完整表达思路。<br>• **老年用户服务**：为可能语速较慢的老年人提供更有耐心的交互体验。<br>• **语言学习**：非母语使用者可能需要更多时间组织语言。<br>• **复杂决策咨询**：涉及多方面考量的咨询服务，用户需要完整阐述需求。 |

> **说明：**

选择合适的灵敏度设置将显著提升用户体验，建议根据具体应用场景和目标用户特点进行配置。

## 功能优势

相比传统断句方法，语义断句具有以下显著优势：

- 更高的断句准确度，尤其在复杂对话场景中。
- 有效减少 AI 抢话行为，使对话更加自然流畅。
- 适应不同用户的语言习惯和表达方式。
- 提升整体用户体验和对话效率。

## 适用场景

语义断句技术特别适用于以下场景：

- 需要高质量交互体验的客服系统。
- 长对话场景中的实时问答。
- 多轮对话中需要精确理解用户意图的应用。
- 用户表达节奏不规则或语言习惯特殊的场景。

版权所有：腾讯云计算（北京）有限责任公司

# 更多功能
# 大模型透传自定义消息

最近更新时间：2025-08-06 15:20:01

## 需求场景

- 大模型返回不参与 TTS 的内容。
- 大模型返回表情甚至业务信息。

## 解决方案

大模型返回内容中增加自定义字段 `metainfo`， TRTC AI 探测到 `metainfo` 后，通过 自定义消息 推送到客户端 SDK 上，完成 `metainfo` 的透传。

## 大模型侧如何发送

### 非流式请求

在大模型返回的 body 中增加 `metainfo` 字段。

```
{
  "id": "chatcmpl-123",
  "object": "chat.completion",
  "created": 1694268190,
  "model": "gpt-xxxx",
  "system_fingerprint": "fp_xxxx",
  "choices": [{
    "index": 0,
    "message": {
      "role": "assistant",
      "content": "\n\nHello there, how may I assist you today?",
      "metainfo": {}  // 这里增加自定义字段
    },
    "logprobs": null,
    "finish_reason": "stop"
  }],
  "usage": {
    "prompt_tokens": 9,
    "completion_tokens": 12,
    "total_tokens": 21,
```

```
    "completion_tokens_details": {
      "reasoning_tokens": 0
    }
  }
}
```

### 流式请求

在大模型流式返回 chat completion chunk 对象时，同时返回 `metainfo` 的 chunk。

```
{"id":"chatcmpl-
123","object":"chat.completion.chunk","created":1694268190,"model":"gpt-
xxxx", "system_fingerprint": "fp_xxxx", "choices":[{"index":0,"delta":
{"role":"assistant","content":""},"logprobs":null,"finish_reason":null}]
}

{"id":"chatcmpl-
123","object":"chat.completion.chunk","created":1694268190,"model":"gpt-
xxxx", "system_fingerprint": "fp_xxxx", "choices":[{"index":0,"delta":
{"content":"Hello"},"logprobs":null,"finish_reason":null}]}

// 增加如下自定义消息
{"id":"chatcmpl-123","type":"meta.info","created":1694268190,"metainfo":
{}}

{"id":"chatcmpl-
123","object":"chat.completion.chunk","created":1694268190,"model":"gpt-
xxxx", "system_fingerprint": "fp_xxxx", "choices":[{"index":0,"delta":
{},"logprobs":null,"finish_reason":"stop"}]}
```

## 客户端侧如何接收

TRTC AI 检测到有 `metainfo` 之后，通过 TRTC SDK 的 自定义消息 进行下发。客户端可通过 SDK 回调中的 `onRecvCustomCmdMsg` 接口接收。

```
{
  "type": 10002, // 自定义消息
  "sender": "user_a", // 发送者userid，这里是机器人的id
  "receiver": [], // 接受者userid列表，该消息实际是在房间内广播，
  "roundid": "xxxxxx",
```

```
    "payload": {}  // metainfo
}
```

版权所有：腾讯云计算（北京）有限责任公司

# AI 对话声纹能力接入

最近更新时间：2025-06-30 21:57:42

## 概述

TRTC AI 对话声纹能力是腾讯实时音视频（TRTC）提供的一项智能语音识别技术，通过声纹识别技术实现对不同说话人的精准识别和区分。该功能可以在 AI 对话过程中自动识别说话人身份，为用户提供更加个性化和智能化的交互体验。
声纹识别基于每个人独特的声音特征，例如音调、语速、音色等生物特征，为 AI 对话系统提供说话人身份验证和识别能力，实现更精准的语音交互控制。

## 功能优势

### 高精度识别

- 基于深度学习算法，声纹识别准确率高。
- 支持在嘈杂环境下的稳定识别。
- 抗干扰能力强，有效过滤背景噪音。

### 个性化体验

- 根据不同说话人提供定制化回答。
- 支持多用户场景下的个性化服务。

## 应用场景

### 场景一：个性化AI助手

在AI对话过程中自动识别说话人，针对不同的说话人提供个性化的回答。

#### 适用情况：

- 智能客服系统：为VIP客户提供专属服务
- 智能家居：家庭成员个性化语音控制

### 场景二：声纹过滤与专注对话

在 AI 对话过程中，只回应特定说话人的内容，起到过滤背景人声或者远场人声的效果。

#### 适用情况：

- 会议场景：只响应主讲人的语音指令
- 嘈杂环境：过滤背景噪音，专注目标用户
- 多人对话：避免误触发，提高交互准确性

版权所有：腾讯云计算（北京）有限责任公司

- 安全场景：只接受授权人员的语音指令

# 使用方式

## 步骤1：声纹注册

1. 在开始 AI 对话前，需要使用 Register 声纹 API 进行声纹注册。
2. 调用声纹注册 API（RegisterVoicePrint），录制8到18秒的有效音频，音频中应保证人声清晰、无明显噪音或空白片段。
3. 注册成功后，系统返回唯一的 VoicePrintId。

详细接口信息查看文档 声纹注册 。

## 步骤2：配置AI对话

1. 开启 AI 对话时，在 AgentConfig.VoicePrint 结构体中传入对应信息。**在对话中会过滤非注册声纹的声音**。
2. 在 AgentConfig.VoicePrint 结构体中进行配置：

| 参数 | 说明 |
| --- | --- |
| Mode | • 默认值为0，表示禁用声纹功能。<br>• 设置为1表示启用固定声纹功能，此时需填写对应的 VoicePrintId。 |
| IdList | 当 VoicePrint Mode 为1时填写该字段，目前仅支持填写单个 VoicePrintId。 |

详细接口信息查看文档 声纹配置 。

## 步骤3：获取声纹结果

在使用注册好的声纹开启对话任务时，系统会在字幕中返回声纹识别结果。具体说明如下：

### 结果返回时机

- 声纹结果仅在字幕的 end 为 true 时返回。
- 声纹匹配不通过的字幕默认不会发送给 LLM 处理。
- 如需将不匹配的字幕发送给 LLM，可通过端上发送自定义消息实现。

具体消息格式如下：

```
{
  "type": 10000,
  "sender": "user_a",
  "receiver": [],
  "payload": {
```

```
    "text": "你好，很高兴认识你。",
    "start_time": "00:00:01",
    "end_time": "00:00:03",
    "roundid": "conversation_123456",
    "end": true,
    "voiceprint_result": 2,
    "voiceprint_audio_name": "your_audio_name"
  }
}
```

| 字段名 | 说明 |
| --- | --- |
| voiceprint_result | 声纹识别结果状态码，状态码及对应说明 见下表 。 |
| voiceprint_audio_name | 注册声纹时设置的 AudioName。 |

### 声纹结果状态码

| voiceprint_result 状态码 | 说明 |
| --- | --- |
| 1 | 声纹暂未锁定。 |
| 2 | 声纹匹配不通过。 |
| 3 | 声纹匹配通过。 |
| 4 | 声纹识别出现错误。 |

## 使用注意事项

### 音频质量要求

- **时长要求**：注册的音频需要8到18秒有效音频时长。
- **音质要求**：保证人声清晰，没有明显的噪音。
- **内容要求**：没有留白内容，确保全程有效人声。

### 注册建议

- 选择在安静环境下录制注册音频。
- 使用正常语速和音调进行录制。
- 避免背景音乐或其他人声干扰。
- 建议录制多段音频以提高识别准确性。

版权所有：腾讯云计算（北京）有限责任公司

## 使用限制

- 目前 IdList 仅支持填写一个声纹 ID。
- 建议定期更新声纹模型以保持最佳识别效果。

## 相关链接

- 注册声纹信息
- 更新声纹信息
- 删除声纹信息
- 查询声纹信息
- 开始 AI 对话任务

## 技术支持

声纹能力在试用阶段，如果您感兴趣可以加入群聊申请测试。



# IM 信令方案

最近更新时间：2025-06-16 17:54:12

> **说明：**
> 目前小程序端暂不支持接收和发送自定义消息，如果您想在小程序端实现接收字幕或发送消息等功能，需要使用 IM 提供的即时通信来实现，可通过商务或 提交工单 联系我们开通 IM 信令通道。开通后，即可参考以下文档集成。

## IM SDK 集成

iOS

### 集成 IM SDK

建议选择使用 CocoaPods 自动加载的方式集成 IM SDK。

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

```
sudo gem install cocoapods
```

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

```
pod init
```

3. 编辑 Podfile 文件。请您按照如下方式设置 Podfile 文件：

```
platform :ios, '8.0'
source 'https://github.com/CocoaPods/Specs.git'

target 'App' do
    # 集成完整版本的 IM SDK，版本号要大于 '8.1.6129'
    pod 'TXIMSDK_Plus_iOS','8.1.6129'
    # 或者集成裁剪体积后的 IM SDK（仅包含 AI 信令相关能力），版本号要大于 '8.2.6361'
    pod 'TXIMSDK_Plus_SignalingSDK','8.2.6361'
end
```

4. 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 IM SDK：

```
pod install
```

或使用以下命令更新本地库版本：

```
pod update
```

> ⚠ **注意：**
> 若您进行上述操作后仍遇到问题，请参见 Xcode 集成常见问题 文档。

## 引用 IM SDK

项目代码中使用 SDK 有两种方式：

### 方式一

在 `Xcode > Build Setting > Header Search Paths` 设置 SDK 头文件的路径，然后在项目需要使用 SDK API 的文件里，引入具体的头文件。

```
#import "ImSDK_Plus.h"
```

### 方式二

在项目需要使用 SDK API 的文件里，引入具体的头文件。

```
#import <ImSDK_Plus/ImSDK_Plus.h>
```

Android

## 集成 SDK（aar）

建议选择使用 Gradle 自动加载的方式集成 IM SDK。

1. 添加 SDK 依赖。

1.1 找到 app 的 build.gradle，在 repositories 中添加 mavenCentral() 的依赖。

```
repositories {
    google()
```

```
    // 增加 mavenCentral 仓库
    mavenCentral()
}
```

1.2 然后在 dependencies 中添加 IM SDK 的依赖。

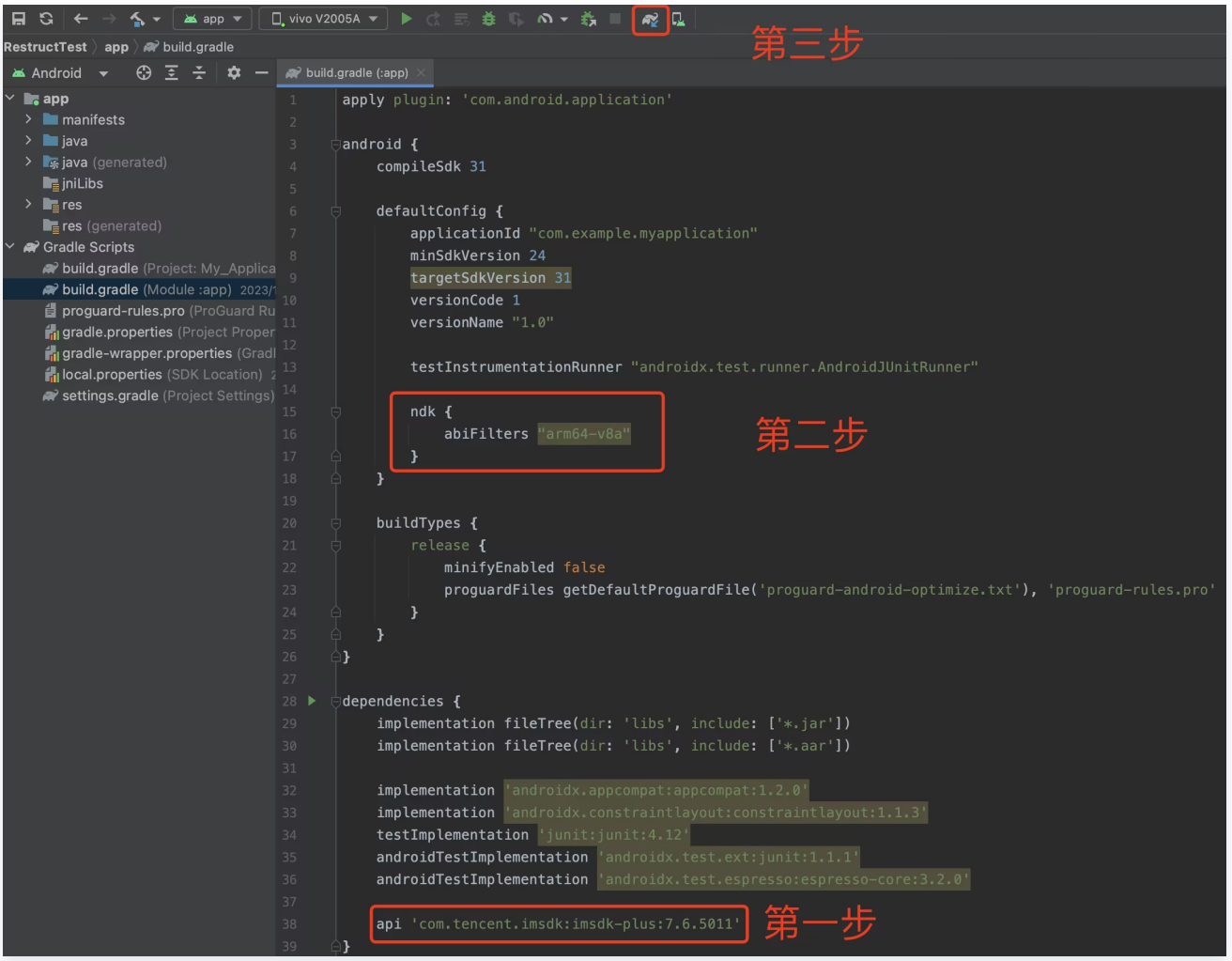
```
dependencies {
    // 集成完整版本的 IM SDK，版本号要大于 '8.1.6129'
    api 'com.tencent.imsdk:imsdk-plus:8.1.6129'

    // 或者集成裁剪体积后的 IM SDK（仅包含 AI 信令相关能力），版本号要大于 '8.2.6361'
    api 'com.tencent.imsdk:signalingsdk:8.2.6361'
}
```

2. 指定 App 使用架构。在 defaultConfig 中，指定 App 使用的 CPU 架构（从 IM SDK 4.3.118 版本开始支持 armeabi-v7a，arm64-v8a，x86，x86_64）。

```
defaultConfig {
    ndk {
      abiFilters "arm64-v8a"
    }
 }
```

3. 同步 SDK。请保证您的网络已连接 maven，单击 Sync 按钮，SDK 就会自动下载集成到工程里。




## 配置 App 权限

在 AndroidManifest.xml 中配置 App 的权限，IM SDK 需要以下权限：

```
    <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" />
```

## 设置混淆规则

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

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

Web & 小程序

## 集成 SDK

建议通过 npm 方式将 IM SDK 集成到您的 Web、小程序中。

```
// 版本号v3.4.5或更高
npm install @tencentcloud/chat
```

> **说明：**
> 若同步依赖过程中出现问题，请切换 npm 源后再次重试。

```
npm config set registry http://r.cnpmjs.org/
```

## 引入模块

```
import TencentCloudChat from '@tencentcloud/chat';
```

# 初始化 SDK

iOS

1. 调用初始化接口。

```
// 1. 从即时通信 IM 控制台获取应用 SDKAppID。
// 2. 初始化 config 对象
V2TIMSDKConfig *config = [[V2TIMSDKConfig alloc] init];
// 3. 指定 log 输出级别。
config.logLevel = V2TIM_LOG_INFO;
// 4. 添加 V2TIMSDKListener 的事件监听器，self 是
id<V2TIMSDKListener> 的实现类，如果您不需要监听 IM SDK 的事件，这个步骤可
以忽略。
[[V2TIMManager sharedInstance] addIMSDKListener:self];
// 5. 初始化 IM SDK，调用这个接口后，可以立即调用登录接口。
```

```
[[V2TIMManager sharedInstance] initSDK:sdkAppID config:config];
```

2. 登录。

```
NSString *userID = @"your user id";
NSString *userSig = @"userSig from your server";
[[V2TIMManager sharedInstance] login:userID userSig:userSig
succ:^{
    NSLog(@"success");
} fail:^(int code, NSString *desc) {
    // 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 IM SDK 登录进入死
循环。
    NSLog(@"failure, code:%d, desc:%@", code, desc);
}];
```

## Android

1. 调用初始化接口。

```
// 1. 从即时通信 IM 控制台获取应用 SDKAppID。
// 2. 初始化 config 对象。
V2TIMSDKConfig config = new V2TIMSDKConfig();
// 3. 指定 log 输出级别。
config.setLogLevel(V2TIMSDKConfig.V2TIM_LOG_INFO);
// 4. 添加 V2TIMSDKListener 的事件监听器，sdkListener 是
V2TIMSDKListener 的实现类，如果您不需要监听 IM SDK 的事件，这个步骤可以忽
略。
V2TIMManager.getInstance().addIMSDKListener(sdkListener);
// 5. 初始化 IM SDK，调用这个接口后，可以立即调用登录接口。
V2TIMManager.getInstance().initSDK(context, sdkAppID, config);
```

2. 登录。

```
String userID = "your user id";
String userSig = "userSig from your server";
V2TIMManager.getInstance().login(userID, userSig, new
V2TIMCallback() {
    @Override
    public void onSuccess() {
        Log.i("imsdk", "success");
    }

    @Override
    public void onError(int code, String desc) {
        // 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 IM SDK 登录进
入死循环。
        Log.i("imsdk", "failure, code:" + code + ", desc:" +
desc);
    }
});
```

## Web & 小程序

**1. 调用初始化接口。**

```
import TencentCloudChat from '@tencentcloud/chat';

let options = {
  SDKAppID: 0 // 接入时需要将0替换为您的即时通信 IM 应用的 SDKAppID
};
// 创建 SDK 实例，`TencentCloudChat.create()`方法对于同一个 `SDKAppID`
只会返回同一份实例
let chat = TencentCloudChat.create(options); // SDK 实例通常用 chat
表示

chat.setLogLevel(0); // 普通级别，日志量较多，接入时建议使用
```

```
// chat.setLogLevel(1); // release 级别，SDK 输出关键信息，生产环境时建
议使用
```

2. 登录。

```
let promise = chat.login({userID: 'your userID', userSig: 'your
userSig'});
promise.then(function(imResponse) {
  console.log(imResponse.data); // 登录成功
  if (imResponse.data.repeatLogin === true) {
    // 标识账号已登录，本次登录操作为重复登录。
    console.log(imResponse.data.errorInfo);
  }
}).catch(function(imError) {
  console.warn('login error:', imError); // 登录失败的相关信息
});
```

⚠ **注意：**

- TRTC 和 IM 的 `sdkAppId` 和 `secretKey` 必须相同。
- 收消息的接收者的 IM 必须要登录成功，既是在线状态。
- 接收者的 TRTC 账号和 IM 账号必须是同一个 `userId`（即使用同一个 `userId` 进入 TRTC 房间和 IM 登录）。

## 接收服务端下行消息

通过 IM SDK 接收单聊自定义消息功能（iOS & Android / Web & 小程序），在客户端上监听回调，来接收实时字幕与 AI 状态的数据。

| type | 说明 |
| --- | --- |
| 10000 | 实时字幕、翻译下发。 |
| 10001 | AI 对话实时状态下发。 |

### 接收实时字幕

```
{
  "type": 10000, // 10000表示是下发的实时字幕
```

```
  "sender": "user_a", // 说话人的userid
  "receiver": [], // 接收者userid列表，该消息实际是在房间内广播
  "payload": {
     "text":"", // 语音识别出的文本
     "start_time":"00:00:01", // 这句话的开始时间
     "end_time":"00:00:02", // 这句话的结束时间
     "roundid": "xxxxx" // 唯一标识一轮对话
     "end": true // 如果为true，代表这是一句完整的话
  }
}
```

## 接收机器人状态

```
{
  "type": 10001, // 机器人的状态
  "sender": "user_a", // 发送者userid，这里是机器人的id
  "receiver": [], // 接受者userid列表，该消息实际是在房间内广播
  "payload": {
    "roundid": "xxx", // 唯一标识一轮对话
    "timestamp": 123
    "state": 1,       // 1 聆听中  2 思考中  3 说话中  4 被打断  5 已说完
  }
}
```

## 代码示例

### iOS

```
// 调用 addSimpleMsgListener 设置事件监听器
V2TIMManager.sharedInstance().addSimpleMsgListener(listener: self)

/// 接收单聊自定义消息
/// @param msgID 消息 ID
/// @param info 发送者信息
/// @param data 自定义消息二进制内容
func onRecvC2CCustomMessage(_ msgID: String!, sender info:
V2TIMUserInfo!, customData data: Data!) {
    do {
```

```
        if let jsonObject = try JSONSerialization.jsonObject(with: data,
options: []) as? [String: Any] {
            print("onRecvGroupCustomMessage: \(jsonObject)")
            handleMessage(jsonObject)
        } else {
            print("The data is not a dictionary.")
        }
    } catch {
        print("Error parsing JSON: \(error)")
    }
}
```

## Android

```
// 调用 addSimpleMsgListener 设置事件监听器
V2TIMManager.getInstance().addSimpleMsgListener(sdkListener);

/**
 * 接收单聊自定义消息
 * @param msgID 消息 ID
 * @param sender 发送方信息
 * @param customData 发送内容
 */
public void onRecvC2CCustomMessage(String msgID, V2TIMUserInfo sender,
byte[] customData) {
    Log.i("onRecvC2CCustomMessage", "msgID:" + msgID + ", from:" +
sender.getNickName() + ", content:" + new String(customData));
    try {
        String jsonString = new String(customData, "UTF-8");
        JSONObject jsonObject = new JSONObject(jsonString);
        System.out.println("onRecvGroupCustomMessage: " + jsonObject);
        handleMessage(jsonObject);
     } catch (UnsupportedEncodingException e) {
        System.out.println("The data is not a dictionary.");
     } catch (JSONException e) {
        System.out.println("Error parsing JSON: " + e);
     }
}
```

**Web & 小程序**

```
const onMessageReceived = (event) => {
  const messageList = event.data;
  messageList?.forEach((msg) => {
    if (msg.type === TencentCloudChat.TYPES.MSG_CUSTOM) {
      console.log('收到自定义消息', event);
      const { data } = msg.payload;
      try {
        const jsonData = JSON.parse(data);
        console.log(`receive custom msg from ${msg.from} data:
${data}`);
        if (jsonData.type === 10000) {
          console.log('字幕消息', jsonData);
          return;
        }
        if (jsonData.type === 10001) {
          console.log('机器人的状态', jsonData);
          return;
        }
      } catch (error) {
        console.error('receive custom msg', data, error);
      }
    }
  });
}

// 监听消息
chat.on(TencentCloudChat.EVENT.MESSAGE_RECEIVED, onMessageReceived);
```

**说明：**
默认通过单聊自定义消息来接收实时字幕与 AI 状态的数据。如果单聊无法满足需求，需要开通群聊自定义消息通道，可通过商务或 提交工单 联系我们。

## 端上发送上行信令

可以通过发送自定义信令，跳过 ASR 过程，直接跟 AI 进行文字沟通，或通过发送打断信令来进行打断。

| type | 说明 |
| --- | --- |

| | |
|---|---|
| 20000 | ai_conversation_chat: 发送 AI 对话文本。 |
| 20001 | ai_conversation_interrupt: 手动打断。 |

**发送上行信令，跳过 ASR 过程，直接跟 AI 进行文字沟通**

```
{
  "type": 20000,
  "sender": "user_a", // 发送者userid，服务端会 check 该 userid 是否有效
  "receiver": ["user_bot"], // 接受者 userid 列表，只需要填写机器人 userid，服务端会 check 该 userid 是否有效
  "payload": {
    "id": "uuid", // 消息 id，可以使用 uuid，排查问题使用
    "message": "xxx", // 消息内容
    "timestamp": 123, // 时间戳，排查问题使用
    "taskid": "v2_20240920_xxxxx",
  }
}
```

**发送打断信令来进行打断**

```
{
  "type": 20001,
  "sender": "userid", // 发送者userid，服务端会 check 该 userid 是否有效
  "receiver": ["user_bot"], // 接受者 userid 列表，只需要填写机器人 userid
  "payload": {
    "id": "uuid", // 消息 id，可以使用 uuid，排查问题使用
    "timestamp": 123 // 时间戳，排查问题使用
    "taskid": "v2_20240920_xxxxx",
  }
}
```

**代码示例**

iOS

```
@IBAction func interruptAi(_ sender: UIButton) {
    let timestamp = Int(Date().timeIntervalSince1970 * 1000)
    let payload = [
```

```
        "id": userId + "_\(roomId)" + "_\(timestamp)", // 消息id，可以使用
uuid，排查问题使用
        "timestamp": timestamp, // 时间戳，排查问题使用
        "taskid": aiTaskId,
    ] as [String : Any]
    let content = [
        "type": 20001,
        "sender": userId,
        "receiver": [botId],
        "payload": payload
    ] as [String : Any]
    let contentData = try! JSONSerialization.data(withJSONObject:
content, options: [])
    let contentString = String(data: contentData, encoding: .utf8)!
    let dataDict = [
        "service_command": "trtc_ai_service.SendCustomCmdMsg",
        "request_content": contentString
    ] as [String : Any]
    do {
        let jsonData = try JSONSerialization.data(withJSONObject:
dataDict, options: [])

V2TIMManager.sharedInstance().callExperimentalAPI("sendTRTCCustomData",
param: jsonData as NSObject) { _ in
            print("sendTRTCCustomData success")
        } fail: { code, desc in
            print("sendTRTCCustomData error, \(code), \(desc ??
"null")")
        }
    } catch {
        print("Error serializing dictionary to JSON: \(error)")
    }
}
```

## Android

```
public void interruptAi() {
    long timestamp = System.currentTimeMillis();
    Map<String, Object> payload = new HashMap<>();
```

```
    payload.put("id", userId + "_" + roomId + "_" + timestamp); // 消息
id，可以使用uuid，排查问题使用
    payload.put("timestamp", timestamp); // 时间戳，排查问题使用
    payload.put("taskid", aiTaskId);

    Map<String, Object> content = new HashMap<>();
    content.put("type", 20001);
    content.put("sender", userId);
    content.put("receiver", Collections.singletonList(botId));
    content.put("payload", payload);

    String contentString = new JSONObject(content).toString();

    Map<String, Object> dataDict = new HashMap<>();
    dataDict.put("service_command", "trtc_ai_service.SendCustomCmdMsg");
    dataDict.put("request_content", contentString);

    try {
        byte[] jsonData = new
JSONObject(dataDict).toString().getBytes("UTF-8");

V2TIMManager.getInstance().callExperimentalAPI("sendTRTCCustomData",
jsonData, new V2TIMValueCallback() {
            @Override
            public void onSuccess(Object o) {
                System.out.println("sendTRTCCustomData success");
            }
            @Override
            public void onError(int code, String desc) {
                System.out.println("sendTRTCCustomData error, " + code +
", " + (desc != null ? desc : "null"));
            }
        });
    } catch (UnsupportedEncodingException e) {
        System.out.println("Error serializing dictionary to JSON: " +
e);
    }
}
```

**Web & 小程序**

```
// 发送打断信令
chat.callExperimentalAPI('sendTRTCCustomData', {
serviceCommand: 'trtc_ai_service.SendCustomCmdMsg',
data: {
  type: 20001,
  sender: "user_a", // 发送者userid， 服务端会check该userid是否有效
  receiver: ["user_bot"], // 接受者userid列表，只需要填写机器人userid
  payload: {
    id: "uuid", // 消息id，可以使用uuid，排查问题使用
    timestamp: 123, // 时间戳，排查问题使用
    taskid: "任务的taskid",
  }
}
});
```

⚠ **注意：**

`type`、`sender`、`receiver` 以及 `payload` 中的 `taskid`、`id`、`timestamp` 是必填字段。

# 行业实践教程
# 情感陪伴

最近更新时间：2024-09-02 17:35:11

## 场景描述

AI 情感陪伴产品迎来井喷式创新，并迅速走向多样化，包括角色扮演、情感陪聊、心理疗愈等多种类型。现有的 AI 语聊场景主要是基于 IM 场景的离线文字聊天或者语音聊天，随着GPT-4o的发布，把多模态大模型的应用场景提升到了实时的语音或者视频交互，用户得以沉浸在一个更为真实、互动的虚拟世界，享受前所未有的娱乐体验。
结合第三方的大模型和 TTS 等，腾讯云实时音视频（TRTC）让边消费边创作、剧情无限延伸且充满无限可能、实时互动与实时变化的情感陪伴体验成为现实，开发者可以轻松创建一个媲美GPT-4o所演示的AI实时交互体验。用户不仅能感受到高度个性化的陪伴，还能在互动中激发创作灵感，共同探索 AI 陪伴 + 实时互动中的无限可能。
腾讯云实时音视频（TRTC）技术的优势尤为显著。TRTC 提供了低延迟、高质量的语音和视频通信，确保了用户在与虚拟角色互动时的流畅性和真实感。此外，TRTC 的高并发处理能力使得大量用户能够同时在线，享受无缝的情感陪伴体验。

## 场景分类

| 场景分类 | 描述 | 核心需求 |
| --- | --- | --- |
| 虚拟伴侣 | 用户可以通过定制“虚拟男友/女友”的形象、性格，使其成为一个更贴心的交流对象。这些AI恋人能够理解用户的情绪，提供安慰和支持，帮助用户建立和维持社交联系 | • **个性化定制：**用户可以根据自己的喜好和需求定制 AI 恋人的性格、外貌、兴趣等，使得 AI 恋人更加贴近用户的期望<br>• **实时陪伴：**AI 恋人可以全天候在线，为用户提供即时的关心和支持，无论用户在何时何地遇到困难或需要倾诉<br>• **真实的交互体验：**理解语音中的情绪、在用户停顿时及时接过对话，音色按需选择 |
| 角色扮演 | 通过故事和剧情构建一个虚拟世界，用户可以选择不同的虚拟角色，在特定的剧情下与其进行文字或语音对话，本质是对 IP 内容的消费 | • **沉浸式互动：**用户的核心诉求和爽点，在于和特定角色进行交互和扮演，建立与该角色的联结；或者通过和角色之间的剧情演绎，获得幻想和创作的快乐<br>• **音色记忆：**AI 角色需要能够记住并模仿特定人物的音色，这对于保持角色的一致性和真实感非常重要 |
| 偶像/网红 AI 分身 | AI 分身通过高度还原偶像/网红的音色和语调，粉丝可以通过付费来获得与AI分身的专属互动，如情感陪伴、日常聊天等 | • **专属音色和语料数据：**建立专属的语音和人格引擎，让用户感觉就像是沉浸式地与偶像本人在面对面交谈 |

| | | |
|---|---|---|
| | | • **多种互动方式变现：**通过 AI 分身，除了付费语音聊天外，网红可以开展多种形式的粉丝经济变现，如虚拟礼物、付费问答、专属活动等 |

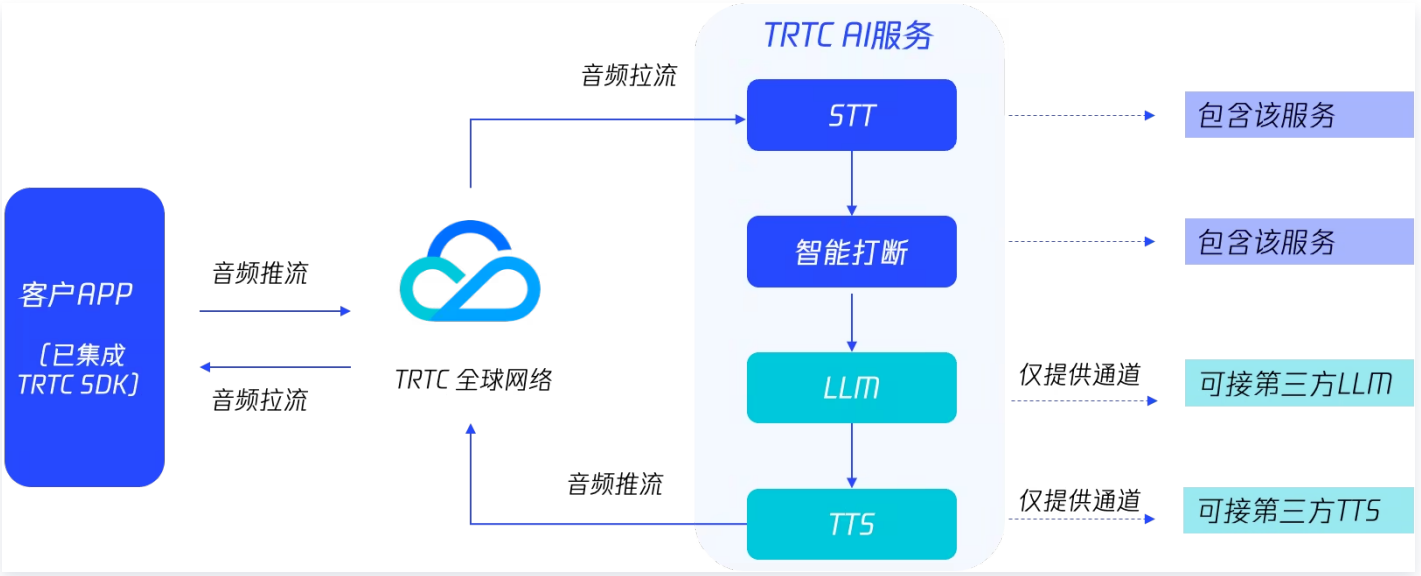
## 场景需求说明

| 角色 | 核心场景 | 场景详情 | 核心需求 |
|---|---|---|---|
| 用户 | 和 AI 虚拟角色对话 | 通过语音和 AI 虚拟角色实时对话 | • 超低延迟：期望与 AI 的交流能够毫无延迟，实现如同面对面交谈般的自然流畅体验，而非对讲机式的断断续续。<br>• 多音色：希望系统能够模拟或处理多种不同的声音，从而享受到更为丰富多彩的语音体验<br>• STT 准确：需要系统能够准确识别其语音，并将其转换成文本，以确保信息传递的准确性与高效性<br>• 随时打断：在任何时候都能够轻松地中断系统的输出或操作，以便更好地掌控对话节奏，提升交互效率<br>• 情感需求：AI 能够理解和响应用户的情感状态，能够提供更加人性化和富有同理心的交互体验 |
| AI Bot | 和用户进行对话 | 通过语音和用户实时对话 | • 理解并响应用户的情感和需求<br>• 提供连贯、自然且富有同理心的对话<br>• 维护用户的隐私和安全<br>• 不断学习和适应用户的行为和偏好，以提供个性化的服务<br>• 在技术上保持先进性，以提供流畅和高质量的交互体验 |
| 开发者 | 开发并上线 AI 陪伴 App | 搭建 AI 陪伴 App，快速上线，保障用户体验 | • 确保用户在进行与 AI 的语音对话时能够体验到超低延迟的实时互动，保障对话的流畅性和自然性<br>• 支持微信小程序、Android、iOS、Web 等多平台，且能互通，降低多平台独立维护的成本<br>• 使用一站式服务，快速上线和迭代 AI 陪伴类产品，缩短产品上市时间<br>• 确保用户数据的安全性和隐私保护，遵守相关法律法规，建立用户信任 |

版权所有：腾讯云计算（北京）有限责任公司

| | | | |
|---|---|---|---|
| | | | • 扩展性灵活，可对接多家 LLM，为之后的 RAG 和 Fine-tune 提供基础 |

## 技术方案




## 核心功能

| 功能 | AI情感陪聊场景应用 |
|---|---|
| 实时音频互动 | 用户可以通过类似“打电话”的方式，与 AI 虚拟角色进行一对一的语音交流。采用流式传输，这种实时互动不仅能够及时缓解用户的孤独感，还能提供即时的反馈和支持 |
| STT | 用户的语音会被精确识别，并实时转换成文字，发送给 LLM。LLM 会根据用户的语音内容和情感状态，提供相应的安慰和建议 |
| LLM | 支持接入第三方 LLM 或自有大模型，可结合 RAG（检索增强生成）/客户知识库，提供更加智能化和个性化的回复 |
| 智能打断 | 用户可以在任何时候打断 AI 的发言，表达自己的想法和感受。这种互动方式不仅增强了对话的自然性，还能让用户感受到更多的控制权和参与感。提供智能打断和手动打断两个模式，方便用户选择 |
| TTS | 仅提供通道，支持接入第三方 TTS，AI 虚拟角色可以以多种音色与用户交流，提供更加亲切和个性化的体验 |
| AI 降噪 | 通过先进的AI降噪技术，确保对话在嘈杂环境中也能保持清晰，减少因噪声引起的语音中断或模糊，提升用户的沟通体验 |
| 弱网卡顿优化 | 即使在网络条件不佳的情况下，AI 实时对话也能保持流畅，确保用户在任何环境下都能获得稳定的情感支持 |

版权所有：腾讯云计算（北京）有限责任公司

| | |
|---|---|
| 多平台互通 | 用户可以在微信小程序、iOS、Android、Web 等多个平台上无缝运行应用，随时随地发起和参与对话，极大提高了 AI 陪伴的便捷性和可达性 |

## 扩展功能

1. **在情感陪伴应用场景中添加文本、语音、图片的多模态交互**

   多模态交互能够提供更丰富、更自然的人机交互体验。通过结合文本、图像、音频等多种信息模态，AI 情感陪伴能够更好地理解用户的意图、情感和需求，从而提供更加个性化和适应性的响应。我们推荐用**即时通信 IM** 来实现该场景拓展，详情参见 即时通信 IM 产品介绍 。

2. **在情感陪伴应用中添加语聊房、多人群聊等玩法**

   越来越多的 AI 陪伴场景里面也集成了社交的功能，与朋友或其他用户一起讨论和分享，甚至分享和AI陪伴聊天的记忆。社交功能的集成使得AI陪伴场景更加生动有趣，增强了用户的使用黏性和满意度。我们推荐使用 语音聊天室 来实现该场景的拓展。

版权所有：腾讯云计算（北京）有限责任公司

# 智能客服

最近更新时间：2024-09-03 18:25:12

## 场景描述

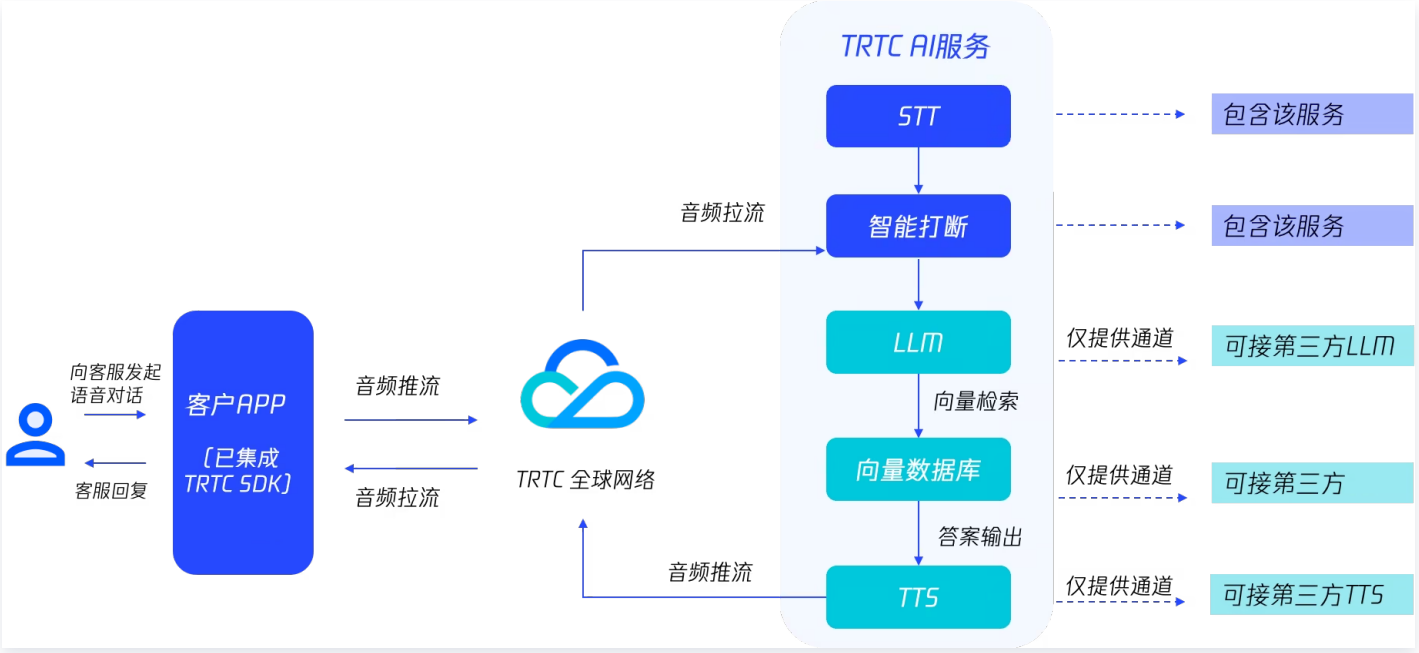
智能语音客服是一种利用人工智能和语音识别技术来实现自动化交互和问题解决的客户服务系统。在 AI 大模型出现之前，智能客服主要利用自然语言处理和机器学习算法来理解客户的意图，依赖于预设的规则和知识库来进行问题解答。随着 LLM 的发展，越来越多的智能客服接入了大模型的能力，LLM 技术使智能语音客服能够更好地理解对话的上下文，从而实现连贯的对话交流。
TRTC 技术的引入，为智能语音客服带来了实时通信的能力。这意味着智能客服可以更加迅速地响应客户的需求，提供即时的反馈和解决方案。同时，TRTC 还支持多人通话、屏幕共享等功能，进一步提升了客户服务的效率和质量。

## 场景需求说明

| 场景分类 | 核心场景 | 场景详情 | 核心需求 |
| --- | --- | --- | --- |
| 用户 | 和智能客服对话 | 通过语音、文字、图片等方式和智能客服对话，解决特定问题 | • 更自然的对话体验：与智能客服系统的对话更接近人与人之间的自然交流，能够实时对话<br>• 个性化服务：通过智能客服可以获得个性化的服务建议和解决方案 |
| 企业 | 接入智能客服 | 保障客户体验，让智能客服具备独立为用户解决问题的能力 | • 客户体验：企业对客服团队的核心考核指标依然是客户满意度、响应速度和接通率。通过LLM+智能客服的方案，可进一步识别客户意图，并生成自然、流畅的回答，帮助智能客服提高对话质量和流畅度 |
| AI客服 | 和用户对话 | 及时、准确地回答客户的问题，并能帮助客户解决问题 | • 多平台互动，可以多平台接入，支持微信小程序、Android、iOS、Web 等多平台，且能互通，降低多平台独立维护的成本<br>• 多语言支持：特别是对于开展国际业务，结合大模型可以很好地对买家和卖家之间的对话语言进行转换，例如买家用英语发文，而卖家用中文回复，但双方看到的都是自己的偏好语言 |

## 技术方案

版权所有：腾讯云计算（北京）有限责任公司




## 核心功能

| 功能 | AI 智能客服场景应用 |
| --- | --- |
| 实时音频互动 | 流式传输技术可以确保语音和视频数据的连续性和稳定性，减少延迟和抖动，提供接近于真人客服通话的高质量体验。用户可以与智能客服系统进行更自然的对话，就像是在和真人客服交谈一样，这种互动体验可以显著提升用户满意度 |
| STT | 在智能客服场景中使用 STT，可以提高语音识别和情感识别的准确率，且可以利用历史对话记录和全局语境来提高语音识别的准确性和语义一致性 |
| 多语言支持 | 智能语音客服在全球范围内使用，需要适应多种语言和跨文化交流。精准的 STT 识别支持多种语言，包括英语、西班牙语、日语、韩语、中文以及23种方言和130种国际语言。 |
| LLM | LLM 技术使智能语音客服能够更好地理解对话的上下文，从而实现连贯的对话交流。LLM 可以捕捉对话中的语义和语境信息，识别用户意图，并将上一轮对话的内容与当前对话关联起来 |
| 智能打断 | 用户可以在任何时候打断 AI 的发言，表达自己的想法和感受。这种互动方式不仅增强了对话的自然性，还能让用户感受到更多的控制权和参与感 |
| TTS | 仅提供通道，支持接入第三方 TTS，通过在模型中引入个性化的训练数据或调整模型的参数，可以生成符合特定要求的语音输出。智能语音客服可以根据用户的偏好或特定场景的需求，提供不同的语音风格 |
| AI降噪 | 通过先进的 AI 降噪技术，确保对话在嘈杂环境中也能保持清晰，减少因噪声引起的语音中断或模糊，提升用户的沟通体验 |

版权所有：腾讯云计算（北京）有限责任公司

| | |
|---|---|
| 弱网卡顿优化 | 即使在网络条件不佳的情况下，AI 实时对话也能保持流畅，确保用户在任何环境下都能获得稳定的客服支持 |
| 多平台互通 | 用户可以在微信小程序、iOS、Android、Web 等多个平台上无缝运行应用，随时随地发起和参与对话，极大提高了智能客服的便捷性和可达性 |

## 扩展功能

1. **在智能客服应用场景中添加文本、语音、图片的多模态交互**

   多模态交互能够提供更丰富、更自然的人机交互体验。通过结合文本、图像、音频等多种信息模态，AI 智能客服能够更好地理解用户的意图、情感和需求，从而提供更加个性化和适应性的响应。我们推荐用**即时通信 IM** 来实现该场景拓展，详情参见 即时通信 IM 产品 介绍。

2. **和通讯中心的互联互通**

   部分客服场景需要 RTC 和通话线路的平滑切换，从 RTC 线路切换到 SIP 线路，快速搭建集电话、在线交流、音视频通话为一体的客户联络平台。我们已有相应的解决方案，详情请见 云联络中心_TCCC 介绍。

版权所有：腾讯云计算（北京）有限责任公司