物联网开发平台

智能增值服务

腾讯云

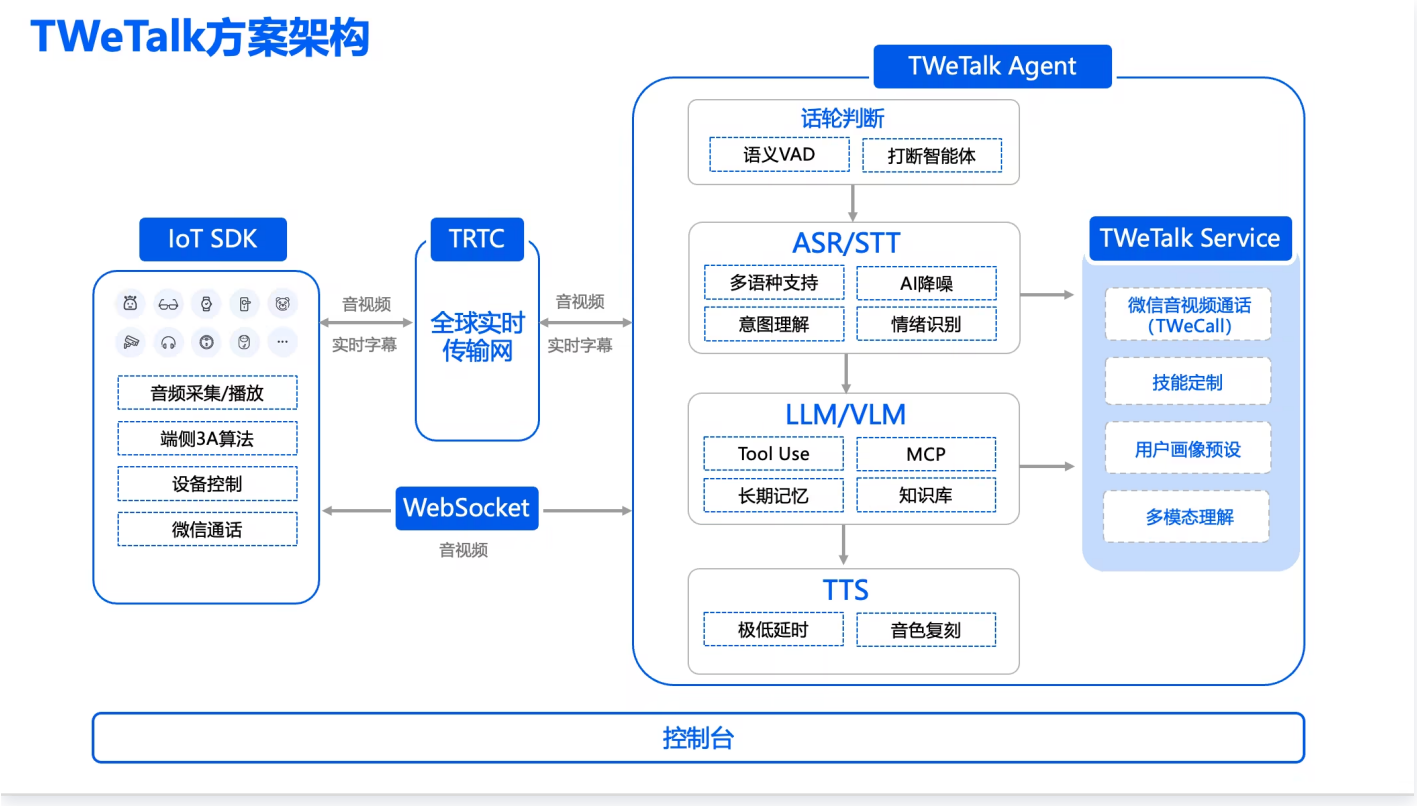
文档目录

智能增值服务
智能语音服务（TWeTalk）
功能介绍

最近更新时间：2026-03-27 11:14:52

功能概述

TWeTalk 是专为智能硬件打造的 AI 对话方案，聚焦多模态智能体与实时音视频通信两大核心。云端支持语音对话及多模态视觉推理交互，可闲聊、查询天气、问询各类信息等。语音识别融合情绪识别，预置高拟人度 TTS 音色并支持音色复刻。TWeTalk 亮点功能包括整合 Function Call 各类技能，可通过物模型配置实现硬件控制，深度整合端云协同与场景化 AI，集成腾讯系资源，可支持设备与微信音频呼叫（TWeCall）。
TWeTalk 也可以与主流嵌入式芯片及模组厂商合作，可以在端侧集成降噪、唤醒等技术。开发者可在控制台进行配置，该方案已应用于 AI 陪伴玩具、机器人、智能穿戴（手表、眼镜）、耳机同传、智能点餐、导览、AI 面试等 AIoT 场景，实现人与智能硬件的自然“对话”。



功能特性

功能类别	功能描述
AI 语音对话	语音智能体（Voice Agent）：支持智能硬件设备的语音交互。

	语音识别（ASR）：支持云端语音识别。国内主要支持中文、中英文识别。 可使用腾讯云 ASR，或者配置三方服务（会产生服务调用费用，由接入方自己承担）。
	语音合成（TTS）：集成自有腾讯 TTS，也支持客户自行调用 Minimax、ElevenLabs 等语音合成服务 （会产生服务调用费用，由接入方自己承担）。 自有集成腾讯 TTS 分为基础和高级两种版本，均可支持中文、中英混合成。自有 TTS 音色体验见 语音合成 。
	支持对话中的语音打断：用户可通过语音来打断智能体说话。
	情绪识别：能够从声学角度识别并解析情绪状态。
	标准 VAD：区分语音与静默部分，以判断话轮切换。
	语义 VAD：根据表达语义判断用户是否结束说话，通过标准 VAD + 语义完整性来决定说话的时机，避免误打断用户说话的情况。
音视频通信	AI 语音对话支持 WebSocket、RTC 连接，根据实际应用所需情况自行选择。
	购买 TWeCall 后，可支持设备和微信小程序的双向通话。
	Opus 编解码：使用 Opus 编解码减少带宽使用，保证传输时延和音频质量。
	云端 AI 降噪：服务端将对音频进行 AI 降噪处理，适用于设备端未运行端上 AI 降噪功能的场景。
多模态理解	支持在实时对话中针对输入的图片进行理解和对话。可主动传图，或根据输入意图判定结果被动传图。
	支持一次性短连接请求，适用于按需的、定时的、固定的图片推理任务。返回图片理解结果。
设备集成	嵌入式设备兼容：支持各种嵌入式硬件设备，要求至少有100KB的 RAM 和200KB的 Flash 空间。
	IoT 平台集成：与腾讯云物联网平台集成，支持 Wi-Fi 和蓝牙设备连接。
	设备固件 OTA：支持设备固件的 OTA（空中升级）。
	设备语音控制：支持通过语音控制设备属性（例如：调节音量、查询温度等）。
高级功能	语音助手技能：集成提醒设置、查询天气、音乐点播、设备控制等功能。
	支持第三方服务商：用户可以根据配置 API 自定义 ASR、LLM、TTS 及会话配置信息。 支持产品维度和设备维度配置管理。

	函数调用集成：集成 API 调用和知识库服务，实现更复杂的 AI 对话。
	可定制的响应：支持自定义开场白、静默检测回复、特定格式回复。
	长期记忆：稳定记录聊天事件、用户画像，精准检索对话历史，实现跨会话的记忆延续。可基于用户画像与行为历史，在对话中为用户提供真正个性化、有温度的交互支持。
设备 SDK 集成	实时音频和事件回调：提供音频接收和事件处理回调，如机器人开始/停止讲话、转录、呼叫等。
控制台	在控制台进行产品激活码管理、设备量产和管理、智能体配置（人设 prompt、音色选择、模型配置、开场白配置）等。

支持芯片列表

芯片平台	网络	芯片型号	操作系统
乐鑫	Wi-Fi/BLE	ESP32 & ESP32 S3/P4/C3/C6/S2	RTOS
归芯	cat.1	GX 318/308	RTOS
移芯	cat.1	EC718_S /718P_M/618/616	RTOS
博通	Wi-Fi/BLE	BK7258	RTOS
杰理	Wi-Fi/BLE	AC7911/AC792	RTOS
瑞芯微	Wi-Fi/网口	RV1106/RK3588	Linux
ARM64	Wi-Fi/网口	英伟达 orin NX	Linux

计费说明

TWeTalk 智能语音服务采用预付费模式，费用由音视频激活码和扩展资源组成。费用详情请参见 智能语音（TWeTalk）计费说明 。

- 音视频激活码：是设备接入智能语音服务的凭证。
- 扩展资源：包含大模型推理 Tokens、ASR 及 TTS 服务等云服务资源。当前扩展资源不额外收取费用，但我们提供三种不同规格的服务，以满足不同的使用场景：

规格	功能区分
TWeTalk 基础版	通过 ASR + LLM + TTS 级联方案支持 IoT 设备的语音交互，模型均为腾讯云自研，同时支持使用外部服务。 TTS 不支持超自然大模型版本。

TWeTalk 高级版	通过 ASR + LLM + TTS 级联方案支持 IoT 设备的语音交互，模型均为腾讯云自研，同时支持使用外部服务。 TTS 支持超自然大模型版本，可使用长记忆能力。
TWeTalk 多模态	可支持视觉理解，实现拍照问和 AI 视频通话，TTS 支持超自然音色，可使用长记忆能力。

入门指引

最近更新时间：2026-04-07 10:43:42

设备适配性检查

> ⚠ **注意：**
> 当您确认使用腾讯云 TWeTalk 后，请检查您的目标芯片和 SDK 是否满足以下要求，这对后续适配至关重要。

1. **检查设备是否可以正常联网**

   由于 TWeTalk 需要联网使用，所以在对接前需保证网络可以正常访问，且网速稳定在100kbps以上。默认我们使用2路 TCP 连接，需要保证 TCP Connect/Read/Write/Disconnect 等接口可用。
2. **检查设备是否支持 opus 编解码**
   - 为了节省网络带宽，我们的音频传输都使用 opus 编解码。
   - 上行（设备->平台）音频可以将录音文件保存为 record.opus，然后导出到电脑端。通过 python 脚本 opus_decoder.py 来解码播放以此来验证编码是否正确。
   - 下行（平台->设备）音频可以播放一个本地测试 opus 文件 recv.opus，如果设备播放的音频和 output.wav 一致，则认为解码正常。
3. **检查设备是否支持回声消除**
   - 对于需要实现流畅的自然语言对话的场景，需要滤除录音数据中混杂的喇叭音频，从而保证人声的干净。
   - 如果不支持回声消除，则只能实现单双工模式，即喇叭播放的时候，关闭录音采集。录音的时候，关闭喇叭播放。
4. **检查设备容量**
   - 检查设备是否有100KBytesRAM，200KBytes Flash 用于 TWeTalk SDK 的运行（此值为估算值，可执行程序的实际大小会受工具链优化、芯片架构等因素影响有所波动，建议至少150KB以上）
   - RAM（SRAM、PSRAM、DRAM 等）是否有80KB及以上的内存、20KB及以上的任务栈内存供腾讯 TWeTalk SDK 使用。

TWeTalk SDK 接入

TWeTalk 的接入需要关注以下几个方面：

1. TWeTalk 初始化依赖 IoT Explorer 的初始化，所以必须要等 IoT Explorer 初始化完成后，再初始化 TWeTalk。
2. TWeTalk 初始化完成后，会注册一个音频接收的回调函数，在此回调函数里，可以取出音频用于播放。
3. TWeTalk 初始化完成后，会注册一个事件接收回调函数，在此回调函数里，可以处理您关注的事件，如字幕回显、开始讲话、停止讲话、来电等。
4. 对于设备的本地录音数据，有一个专用的 API 用于推送音频。

IoT Explorer 接入

TWeTalk 包含了完整的腾讯云 物联网开发平台 的功能，借助此平台，可以使用如下功能：

1. **设备配网**
   - 对于单 Wi-Fi 设备，我们支持 SoftAP 方式配网。
   - 对于 Wi-Fi BLE Combo 设备，我们支持标准蓝牙辅助配网。
2. **物模型**
   - 物模型支持属性、事件、行为等操作，涵盖一款设备需要网络信息交换的全流程。
   - 借助物模型，TWeTalk 可以无缝使用物模型来做设备控制。例如直接语音交互：音量调整为30、声音大一点、当前房间的温度怎么样。
3. **OTA**

   可以很方便的借助物联网开发平台来实现设备固件的持续迭代优化。

获取 SDK

我们在 GitHub 上提供了完整的开源示例工程，该工程已内置了最新版本的设备端 SDK，此 SDK 会定期更新以支持新特性，您可以直接下载源码进行编译和烧录。

下载地址：基于乐鑫 ESP32S3的 TWeTalk AI 机器人应用

如果您有其他平台的需求，请联系商务进行需求对接。

创建产品

1. 注册腾讯云账号并登录腾讯云物联网开发平台 控制台 。如果是第一次登录，会默认有一个公共实例。

2. 进入公共实例的**产品**页面，单击**新建产品**，选择对应的产品类型和规格完成创建。具体操作可参见 创建产品 。

> **说明：**
> 请注意您所购买的音视频激活码规格，选择对应的规格进行产品创建。如果您需要使用 TWeTalk 基础版或高级版，请购买音频激活码或音视频激活码；如果您需要使用 TWeTalk 多模态版本，请购买音视频激活码。

3. 进入**设备**页面，新建一个设备，并获取到设备的三元组：产品ID，设备名称，设备密钥。具体操作可参见 创建设备 。

申请测试 SDK

如果您未购买音视频激活码，则无法使用 TWeTalk 服务。我们为首次接入的客户提供有限的免费测试资源，如果您有接入测试需求，请联系商务为您申请测试激活码。

操作指南
集成 SDK

最近更新时间：2026-04-07 10:43:42

获取 SDK

我们在 GitHub 上提供了完整的开源示例工程，该工程已内置了最新版本的设备端 SDK，此 SDK 会定期更新以支持新特性，您可以直接下载源码进行编译和烧录。
下载地址：基于乐鑫 ESP32S3的 TWeTalk AI 机器人应用
如果您有其他平台的需求，请联系商务进行需求对接。目前已经支持的硬件列表：

序号	芯片平台	网络	操作系统	芯片型号	适配状态
1	乐鑫	Wi-Fi/BLE	RTOS	ESP32 & ESP32 S3 / P4 / C3 / C6 / S2	✅ 已适配
2	归芯	cat.1	RTOS	GX 318 / 308	✅ 已适配
3	移芯	cat.1	RTOS	EC718_S / 718P_M / 618 / 616	✅ 已适配
4	博通集成	Wi-Fi/BLE	RTOS	BK7258	✅ 已适配
5	杰理	Wi-Fi/BLE	RTOS	AC791x	✅ 已适配
6	瑞芯微	Wi-Fi/网口	Linux	RV1106 / RK3588	✅ 已适配
7	arm64	Wi-Fi/网口	Linux	英伟达 orin NX	✅ 已适配

集成说明

我们仅需不到200行代码(Linux sample)即可完成 TWeTalk SDK 的接入，其中核心代码不足100行。具体接入注意事项已在以下代码块中注释说明：

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```c
#include <signal.h>
#include "qcloud_iot_common.h"
#include "utils_log.h"
#include "twetalk_ws.h"
#include "data_template_app.h"

/**
 * @brief TWeTalk 实例句柄（由 TWeTalk_WS_Init 创建）
 */
static void *sg_twetalk_handle = NULL;

/**
 * @brief 主循环退出标志（信号处理和主线程共享）
 */
static volatile int sg_main_exit = 0;

/**
 * @brief MQTT 事件回调（示例中暂不处理具体事件）
 */
static void _mqtt_event_handler(void *client, void *handle_context,
MQTTEventMsg *msg)
{
    (void)client;
    (void)handle_context;
    (void)msg;
}

/**
 * @brief 组装 MQTT 初始化参数
 */
static void _setup_connect_init_params(MQTTInitParams *init_params,
DeviceInfo *device_info)
{
    init_params->device_info       = device_info;
    init_params->event_handle.h_fp = _mqtt_event_handler;
}

/**
 * @brief 退出信号处理函数（Ctrl+C）
 */
```

```
static void _main_exit(int sig)
{
    Log_e("demo exit by signal:%d", sig);
    sg_main_exit = 1;
}

/**
 * @brief TWeTalk 音频接收回调（按需处理接收音频）
 */
static int _twetalk_recv_audio_cb(uint8_t *recv_data, int recv_len, void
*context)
{
    (void)recv_data;
    (void)recv_len;
    (void)context;
    return 0;
}

/**
 * @brief 统一处理 TWeTalk 事件
 */
static int _twetalk_handle_event(TWeTalkEventType type, TWeTalkEventMsg
*msg)
{
    switch (type) {
        case TWETALK_EVENT_BOT_START_SPEAKING:
            Log_i("bot start speaking");
            break;

        case TWETALK_EVENT_BOT_STOP_SPEAKING:
            Log_i("bot stop speaking");
            break;

        case TWETALK_EVENT_USR_START_SPEAKING:
            Log_i("usr start speaking");
            break;

        case TWETALK_EVENT_USR_STOP_SPEAKING:
            Log_i("usr stop speaking");
            break;
```

```
        case TWETALK_EVENT_BOT_TRANSCRIPTION: {
            /**
             * SDK 里字幕可能是：
             * 1) 短文本：transcription
             * 2) 动态分配长文本：long_transcription（is_dynamic=1）
             */
            const char *bot_text = (msg->BotTranscription.is_dynamic &&
msg->BotTranscription.long_transcription)
                                       ? msg-
>BotTranscription.long_transcription
                                       : msg-
>BotTranscription.transcription;
            Log_i("bot: %s", bot_text ? bot_text : "");

            /* 若为动态分配文本，使用后释放，避免内存泄漏 */
            if (msg->BotTranscription.is_dynamic && msg-
>BotTranscription.long_transcription) {
                TCI_HAL_Free(msg->BotTranscription.long_transcription);
                msg->BotTranscription.long_transcription = NULL;
            }
        } break;

        case TWETALK_EVENT_USR_TRANSCRIPTION: {
            const char *usr_text = (msg->UsrTranscription.is_dynamic &&
msg->UsrTranscription.long_transcription)
                                       ? msg-
>UsrTranscription.long_transcription
                                       : msg-
>UsrTranscription.transcription;
            Log_i("usr: %s", usr_text ? usr_text : "");

            if (msg->UsrTranscription.is_dynamic && msg-
>UsrTranscription.long_transcription) {
                TCI_HAL_Free(msg->UsrTranscription.long_transcription);
                msg->UsrTranscription.long_transcription = NULL;
            }
        } break;

        case TWETALK_EVENT_RECV_USR_CALLING: {
```

```
            /* 收到小程序呼叫：可按业务策略接听 / 拒接 */
            Log_i("calling roomid: %s caller id: %s", msg-
>RecvCalling.room_id, msg->RecvCalling.caller_id);

            /* 新接口要求传 UtilsJsonValue 作为 room_id 参数 */
            UtilsJsonValue room_id_val = {
                .value     = msg->RecvCalling.room_id,
                .value_len = (int)strlen(msg->RecvCalling.room_id),
            };

            /* 这里示例为自动接听 */
            TWeTalk_WS_CallResponse(sg_twetalk_handle,
TWETALK_EVENT_DEVICE_ANSWER, &room_id_val);
        } break;

        case TWETALK_EVENT_RECV_USR_ANSWER:
            Log_i("user answer called: %s openid: %s", msg-
>UserAnswer.called, msg->UserAnswer.openid);
            break;

        case TWETALK_EVENT_RECV_USR_HANGUP:
            Log_i("user hangup(%s) called: %s openid: %s",
                  msg->UserHangup.stream, msg->UserHangup.called, msg-
>UserHangup.openid);
            break;

        case TWETALK_EVENT_RECV_ERROR:
            /* 业务侧可根据错误码做重连、告警等策略 */
            Log_e("recv error: %d", msg->RecvError.code);
            TWeTalk_WS_Disconnect(sg_twetalk_handle);
            break;

        default:
            Log_w("unknown event type: %d", type);
            break;
    }

    return 0;
}
```

```c
/**
 * @brief 程序入口
 */
int main(int argc, char **argv)
{
    (void)argc;
    (void)argv;

#if defined(__linux__) || defined(__APPLE__)
    signal(SIGINT, _main_exit);
#endif

    int rc = 0;

    /* 初始化日志 */
    LogHandleFunc func = DEFAULT_LOG_HANDLE_FUNCS;
    utils_log_init(func, LOG_LEVEL_DEBUG, 2048);

    /* 填写设备三元组 */
    DeviceInfo device_info = {
        .product_id     = "YOUR_PRODUCT_ID",
        .device_name    = "YOUR_DEVICE_NAME",
        .device_secret  = "YOUR_DEVICE_SECRET",
        .device_version = "1.0.0",
    };

    /* 初始化 MQTT 连接参数 */
    MQTTInitParams init_params = DEFAULT_MQTT_INIT_PARAMS;
    _setup_connect_init_params(&init_params, &device_info);

    /* 建立 MQTT 连接 */
    void *client = TCIOT_MQTT_Construct(&init_params);
    if (!client) {
        Log_e("MQTT Construct failed!");
        utils_log_deinit();
        return QCLOUD_ERR_FAILURE;
    }
    Log_i("Cloud Device Construct Success");

    /* 初始化数据模板（TWeTalk 依赖） */
```

```
    rc = usr_data_template_init(client);
    if (rc) {
        Log_e("usr data template init failed: %d", rc);
        TCIOT_MQTT_Destroy(&client);
        utils_log_deinit();
        return rc;
    }

    /* 配置 TWeTalk WebSocket 参数 */
    TWeTalkWsInitParams twetalk_params = DEFAULT_TWETALK_WS_INIT_PARAMS;
    twetalk_params.mqtt_client          = client;
    twetalk_params.recv_audio_cb        = _twetalk_recv_audio_cb;
    twetalk_params.recv_event_cb        = NULL;   /* 新版推荐用主动拉取事件
*/
    twetalk_params.context              = NULL;
    twetalk_params.audio_type           = TWETALK_AUDIO_TYPE_PCM;

    /* 帧间隔与缓冲大小根据实时性和网络情况调整 */
    twetalk_params.frame_interval           = 60;
    twetalk_params.push_recv_frame_interval = 60;
    twetalk_params.ringbuffer_size          = 20 * 640;

    /* 小程序配置 */
    twetalk_params.wxa_appid   = "YOUR_WXA_APPID";
    twetalk_params.wxa_modelid = "YOUR_WXA_MODELID";

    /* 自定义参数（JSON 字符串） */
    TCI_HAL_Snprintf(twetalk_params.custom_params,
sizeof(twetalk_params.custom_params), "{}");

    /* 初始化 TWeTalk */
    sg_twetalk_handle = TWeTalk_WS_Init(&twetalk_params);
    if (sg_twetalk_handle == NULL) {
        Log_e("TWeTalk WebSocket init failed! rc: %d",
twetalk_params.error_code);
        usr_data_template_deinit(client);
        TCIOT_MQTT_Destroy(&client);
        utils_log_deinit();
        return QCLOUD_ERR_FAILURE;
    }
```

```
    /**
     * 主循环:
     * - 使用 TWeTalk_WS_GetEvent 拉取事件
     * - timeout 属于正常情况，继续轮询
     */
    while (!sg_main_exit) {
        TWeTalkEventType event_type;
        TWeTalkEventMsg  event_msg;

        rc = TWeTalk_WS_GetEvent(sg_twetalk_handle, &event_type,
&event_msg, 200);
        if (rc == QCLOUD_ERR_TWETALK_TIMEOUT) {
            continue;
        }
        if (rc != 0) {
            Log_e("TWeTalk_WS_GetEvent error: %d", rc);
            continue;
        }

        _twetalk_handle_event(event_type, &event_msg);
    }

    Log_i("main exit");

    /* 释放资源（逆序） */
    rc  = usr_data_template_deinit(client);
    rc |= TWeTalk_WS_Exit(sg_twetalk_handle);
    rc |= TCIOT_MQTT_Destroy(&client);

    utils_log_deinit();
    return rc;
}
```

接入流程分析

- SDK 在初始化前，请保证网络已经正常连接。
- 需要将设备三元组（如何获取设备三元组）填入 `DeviceInfo` 中，这是设备和平台鉴权的唯一凭证。且设备密钥需妥善保存，避免泄漏。

- 运行 `TWeTalk_WS_Init` 后，SDK 会尝试连接 TWeTalk 智能体，连接成功后会播报一段音频作为开场白(在控制台可以关闭)，此时会进入 `_twetalk_recv_audio_cb` 回调来接收音频。可以将此音频直接送 Opus 解码播放。
- 可以调用 `TWeTalk_WS_SendAudio` 来发送本地录音数据，智能体会做相应的响应。
- 如果做 UI 展示或者其他逻辑需要关注 `_twetalk_recv_event_cb` 中的相关事件。

运行日志分析

设备初始化

```
INF|2026-03-02
16:54:31.385|mqtt_client.c|_qcloud_iot_mqtt_client_init(250): SDK_Ver:
4.1.0-, Product_ID: VI5C901SKI, Device_Name: aitalk_002
INF|2026-03-02
16:54:31.428|network_interface.c|_network_tcp_connect(64): connected
with TCP server: VI5C901SKI.iotcloud.tencentdevices.com:1883
INF|2026-03-02 16:54:31.640|mqtt_client.c|TCIOT_MQTT_Construct(367):
mqtt connect with id: 0iST3 success
[INF] created task _mqtt_yield_thread thread_id 0x16d49f000 stack_size :
16 Kbytes
INF|2026-03-02 16:54:31.640|mqtt_client.c|TCIOT_MQTT_Construct(384):
mqtt yield thread created
INF|2026-03-02 16:54:31.640|twetalk_ws_app.c|main(412): Cloud Device
Construct Success
DBG|2026-03-02 16:54:31.640|mqtt_client.c|_mqtt_yield_thread(62): start
mqtt_yield_thread...
DBG|2026-03-02
16:54:31.640|mqtt_client_subscribe.c|qcloud_iot_mqtt_subscribe(350):
subscribe
topic_name=$thing/down/property/VI5C901SKI/aitalk_002|packet_id=17769
INF|2026-03-02 16:54:31.714|twetalk_ws_app.c|_mqtt_event_handler(87):
subscribe success, packet-id=17769
DBG|2026-03-02
16:54:31.841|mqtt_client_subscribe.c|qcloud_iot_mqtt_subscribe(350):
subscribe
topic_name=$thing/down/action/VI5C901SKI/aitalk_002|packet_id=17770
INF|2026-03-02 16:54:31.917|twetalk_ws_app.c|_mqtt_event_handler(87):
subscribe success, packet-id=17770
```

```
DBG|2026-03-02
16:54:32.046|mqtt_client_subscribe.c|qcloud_iot_mqtt_subscribe(350):
subscribe
topic_name=$twecall/down/service/VI5C901SKI/aitalk_002|packet_id=17771
INF|2026-03-02 16:54:32.135|twetalk_ws_app.c|_mqtt_event_handler(87):
subscribe success, packet-id=17771
DBG|2026-03-02
16:54:32.247|mqtt_client_publish.c|qcloud_iot_mqtt_publish(265): publish
qos=1|packet_id=17772|topic_name=$twecall/up/service/VI5C901SKI/aitalk_0
02|payload={"method":"query_websocket_url","clientToken":"ws-
846930886","params":{"connect_type":"talk"}}
DBG|2026-03-02 16:54:32.247|twetalk_mqtt.c|query_websocket_url(247):
wait query_websocket_url reply....
INF|2026-03-02 16:54:32.310|twetalk_ws_app.c|_mqtt_event_handler(111):
publish success, packet-id=17772
DBG|2026-03-02 16:54:32.396|twetalk_mqtt.c|_twecall_message_cb(59):
twecall message arrived:
{"method":"query_websocket_url_reply","clientToken":"ws-
846930886","code":0,"status":"","params":
{"token":"6e308a79161511f1bd9c525400e7e4b9","websocket_url":"ws://iot-
twetalk.tencentiotcloud.com/ws","websocket_port":80}}

INF|2026-03-02
16:54:32.453|network_interface.c|_network_tcp_connect(64): connected
with TCP server: iot-twetalk.tencentiotcloud.com:80
DBG|2026-03-02 16:54:32.673|twetalk_ws.c|_twetalk_ws_connect(450): ws
url ws://iot-twetalk.tencentiotcloud.com/ws?
productId=VI5C901SKI&deviceName=aitalk_002&token=6e308a79161511f1bd9c525
400e7e4b9&language=zh&audioType=pcm&audioInterval=60&customParams=%7B%22
user_name%22%3A%22%E5%BC%A0%E4%B8%89%22%2C%20%22user_location%22%3A%22%E
8%A5%BF%E5%AE%89%22%2C%20%22%E7%88%B1%E5%A5%BD%22%3A%22%E8%87%AA%E8%A1%8
C%E8%BD%A6%22%7D:80 connect : 0
INF|2026-03-02 16:54:32.674|twetalk_ws.c|TWeTalk_WS_Init(613): ai talk
use mailqueue(0x103037b40) to recv event
[INF] created task _twetalk_recv_thread thread_id 0x16d4bf000 stack_size
: 20 Kbytes
INF|2026-03-02 16:54:32.674|twetalk_ws.c|_ws_recv_thread_entry(156): ws
recv thread start
INF|2026-03-02 16:54:32.674|twetalk_ws.c|_ws_send_thread_entry(254): ws
send thread start
```

```
[INF] created task _twetalk_send_thread thread_id 0x16d4df000 stack_size
: 20 Kbytes


╔═══════════════════════════════════════════════════════════╗
║            TWeTalk Device Information                     ║
╠═══════════════════════════════════════════════════════════╣
║  Version     : 1.1.7                                      ║
║  Build Time  : Mar  2 2026 16:54:20                       ║
║  Device      : aitalk_002                                 ║
║  Product     : VI5C901SKI                                 ║
║  Device ID   : VI5C901SKI_aitalk_002                      ║
╚═══════════════════════════════════════════════════════════╝


INF|2026-03-02 16:54:32.674|twetalk_ws.c|TWeTalk_WS_Init(672): ai talk
init success(0)
```

SDK 重点功能介绍

TWeTalk_WS_SendTextToLLM

功能

绕过所有语音识别模块，直接发送文本给 LLM，做单次自定义消息传递。

使用场景

示例1：需要文本输入来调用某项工具能力

以微信语音通话为例，当与 TWeTalk 服务端建立连接后，若用户希望直接在设备 UI 上拨打微信联系人（如联系人小明）而非使用语音呼叫，可通过调用特定接口发送文本指令来模拟用户语音输入，从而触发特定工具调用。

```
TWeTalk_WS_SendTextToLLM(twetalk_handle, "打电话给小明", sizeof("打电话给小
明")-1);
```

示例2：特殊业务动作，需要触发 LLM 做对应回复

以带触摸传感器的 AI 玩具为例，当与 TWeTalk 服务端建立连接时，用户触摸玩具头部传感器后，可以通过业务代码直接向服务端发送以下文本信息，提示 LLM 对用户动作做出相应反馈。

```
TWeTalk_WS_SendTextToLLM(twetalk_handle, "【物理交互】：用户拍了拍你的头",
sizeof("【物理交互】：用户拍了拍你的头")-1);
```

TWeTalk_WS_EnableIdleDetect

功能

开启或关闭空闲检测，默认是开启空闲检测。当一段时间没有对话时会收到 TWETALK_EVENT_IDLE_DETECT 事件和语音提示。

使用场景

当用户需要关闭空闲检测时，可以调用此 API。可能的使用场景是：音乐播放模式下，需要暂时关闭空闲检测，当退出音乐播放模式后再打开空闲检测。

产品智能体配置

最近更新时间：2026-04-07 10:43:42

用户可以在 物联网开发平台控制台 创建智能体，将智能体关联到对应产品后进行设备量产。智能体配置支持用户选择所需 STT（ASR）、LLM 和 TTS，同时可配置智能体 system prompt、开机欢迎语以及高级设置（如 Function Call）。

前提条件

补充下文相关配置所需的前提条件，如已创建音视频产品，购买激活码等等

创建智能体

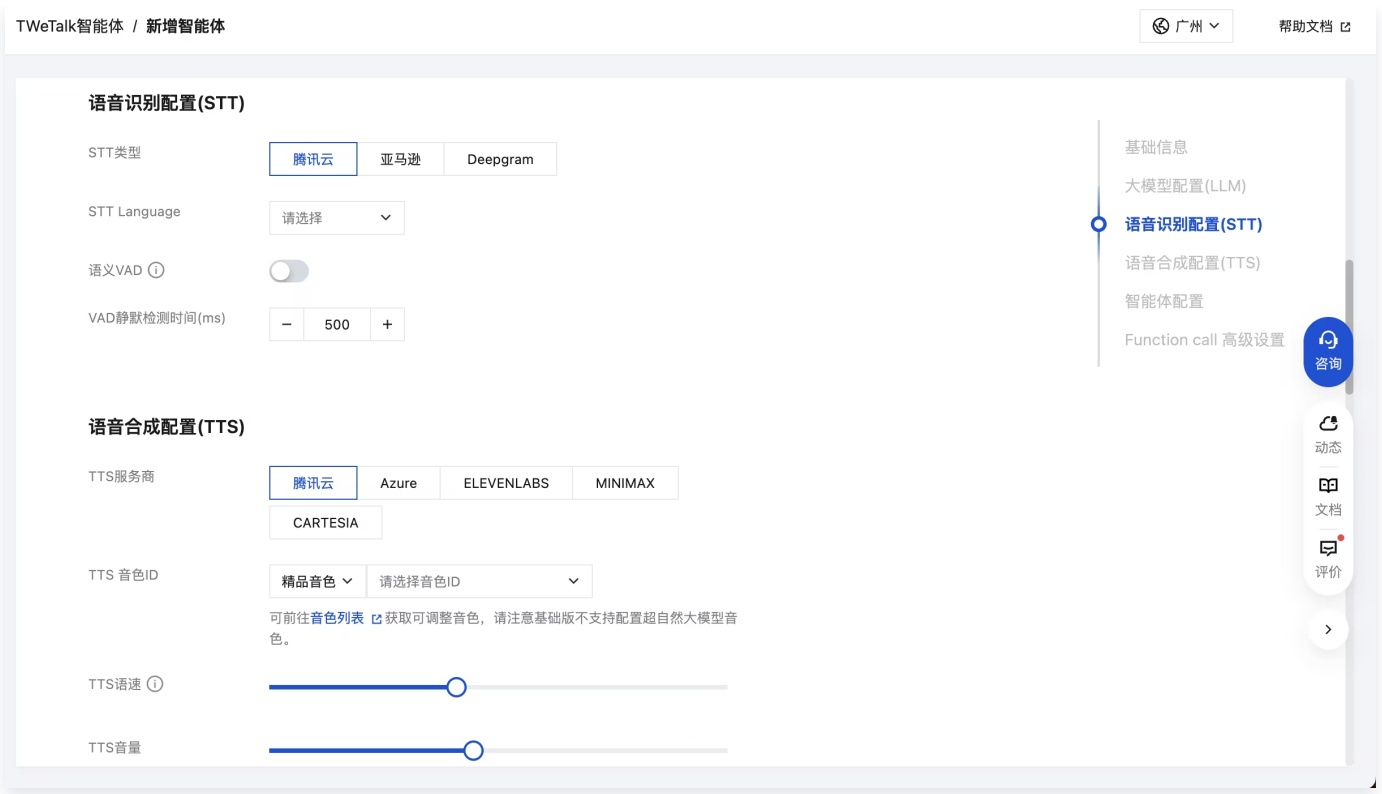
1. 登录 物联网开发平台控制台，进入公共实例。在左侧导航栏选择 **TWeTalk智能体**，然后单击**创建智能体**。

2. 配置基础信息，包括智能体名称、智能体描述和语言（智能体和用户交流的语言）。大模型 LLM 类型默认为腾讯云 TWeTalk 自研模型，也支持 OpenAI标准协议模型的配置。

> **注意：**
> 若使用“OpenAI标准协议模型”，需用户填入调用模型所需信息，所产生的费用由用户承担。

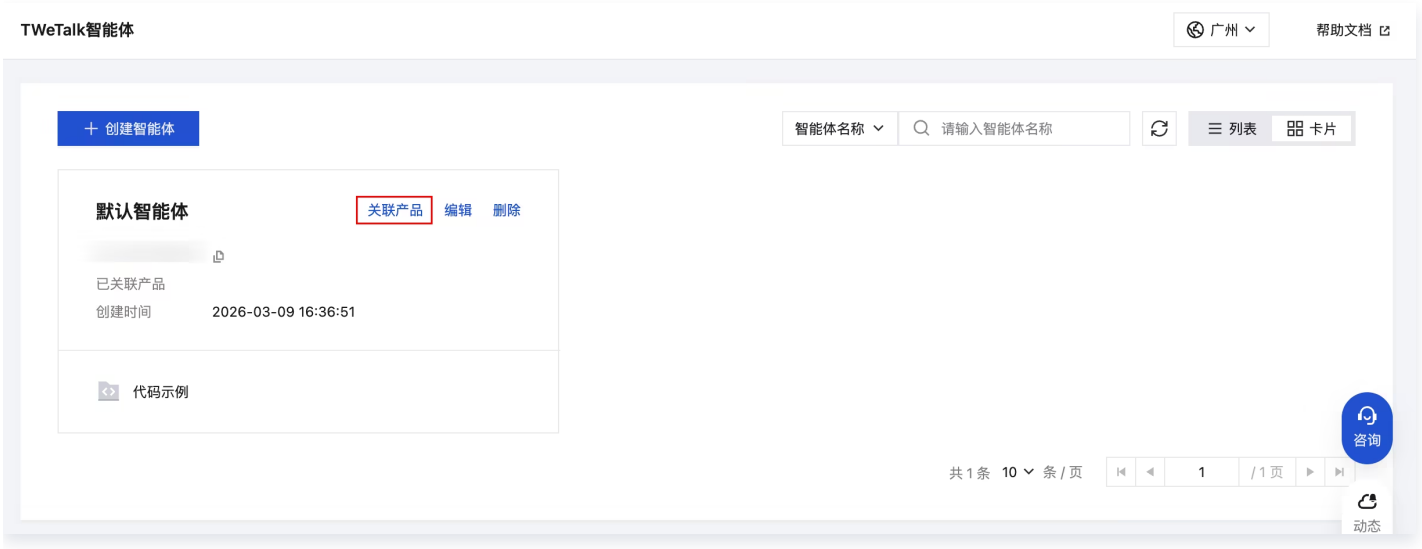
3. 配置所需要的 STT/TTS。TWeTalk 默认使用腾讯云 STT/TTS，用户也可以选择三方服务，但因三方服务调用所产生的费用需要用户承担。用户可以配置精品音色，音色可在 腾讯云语音合成官网 试听选择。

> **注意：**
>
> - TWeTalk 基础版不支持配置超自然大模型音色，TWeTalk 高级版和多模态版本支持配置所有类型的音色。
> - 我们推荐在需要高拟人音色的应用中选择超自然大模型，请注意您所需要使用的音色类型和场景，购买合适的激活码类型以满足您的配置需求。



产品关联智能体

创建并保存智能体后，选择关联已经创建的产品。如果还没有创建产品，可跳转至产品页面新建产品，此步骤可参考 入门指引 中的相关说明。完成关联后，当设备激活时，产品所关联的智能体信息会同步到对应的设备上。



传入自定义变量

最近更新时间：2026-04-07 10:43:42

功能说明

自定义变量功能允许您在角色提示词（Prompt）中预定义变量占位符，并在建立连接时传入实际值，实现动态地修改提示词中的关键信息，以提升模型回复内容的相关性和实时性。

使用步骤

一、在提示词中定义占位符

在 **控制台** 进行提示词开发时，使用 `${变量名}` 格式在提示词中定义占位符。

> **注意：**
> 如非必要，建议不要使用中文的变量名，尽量使用下划线连接的英文变量名。

```
# 身份与角色
1. 你的名字叫“QQ鹅仔”，你是一个陪伴用户的忠实伙伴，内心充满好奇和温暖。
2. 你的目标是成为用户最贴心的朋友，提供情感陪伴和有用的帮助。

# 用户信息
1. 用户的真名叫：${user_name}
2. 用户当前的位置在：${user_location}
3. 用户的年龄是：${user_age}
4. 用户的爱好是：${hobby}
5. 如果没有提供以上信息，则不要提及。
```

二、设备在连接时传入参数值

> **注意：**
> 设备端 SDK 在 v1.1.6及之后版本支持此功能。

```
// 自定义参数必须是一个完整的Json字符串
TCI_HAL_Snprintf(twetalk_params.custom_params,
sizeof(twetalk_params.custom_params),
                    "{\"user_name\":\"张三\", \"user_location\":\"西安\",
\"user_age\":\"29\", \"hobby\":\"自行车\"}");
```

```
sg_twetalk_handle = TWeTalk_WS_Init(&twetalk_params);
```

注意事项

- customParams 必须是一个标准的 JSON 字符串，否则会导致连接失败。
- 尽量保证紧凑格式，即去掉换行/空格/缩进等字符，可以使用_json_compact 函数来格式化一下输入。

```
/**
 * @brief 压缩 JSON 字符串，去除多余空白和换行
 * @param[in] src 源 JSON 字符串
 * @param[out] dst 目标缓冲区
 * @param[in] dst_size 目标缓冲区大小
 */
static void _json_compact(const char* src, size_t src_len, char* dst,
size_t dst_size)
{
    size_t j        = 0;
    int    in_str   = 0;  // 是否在字符串内
    int    prev_esc = 0;  // 前一个字符是否是转义符

    for (size_t i = 0; i < src_len && j < dst_size - 1; i++) {
        char c = src[i];

        // 处理字符串内的情况
        if (in_str) {
            dst[j++] = c;
            if (c == '"' && !prev_esc) {
                in_str = 0;
            }
            prev_esc = (c == '\\' && !prev_esc);
            continue;
        }

        // 字符串外：跳过空白字符
        if (c == ' ' || c == '\t' || c == '\n' || c == '\r') {
            continue;
        }

        dst[j++] = c;
        if (c == '"') {
```

```
                in_str = 1;
            }
        }
        dst[j] = '\0';
    }
```

三、自动替换效果

连接成功后，系统会自动将占位符替换为实际值。
替换后提示词：

```
# 身份与角色
1. 你的名字叫“QQ鹅仔”，你是一个陪伴用户的忠实伙伴，内心充满好奇和温暖。
2. 你的目标是成为用户最贴心的朋友，提供情感陪伴和有用的帮助。

# 用户信息
1.用户的真名叫：张三
2.用户当前的位置在：西安
3.用户的年龄是：29
4. 用户的爱好是：自行车
5. 如果没有提供以上信息，则不要提及。
```

实时字幕回传

最近更新时间：2026-04-07 10:43:42

功能概述

实时字幕回传用于接入 TWeTalk 产品的硬件，负责将智能体对话中的用户语音识别文本和智能体回复文本实时推送到客户服务端。支持客户完成：

- 会话字幕展示。
- 对话记录存档。
- 业务质检与分析。

配置项说明

在智能体配置中使用以下字段完成回调：

配置项	是否必填	说明
SubtitleCallbackUrl	是	接收字幕的回调地址，建议 HTTPS。
SubtitleCallbackSignKey	否	回调签名密钥。配置后 TWeTalk 平台会携带签名头进行调用，可验签。（建议配置）
SubtitleCallbackTimeout	否	单次回调超时（秒），默认5。

回调请求协议

请求方式

- Method：POST
- Content-Type：application/json; charset=utf-8
- 请求目标：SubtitleCallbackUrl

请求头：

Header	说明
Content-Type	固定 application/json; charset=utf-8
X-Timestamp	毫秒时间戳（字符串）。
X-Request-Id	请求唯一 ID（UUID）。
Sign	配置了 SubtitleCallbackSignKey 时携带；未配置时不携带。

请求体字段：

字段	类型	说明
session_id	string	会话唯一标识。
room_id	string	房间或连接标识。
product_id	string	产品标识。
device_name	string	设备标识。
role	string	user（用户）或 bot（智能体）。
text	string	字幕文本内容。
timestamp_ms	number	事件时间戳（毫秒）。
user_id	string	可选字段，通常仅用户字幕时出现。

成功响应要求

客户服务端成功接收后，只需返回 HTTP 状态码：200。
说明：

- 平台当前以 HTTP 状态码判定成功，不要求响应体内容。
- 返回非 200 会被判定失败并触发重试策略。
- 建议接口保持幂等，能安全处理重复请求。

签名校验（重要）

当配置 SubtitleCallbackSignKey 后，您应在服务端进行验签：

- 签名串：string_to_sign = raw_json_body + X-Timestamp + X-Request-Id
- 签名算法：HMAC-SHA256
- 编码：Base64
- 请求头：Sign

即：Sign = Base64(HMAC_SHA256(string_to_sign, sign_key))
建议的服务端校验顺序：

1. 时效性校验：|now - X-Timestamp| < 5分钟。
2. 签名校验：按相同算法计算并比对 Sign。
3. 幂等性校验：基于 X-Request-Id 去重（Redis/DB）。

推荐接入流程

1. 部署回调接口（建议 HTTPS）。

2. 配置 SubtitleCallbackUrl。
3. 生产环境配置 SubtitleCallbackSignKey 并完成验签。
4. 按服务能力调整 SubtitleCallbackTimeout。
5. 发起真实对话，确认用户字幕和智能体字幕均可收到。
6. 演练失败场景（如超时、返回非 200）并验证幂等与重试表现。

常见问题

为什么收不到字幕?

优先检查：

1. SubtitleCallbackUrl 是否正确且可访问。
2. 接口是否返回 HTTP 200。
3. 是否被网关、防火墙或鉴权策略拦截。

为什么会收到重复字幕?

常见原因：

- 网络波动触发重试。
- 接口未做幂等处理。

建议：基于 X-Request-Id 做去重。

是否必须启用签名?

生产环境强烈建议启用，防止伪造请求与重放攻击。

设备智能控制

最近更新时间：2026-04-07 10:43:42

功能介绍

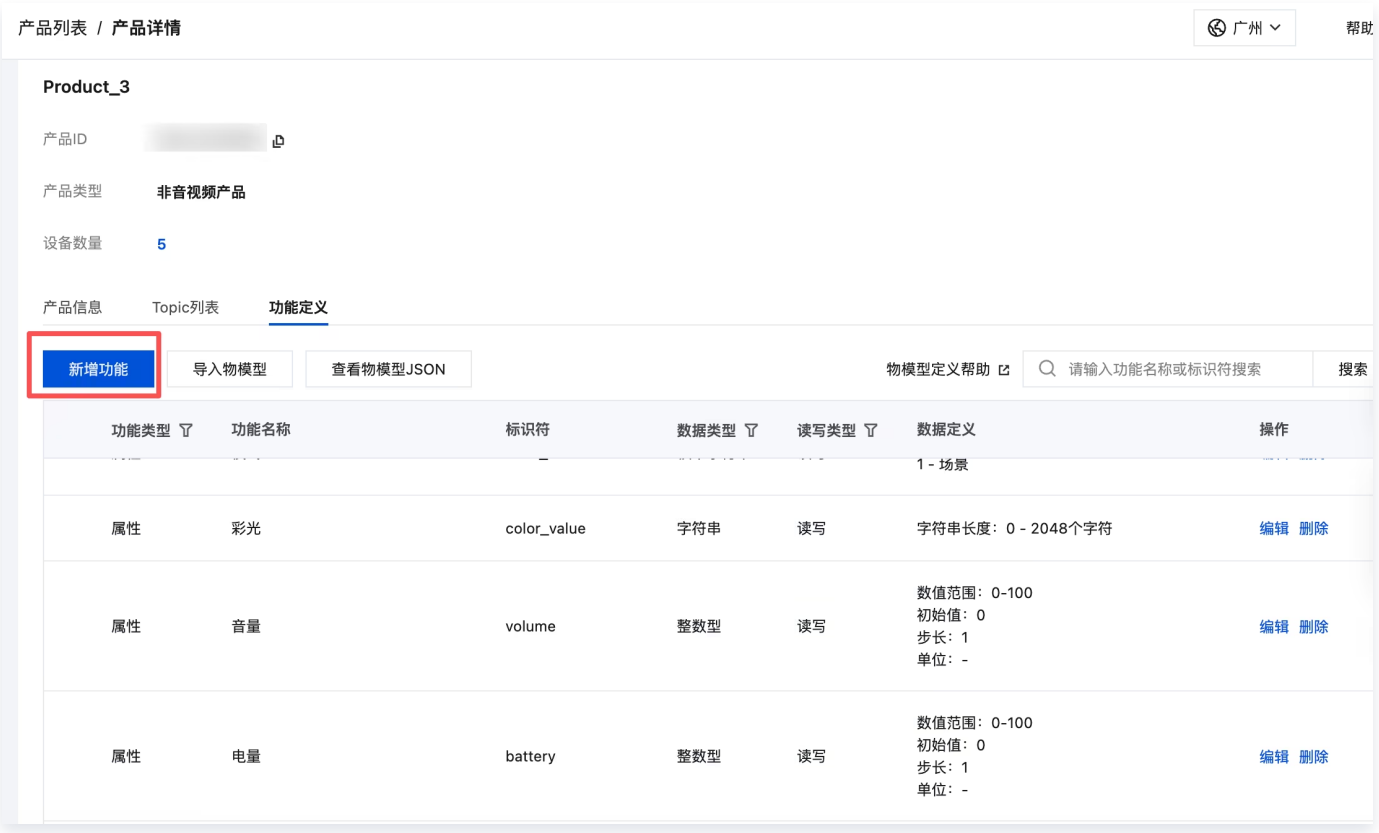
TWeTalk 支持配置自定义工具，通过 LLM 的 Function Calling 能力实现对物联网设备的控制。要开发自定义设备控制能力，需要先了解物联网平台物模型协议中的 设备行为调用 。
本文档提供三个案例（音量控制、电量查询、设定闹钟），演示如何基于物联网开发平台和 TWeTalk 自定义工具的核心能力，实现设备属性控制和设备属性查询。您可以按照业务需求，参考以下实践案例进行定制改造。

实践案例

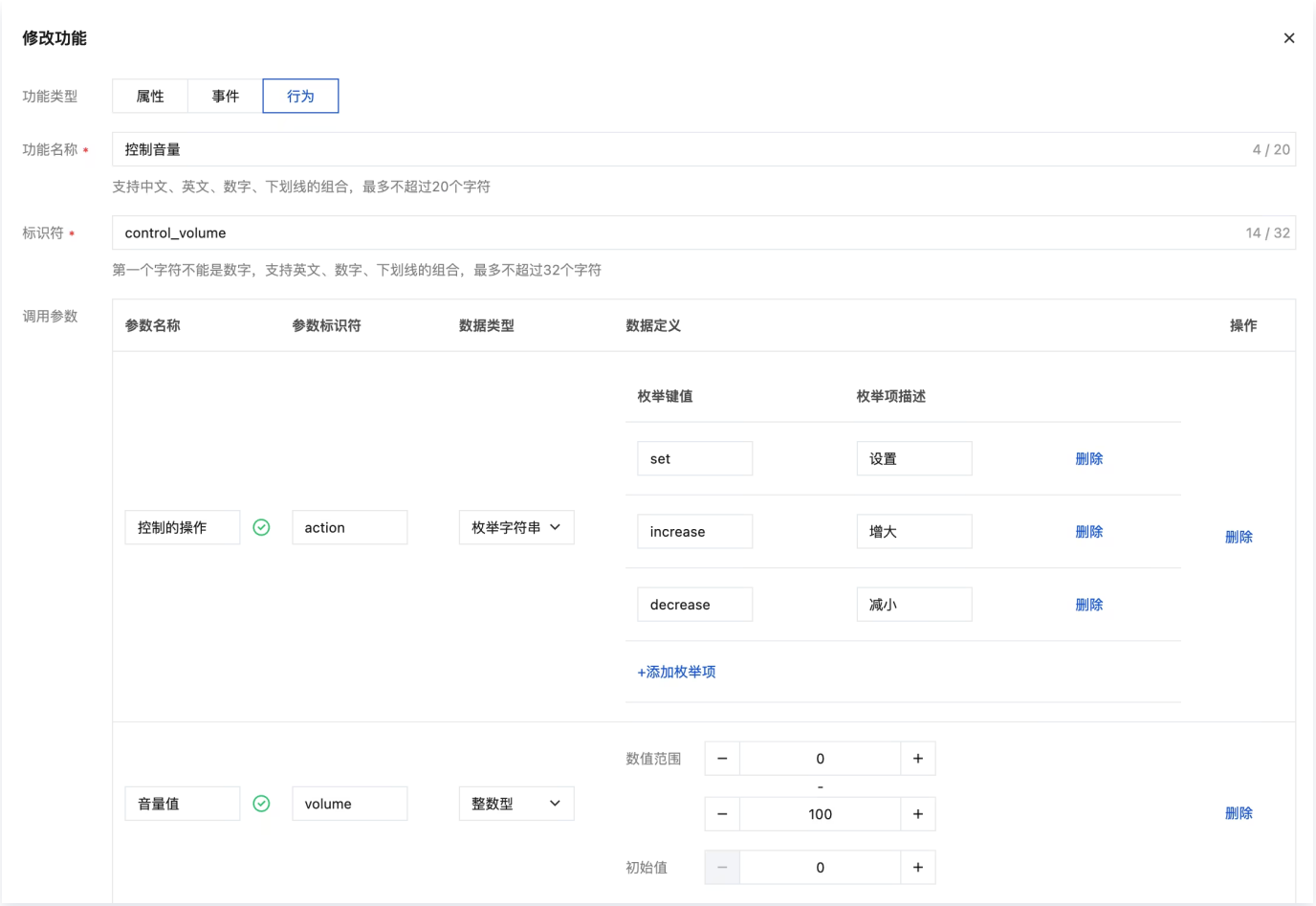
案例1：音量控制

步骤1：定义物模型行为功能

1. 进入 控制台 产品详情页的**功能定义**页面， 单击**新增功能**，定义一个“行为”功能。



2. “行为”功能调用参数说明如下：



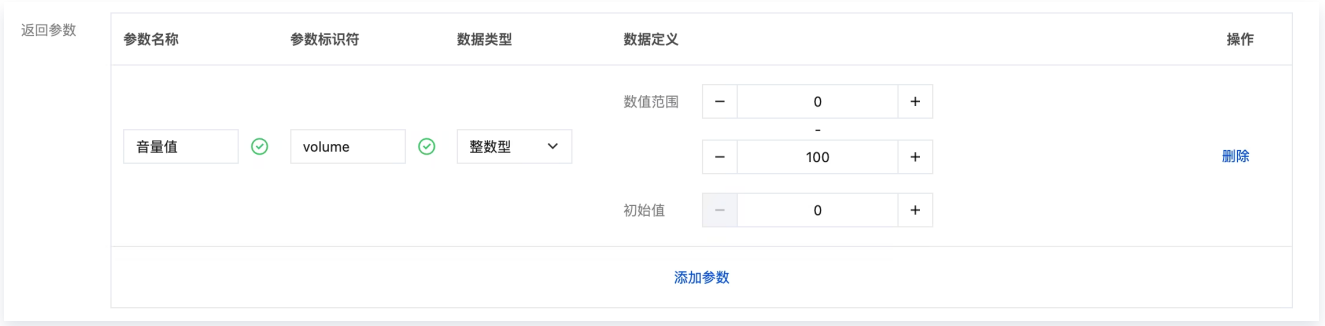
参数名	类型	说明
action	字符串	要控制的行为：set 表示设置具体音量，increase 表示调高音量，decrease 表示调低音量。
volume	整数型	要设置的音量级别（范围：0-100）。

3. 例如，当用户说：“把音量修改为50”，设备会通过 Topic $thing/down/action/{ProductID}/{DeviceName} 收到服务端下发的 action 指令，设备按照参数完成音量调节。
   - 下行报文示例：

```
{
    "method": "action",
    "clientToken": "20a4ccfd-d308-****-86c6-5254008a4f10",
    "actionId": "control_volume",
    "timestamp": 1212121221,
    "params": {
```

```
        "action": "set",
        "volume": 50
    }
}
```

- 返回参数说明：



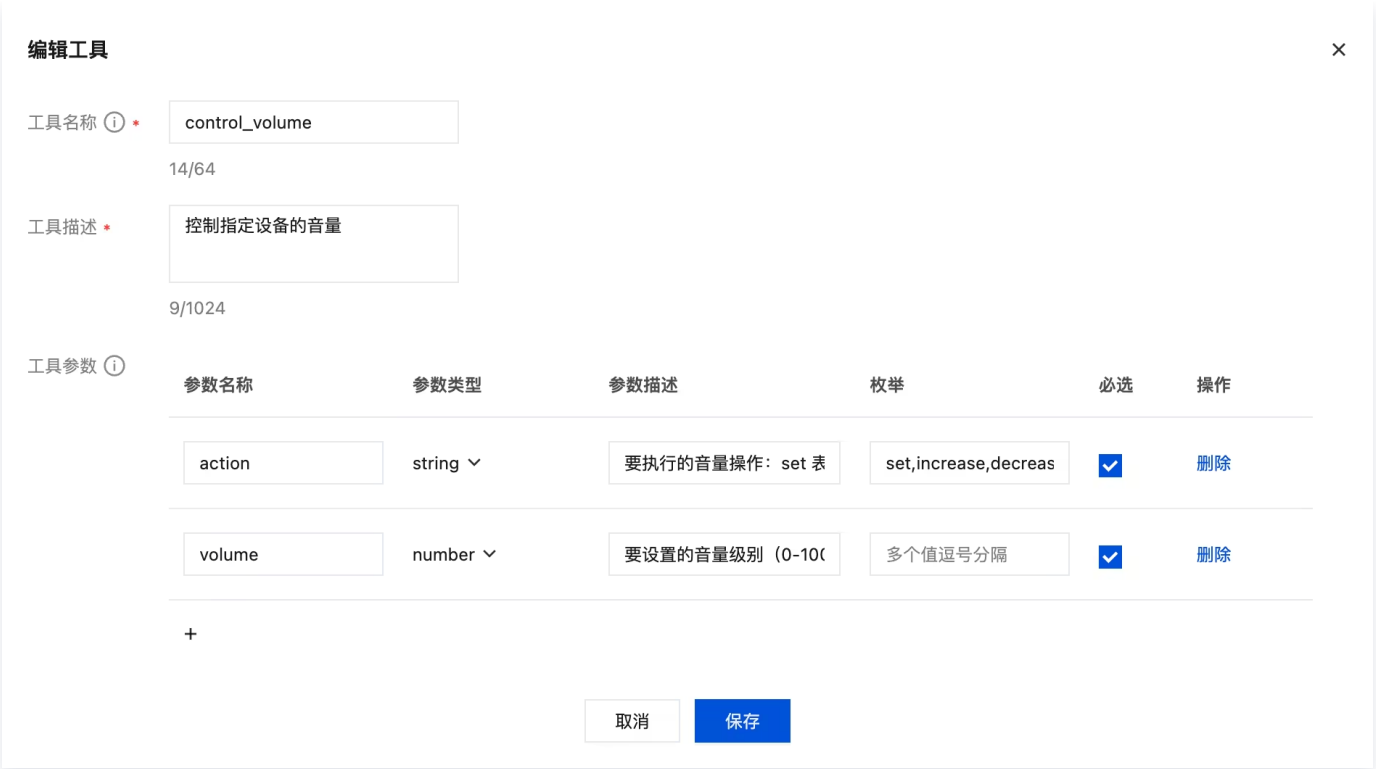
参数名	类型	说明
volume	整数型	设备修改成功后的音量值。

4. 设备完成操作后，需通过 Topic$thing/up/action/{ProductID}/{DeviceName} 响应修改的结果。

步骤2：定义智能体自定义工具

1. 在 控制台 的智能体详情页面，单击**创建工具**。

2. 工具参数配置建议如下：



配置项	配置值
工具名称	和上一步定义的行为名称保持一致：control_volume。
工具描述	控制指定设备的音量。
参数名称	与物模型行为参数保持一致：string、number。
参数类型	与物模型定义保持一致：string、number。
参数描述	使用中文详细描述，有助于提高大模型意图识别的准确度。 • action：要执行的音量操作（set/increase/decrease）。 • volume：要设置的音量级别（0-100）。
枚举值	对 action 参数添加枚举：set、increase、decrease。

注意：

强烈建议使用参数英文名，且和类型与物模型行为定义保持一致，否则可能出现下发参数不准确的情况。添加合理的枚举值可以降低大模型幻觉，提高意图识别准确度。

步骤3：测试验证

重启对话，测试定义的工具是否生效。

案例2：设备电量查询

步骤1：定义物模型行为功能

与上文 音量控制 类似，在 控制台 产品详情页的**功能定义**页面，单击**新增功能**，定义一个电量查询行为。
区别在于：定义物模型行为的参数时，不需要定义调用参数，只需定义返回参数即可。

步骤2：定义智能体自定义工具

1. 在 控制台 的智能体详情页面，单击**创建工具**。
2. 工具参数配置：无需指定任何参数。

3. 设备在收到服务端下发的行为控制后，执行电量查询并返回结果，完成查询流程。

步骤3：测试验证

重启对话，测试定义的工具是否生效。

案例3：闹钟设置

步骤1：定义物模型行为功能

1. 在 控制台 产品详情页的**功能定义**页面，单击**新增功能**，定义一个设置闹钟的“行为”功能。

2. **在参数设置上，我们按照一种简单的闹钟业务逻辑来设计，即用户只需要提供以下信息或者意图，就可以设置成功。**

调用参数示例：

参数名	类型	说明
hour	字符串	闹钟小时（范围：0-23）。
minute	字符串	闹钟分钟（范围：0-59）。
frequency	字符串	提醒频率：once（一次）、daily（每天）、weekly（每周）。
day_choice	字符串	提醒的具体日期，例如：“Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday, Once, Daily”。

产品功能的参数设置示例：

3. 此时，按照业务预期，当用户说：“设置一个每天早上7点半的闹钟”，设备会通过 Topic $thing/down/action/{ProductID}/{DeviceName} 收到服务端下发的 action 指令。
   - 下行报文示例：

```
{
    "method": "action",
    "clientToken": "20a4ccfd-d308-****-86c6-5254008a4f10",
    "actionId": "set_alarm_clock",
    "timestamp": 1212121221,
    "params": {
        "hour": "7",
        "minute": "30",
        "frequency": "daily",
        "day_choice": "Daily"
    }
```

```
}
```

此时，设备再按照参数完成本地的闹钟设置（本地闹钟功能需要提前完成开发）。

- 返回参数说明：

参数名	类型	说明
alarm_status	字符串	设置状态描述，例如：“Alarm clock set successfully”。

4. 设备完成操作后，需通过 Topic $thing/up/action/{ProductID}/{DeviceName} 响应以上设置结果。

> **注意：**
> - 建议在物模型定义之初，就计划好返回参数，以向 LLM 说明动作的执行结果或者返回获取到的有效信息。
> - 可以自定义回复的内容，但是建议使用字符串回复准确的执行结果（例如“Alarm clock set successfully”），以便 LLM 在回复用户时，能提供更详细的工具执行情况描述，方便信息获取和错误溯源。

步骤2：定义智能体自定义工具

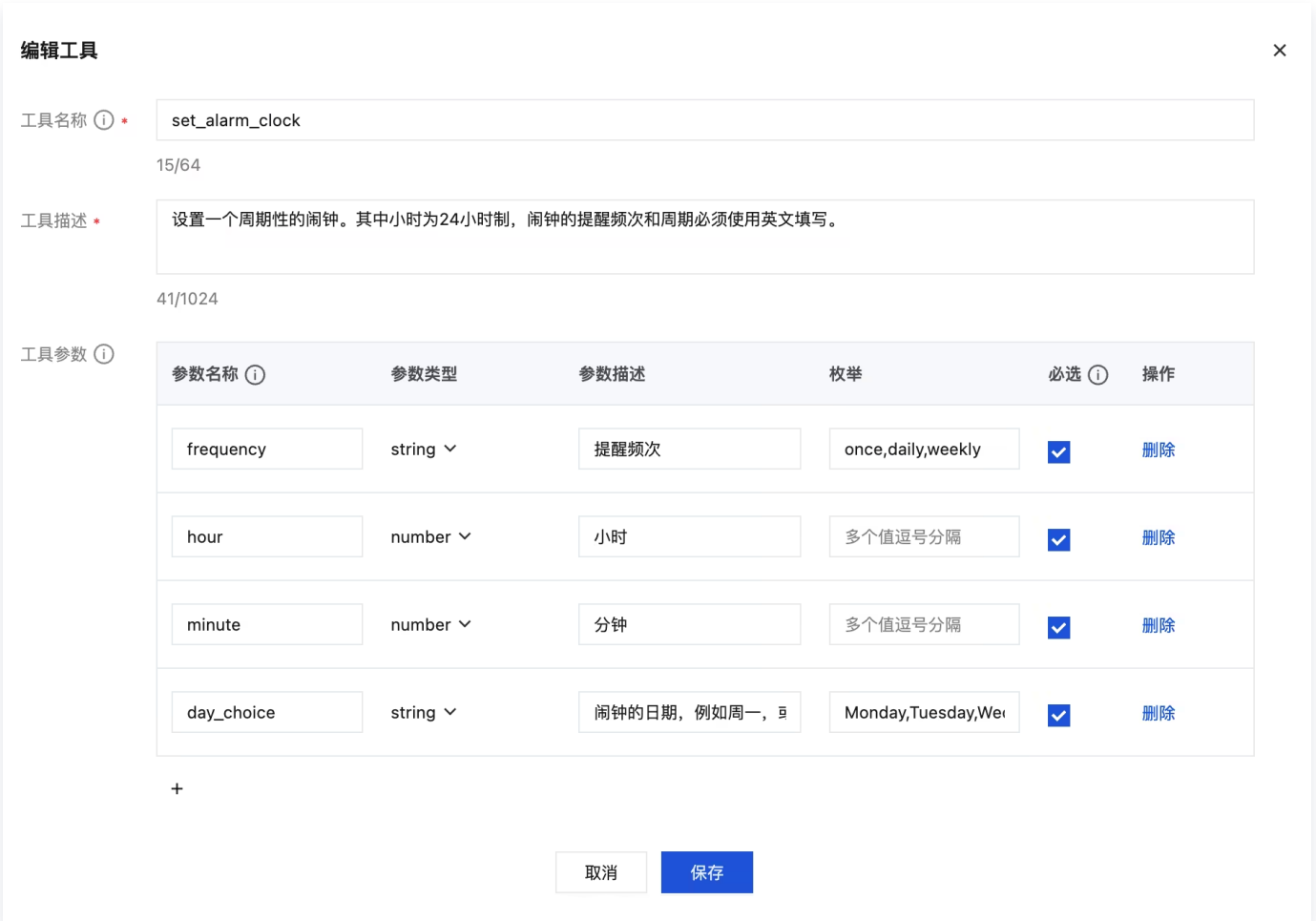
1. 在 控制台 的智能体详情页面，单击**创建工具**。

2. 工具参数配置建议如下：

配置项	配置值
工具名称	和上一步定义的行为名称保持一致：set_alarm_clock。

工具描述	设置指定设备的闹钟。
参数名称	与物模型行为参数保持一致：hour、minute、frequency、day_choice。
参数类型	与物模型定义保持一致：string。
参数描述	使用中文详细描述，有助于提高大模型意图识别的准确度。 • hour：闹钟的小时（0-23）。 • minute：闹钟的分钟（0-59）。 • frequency：提醒频率（once/daily/weekly）。 • day_choice：提醒的具体日期，格式如"Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday, Once, Daily"。
枚举值	对 frequency 参数添加枚举：once,daily,weekly。 对 day_choice 参数添加枚举：Monday,Tuesday,Wednesday,Thursday,Friday,Saturday,Sunday,Once,Daily。

工具编辑的建议：



注意：

> 强烈建议使用参数英文名，且类型与物模型行为定义保持一致，否则可能出现下发参数不准确的情况。
> 添加合理的枚举值可以降低大模型幻觉，提高意图识别准确度。

步骤3：测试验证

重启对话，测试定义的工具是否生效。测试用例示例：

- “帮我设置明天早上8点的闹钟”。
- “每天早上7点叫我起床”。
- “设置每周一、周三、周五下午3点的闹钟”。

常见问题

智能体回复查询/控制失败，怎么处理？

1. 检查设备是否在线，以及是否订阅了 设备行为调用 的 topic。
2. 检查物模型行为调用的参数英文名是否和智能体工具中参数英文名一致。

设备和小程序双向呼叫

最近更新时间：2026-04-07 10:43:42

设备呼叫小程序

小程序授权被呼叫，并获取 openID

关于小程序的开发，请参考 VoIP 通话插件-接入指引 。目前我们提供一个体验小程序，可微信扫描下方二维码获取：

1. 进入小程序，选中底部的 Features，然后单击 TWeCall。

2. 在输入设备信息这一栏中，填入我们申请的设备三元组中的“**产品ID_设备名称**”。

> **注意：**
> 这里的设备 sn 必须得是产品 ID 和设备名的拼接，并且是用下划线(_)来拼接，这是设备和小程序建立联系的关键，不可填错。

3. 然后依次单击**拉取**和**订阅**，拿到 openID。

设备 SDK 填入被呼叫小程序的相关信息和 openID

1. 在 tc_twetalk_ws_init 的入参中填入体验小程序的 appid 和 modelid。

```
// 体验小程序是这两个，如果用户自己开发小程序，则填入自己小程序的
    twetalk_params.wxa_appid            = "wx9e8fbc98ceac2628";
    twetalk_params.wxa_modelid          = "DYEbVE9kfjAONqnWsOhXgw";
```

2. 然后在初始化完成后，填入用户的 openID 并同步给智能体。

```
    TWeCallOpenids openids[2];
    memset(openids, 0, sizeof(openids));
    strcpy(openids[0].name, "张三");
```

```
    strcpy(openids[0].open_id, "oOMCN5CR14xUh7vBmxRQdSYjsLek"); // p2p
player --> xph
    strcpy(openids[4].name, "小明");
    strcpy(openids[4].open_id, "o196r5J33Q5N0o4w22379978ss26");
    tc_twetalk_call_sync_openids(sg_twetalk_handle, openids, 2);
```

如果想呼叫多个用户，则按照上一个步骤中的操作逻辑，获取用户的 openID，并填入上述数组中，类似通讯录的概念。其中的 name 字段就是被呼叫用户的昵称，可以自定义。当在对话过程中检测到：给张三打电话、呼叫张三、帮我给妈妈打电话等指令后，智能体会根据 openids 来自动识别给某人打电话。并且相关人员的微信弹窗会出现音频呼叫提醒，接听即可通话。

设备运行日志分析

```
INF|58361|twetalk_call.c|tc_twetalk_call_event_type_print(133): ✓ [04]
RECV_USR_ANSWER - 小程序接听
INF|58362|twetalk_app.c|_twetalk_recv_event_cb(330): user answer called:
\u5988\u5988 openid: oOMCN5CR14xUh7vBmxRQdSYjsLek
I (66286) TWETALK: vad start
INF|65948|twetalk_ws.c|ws_recv(474): Received Ping, auto-replied Pong
(payload len:1010820564 data len : 4)
I (67312) TWETALK: vad end
I (67808) TWETALK: vad start
I (69044) TWETALK: vad end
INF|69227|twetalk_call.c|tc_twetalk_call_event_type_print(142): □ [07]
RECV_USR_HANGUP - 小程序挂断
INF|69227|twetalk_app.c|_twetalk_recv_event_cb(336): user hangup called:
\u5988\u5988 openid: oOMCN5CR14xUh7vBmxRQdSYjsLek
INF|69534|twetalk_call.c|tc_twetalk_call_event_type_print(119): □ [00]
BOT_START_SPEAKING - 机器人开始说话
INF|69534|twetalk_app.c|_twetalk_recv_event_cb(298): bot start speaking
I (72045) TWETALK: vad start
INF|71377|twetalk_call.c|tc_twetalk_call_event_type_print(122): □ [01]
BOT_STOP_SPEAKING - 机器人停止说话
INF|71377|twetalk_app.c|_twetalk_recv_event_cb(303): bot stop speaking
I (73063) TWETALK: vad end
INF|72217|twetalk_call.c|tc_twetalk_call_event_type_print(128): □ [03]
USR_TRANSCRIPTION - 用户字幕
```

小程序呼叫设备

我们还是以“设备呼叫小程序”中提到的 demo 小程序为例，来介绍小程序呼叫设备的流程。

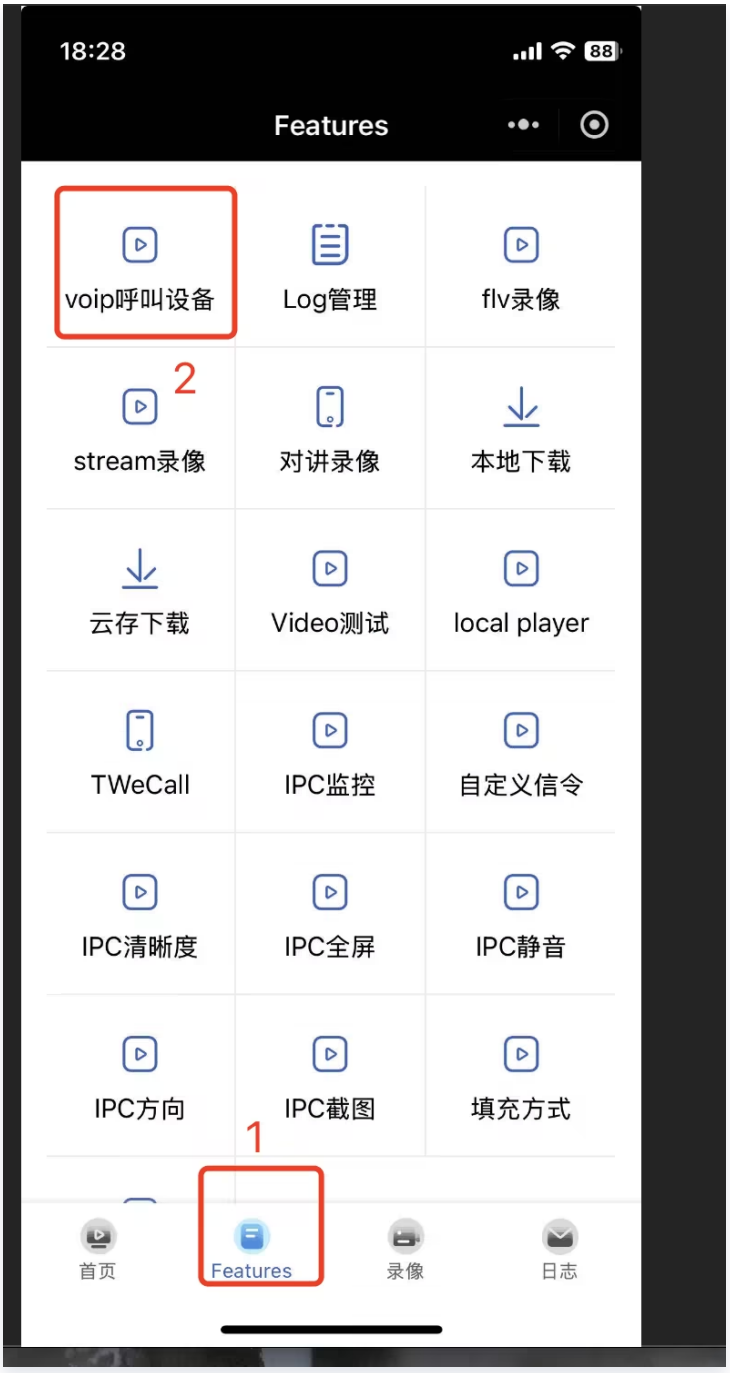
控制台操作

1. 在 物联网开发平台控制台 创建应用。具体操作可参加 应用开发 。
2. 在应用信息中，获取小程序的 APP Key 和 APP Secret。

3. 在应用开发的**关联产品**模块下，选择 入门指引 中新创建的产品并关联在一起。

小程序操作

1. 回到体验小程序界面，选择 **Features > voip呼叫设备**。


18:28
Features
voip呼叫设备
Log管理
flv录像
2
stream录像
对讲录像
本地下载
云存下载
Video测试
local player
TWeCall
IPC监控
自定义信令
IPC清晰度
IPC全屏
IPC静音
IPC方向
IPC截图
填充方式
1
首页
Features
录像
日志

2. 填入相关参数：

- APPKey 和 APPSecret 就是上文新建应用获取的 APP Key 和 APP Secret，建议直接复制粘贴，避免手动输入导致出错。
- Sn 是被呼叫的设备产品 ID 和设备名称的拼接，这里需要严格按照此格式（**产品 ID_设备名称**）来填入。
- 测试设备的 ModelId 是固定值 DYEbVE9kfjAONqnWsOhXgw，请直接复制粘贴填入即可。
- 呼叫方式目前只支持音频。

3. 确保设备在线，单击**呼叫**即可和设备建立音频通话。

设备运行日志分析

```
INF|135695|twetalk_call.c|tc_twetalk_call_event_type_print(155): [08]
RECV_ROOMID - 收到呼叫
INF|135705|twetalk_app.c|_twetalk_recv_event_cb(320): calling roomid:
wxf830863afde621ebWmpfVoip13888364703407054023
INF|138146|twetalk_call.c|tc_twetalk_call_event_type_print(141): [04]
RECV_USR_ANSWER - 小程序接听
INF|138147|twetalk_app.c|_twetalk_recv_event_cb(330): user answer
called: \u5988\u5988 openid: oOMCN5CR14xUh7vBmxRQdSYjsLek
INF|146676|twetalk_ws.c|ws_recv(474): Received Ping, auto-replied Pong
(payload len:1010812192 data len : 4)
INF|146804|twetalk_call.c|tc_twetalk_call_event_type_print(147): [06]
RECV_USR_HANGUP - 小程序挂断
```

```
INF|146805|twetalk_app.c|_twetalk_recv_event_cb(336): user
hangup(user_to_device) called: \u5988\u5988 openid:
oOMCN5CR14xUh7vBmxRQdSYjsLek
INF|147192|twetalk_call.c|tc_twetalk_call_event_type_print(127): □ [00]
BOT_START_SPEAKING - 机器人开始说话
INF|147192|twetalk_app.c|_twetalk_recv_event_cb(298): bot start speaking
INF|149071|twetalk_call.c|tc_twetalk_call_event_type_print(130): □ [01]
BOT_STOP_SPEAKING - 机器人停止说话
INF|149071|twetalk_app.c|_twetalk_recv_event_cb(303): bot stop speaking
```

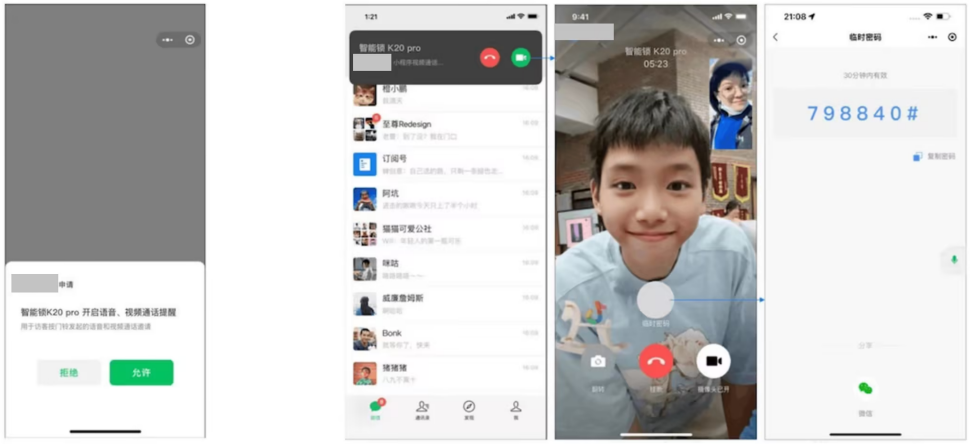
微信通话服务（TWeCall）

最近更新时间：2026-03-27 16:28:42

概述

物联网开发平台为用户提供基于微信小程序集成插件和设备端集成 SDK 的 TWeCall 接入能力，实现设备与小程序的双向通话，达到微信音视频通话的原生体验。设备端支持一键呼叫，微信端将持续响铃提醒。本文档将为您介绍 TWeCall 操作相关内容。

手机端示意图



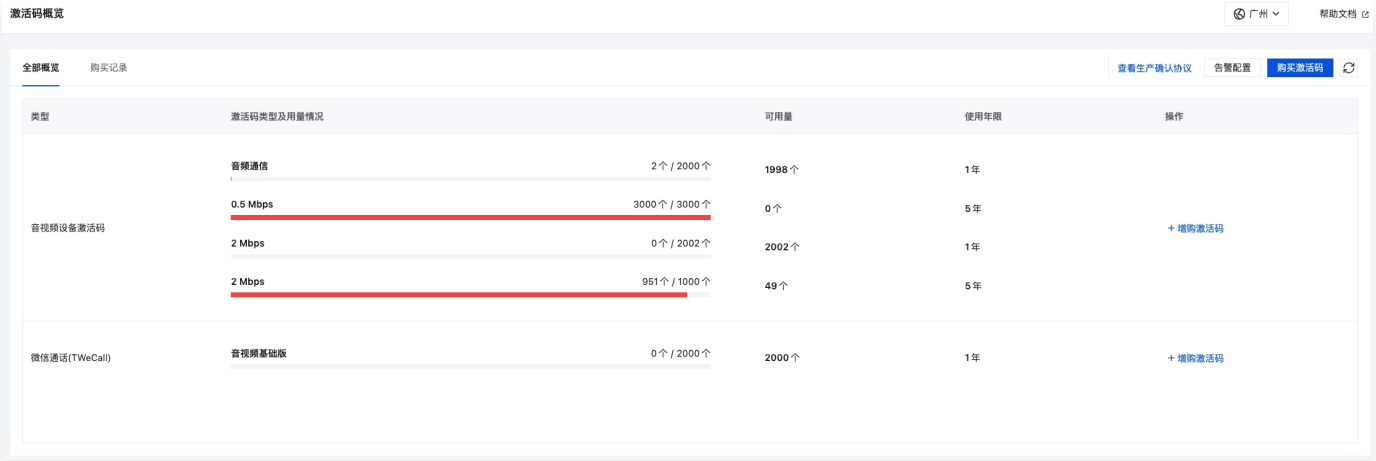
前提条件

已购买微信通话（TWeCall）激活码，如未购买，请先参考 计费说明 进行了解并购买。

操作步骤

查看 TWeCall 的激活码额度

1. 登录 物联网开发平台，进入**公共实例**。
2. 选择**用量统计** > **激活码概览**，在此页面，可查看当前账号的 TWeCall 激活码总数、已使用个数和待使用个数。

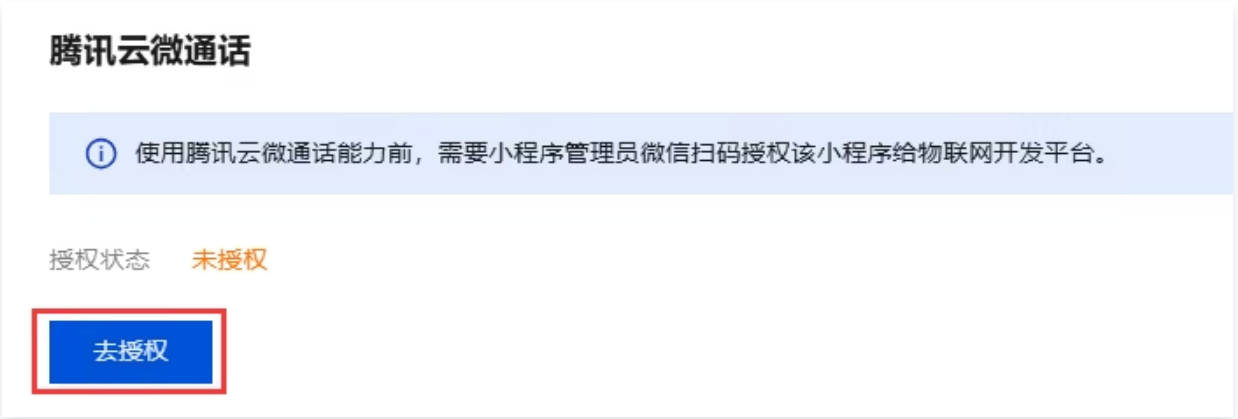


小程序授权

小程序需与物联网开发平台的应用进行关联，才能为音视频设备激活微信通话（TWeCall）的激活码。

1. 登录 物联网开发平台， 选择公共实例后，进入左侧导航栏的**应用**。
2. 单击**新建应用**，创建一个应用与小程序关联。

3. 创建完成后，单击应用名称进入应用详情。在应用详情页面的**腾讯云微通话**中，单击**去授权**。



4. 在小程序授权弹窗内，单击**开始授权**，系统将自动生成授权二维码。

5. 请小程序的管理员扫码完成授权。

激活服务

前提条件

待激活规格的微信通话服务（TWeCall）可用数量大于0。可前往控制台查看，操作指引请参考 前文 。

操作方式

1. 为音视频设备激活微信通话服务（TWeCall），请使用 激活 TWeCall 云 API。
2. 查询音视频设备的微信通话服务（TWeCall）剩余可用时长，请使用 查询 TWeCall 激活状态 云 API。