游戏多媒体引擎

进阶功能开发指南

腾讯云

文档目录

进阶功能开发指南
SDK 合规接入详细操作指引

最近更新时间：2025-03-28 17:56:32

此篇文档为您介绍 GME SDK 合规接入的详细操作。在阅读此文档前，请先充分阅读和了解《 SDK 接入使用说明 》。

> **说明：**
> HarmonyOS SDK 鸿蒙 SDK 现已发布，该版本已完成基础功能验证，现优先对部分企业级付费客户开放试用申请，如需使用，请 填写表单 联系您的商务。后续通过文档进行合规接入。

涉及到个人信息的接口

目前 GME SDK 涉及到个人信息的接口如下：

初始化接口

初始化接口用于初始化整个 GME SDK，启动后会收集个人信息。

> **注意：**
> 确保在用户阅读 App 隐私政策并取得用户授权之后，按APP功能需要在合适时机调用正式初始化函数 Init 初始化 SDK。反之，如果用户不同意《隐私政策》授权，则不能调用正式初始化函数。

接口原型

Android

```
public abstract int Init(String sdkAppId, String openId);
```

iOS

```
-(int)InitEngine:(NSString*)sdkAppID openID:(NSString*)openID;
```

示例代码

Android

```
String sdkAppID = "14000xxxxx";
String openID = "100";
```

```
int ret = 0;
//在用户同意APP隐私政策之后，按APP功能需要在合适时机再正式初始化SDK
//ret = 0，表示用户同意APP隐私合规政策
//ret = 1，表示用户不同意APP隐私合规政策
//如果用户不授权隐私策略，则 ret 修改为非 0
if(ret != 0){
    Log.e(TAG,"用户不同意APP隐私合规政策");
}else{
        ITMGContext.GetInstance(this).Init(sdkAppId, openId);
}
```

iOS

```
_openId = _userIdText.text;
_appId = _appIdText.text;
int result = 0;
//在用户同意APP隐私政策之后，按APP功能需要在合适时机再正式初始化SDK
//result = 0，表示用户同意APP隐私合规政策
//result = 1，表示用户不同意APP隐私合规政策
//如果用户不授权隐私策略，则 ret 修改为非 0
if (result == 0) {
    [[ITMGContext GetInstance] InitEngine:SDKAPPID openID:_openId];}
    else{
    log = [NSString stringWithFormat:@"用户不同意APP隐私合规政策"];
}
```

可延迟申请权限

请您注意，启动麦克风设备接口会需要用到麦克风权限，建议在需要启动麦克风的时候才对权限进行申请，在用户授权的情况下获取麦克风权限。需要麦克风权限的接口如下：

接口原型

Android

```
public abstract int EnableMic(boolean isEnabled);
public abstract int EnableAudioCaptureDevice(boolean isEnabled);
public abstract int StartRecording(String filePath);
public abstract int StartRecordingWithStreamingRecognition (String
filePath);
```

```
public abstract int StartRecordingWithStreamingRecognition (String
filePath,String language,String translatelanguage);
```

iOS

```
-(QAVResult)EnableMic:(BOOL)enable;
-(QAVResult)EnableAudioCaptureDevice:(BOOL)enabled;
-(int)StartRecording:(NSString*)filePath;
-(int)StartRecordingWithStreamingRecognition:(NSString *)filePath;
-(int)StartRecordingWithStreamingRecognition:(NSString *)filePath
language:(NSString*)speechLanguage translatelanguage:
(NSString*)translateLanguage;
```

检查权限

关于麦克风权限的检查，可以通过 GME TestMic 接口进行实现。

可选权限配置

请您注意，SDK 不强制获取可选权限，即使没有获取可选权限，SDK 基本功能也能正常运行。您可以配置可选权限，以便使用 SDK 提供的其他可选功能。建议调用请求前在合适的时机调用 SDK 提供的方法，在用户授权的情况下获取声明中的权限。

Android 配置可选权限

在 Android 平台中，读写权限不是必须添加的，请根据以下规则进行判断是否添加：

- 如果使用的是默认的日志路径（/sdcard/Android/data/xxx.xxx.xxx/files），即表示未对 SetLogPath 进行调用，则不需要 WRITE_EXTERNAL_STORAGE 权限。
- 如果调用 SetLogPath 接口将日志路径放在外部存储设备，以及使用语音消息功能在录制时的存储路径是在外部存储设备，则需要向用户申请 WRITE_EXTERNAL_STORAGE 权限，并得到用户明确批准。

iOS 配置可选权限

在 iOS 平台中，Required background modes 权限用于允许后台运行，可根据业务决定是否配置。

用户权利保障机制

终端用户撤销同意处理其个人信息的授权时，您可通过调用 Uninit 接口停止使用 SDK 功能并停止采集与关闭功能相应的用户数据。

接口原型

Android

```
public abstract int Uninit;
```

iOS

```
-(int)Uninit;
```

合规使用指南

最近更新时间：2025-10-20 10:09:31

为帮助使用游戏多媒体引擎 GME SDK 的开发运营者（以下简称“您”）在符合个人信息保护相关法律法规、政策及标准的规定下合规接入、使用第三方 SDK，游戏多媒体引擎 GME（以下简称"我们"）特制定《GME SDK 接入使用说明文档》（以下简称“文档”），便于您使用游戏多媒体引擎 GME SDK 过程中符合相应的合规要求。
请您在接入、使用游戏多媒体引擎 GME SDK 前，充分阅读和了解本文档内容。

一、接入/升级至满足监管新规的最新 SDK 版本

我们高度重视 SDK 的功能优化、个人信息安全和保护，将适时升级迭代 SDK 版本以提升产品的安全性和稳定性，确保符合相关法律法规及、监管及标准的最新合规要求。强烈建议您升级使用最新版本 SDK，以便保障您正常使用 SDK 最新功能、避免因您更新不及时产生的不利影响（例如 App 被通报或下架等）。
SDK 更新后，我们会及时通过官网公告通知或其他适当的方式提醒您更新的内容，以便您及时了解 SDK 最新版本信息。同时，您可以访问 SDK 最新版本 SDK 下载指引 。

二、App 隐私政策中应披露第三方 SDK 相关情况

请您确保您开发或运营的 APP 配备了符合监管要求的《隐私政策》文本。请您务必明确告知终端用户您的 App集成了第三方 SDK 服务。您应在《隐私政策》中添加关于本 SDK 收集使用个人信息的目的、方式和范围等，并显示本 SDK 的开发运营者名称及隐私政策链接。您应在 APP 登录注册页面及 APP 首次运行时，通过弹窗、文本链接及附件等简洁明显且易于访问的方式，应当以清晰易懂的语言告知用户《隐私政策》，由用户在充分知情的前提下，作出自愿明确的意思表示。
我们提供以下告知文案示例供您参考，您可以通过文字或表格方式向用户告知。请您理解 SDK 不同版本提供的功能服务及所需的字段信息可能会因开发者的选择或配置不同而存在差异，因此请您参考 SDK 隐私政策及您实际接入使用的 SDK 运行情况向用户进行充分告知并获得用户的同意。
仅 Android 参考示例：

```
第三方SDK名称：GME SDK
第三方SDK提供方的公司名称：腾讯云计算（北京）有限责任公司
使用目的及功能场景：当用户进行组队时，使用实时语音功能进行语音沟通交流
处理的个人信息类型：Wi-Fi 状态、设备型号、操作系统、IP 地址、录音
实现SDK产品功能所需的权限：录音（麦克风）、存储、蓝牙、网络连接、网络信息状态、Wi-Fi 状态、声音设置信息
第三方SDK隐私政策链接：https://cloud.tencent.com/document/product/607/59299
```

仅 iOS 参考示例：

```
第三方SDK名称：GME SDK
第三方SDK提供方的公司名称：腾讯云计算（北京）有限责任公司
```

使用目的及功能场景：当用户进行组队时，使用实时语音功能进行语音沟通交流
处理的个人信息类型：Wi-Fi 状态、设备型号、操作系统、IP 地址、录音
实现SDK产品功能所需的权限：Microphone（麦克风）
第三方SDK隐私政策链接：https://cloud.tencent.com/document/product/607/59299

仅 Harmony 参考示例：

第三方SDK名称：GME SDK
第三方SDK提供方的公司名称：腾讯云计算（北京）有限责任公司
使用目的及功能场景：当用户进行组队时，使用实时语音功能进行语音沟通交流
处理的个人信息类型：Wi-Fi 状态、设备型号、操作系统、IP 地址、录音
实现 SDK 产品功能所需的权限：录音权限、互联网访问权限、获取网络信息权限、修改音频设置权限、保持后台运行权限
第三方SDK隐私政策链接：https://cloud.tencent.com/document/product/607/59299

三、获得用户同意后再初始化 SDK

为满足法律法规及监管要求，您应确保在获得用户的同意后再初始化 SDK，并在用户触发 SDK 具体功能服务后通过配置 SDK 的相关参数完成发送请求的调用，此时 SDK 才会按照您设置的配置方式采集功能所需的个人信息或申请功能所需的权限。为了避免您在获取用户同意前，提前启动 SDK 收集使用用户个人信息，SDK 提供了延迟 SDK 初始化调用的 API 接口、合规初始化技术配置方案，点击此处查看详细操作指引 。

1. 确保在用户阅读 App 隐私政策并取得用户授权之后，按 App 功能需要在合适时机调用init接口初始化SDK。反之，如果用户不同意《隐私政策》授权，则不能调用正式初始化函数。该接口仅进行初始化，不会获取个人信息。
2. 请勿在用户同意隐私政策之前动态申请涉及用户个人信息的敏感设备权限；请勿在用户同意隐私政策前私自采集和上报个人信息（尤其注意 Android_ID、OAID、IMEI、MAC 地址、硬件序列号、应用安装列表等用户信息）。请勿在 App 处于未激活状态时（例如 App 在后台运行），请求 SDK 相关服务。

四、可选信息配置开关

SDK 无可选个人信息。

1、配置可选权限

请您注意，SDK 不强制获取可选权限，即使没有获取可选权限，SDK 提供的基本功能也能正常运行。您可以配置可选权限，以便使用 SDK 提供的其他可选功能。建议调用请求前在合适的时机调用 SDK 提供的方法，在用户授权的情况下获取声明中的权限。单击此处查看详细操作指引 。

操作系统	权限名称	使用目的	功能场景（申请时机）	是否可选

Android	android.permission.RECORD_AUDIO	用于采集用户声音，与其他使用者进行交互、录制声音。	打开麦克风时。	否
	android.permission.WRITE_EXTERNAL_STORAGE	存储 SDK 配置文件和日志文件。	初始化SDK时。	是
	android.permission.READ_EXTERNAL_STORAGE	读取 SDK 配置文件和日志文件。	初始化SDK时。	是
	android.permission.BLUETOOTH	需要支持蓝牙耳机和耳麦的接入。	初始化SDK时。	Android 12以下不可选，Android 12以上可选
	android.permission.READ_PHONE_STATE	SDK 需要监听电话的打断，在电话呼入时，停止音频的采集。	初始化SDK时。	否
	android.permission.INTERNET	访问网络连接。	初始化SDK时。	否
	android.permission.ACCESS_NETWORK_STATE	获取网络信息状态。	初始化SDK时。	是
	android.permission.ACCESS_WIFI_STATE	允许获取当前 Wi-Fi 接入的状态以及 WLAN 热点的信息。	初始化SDK时。	是
	android.permission.MODIFY_AUDIO_SETTINGS	修改声音设置信息。	初始化SDK时。	是
iOS	NSMicrophoneUsageDescription	使用音频通话功能，需要开启麦克风。	打开麦克风时。	否
MacOS	NSMicrophoneUsageDescription	使用音频通话功能，需要开启麦克风。	打开麦克风时。	否
HarmonyOS	ohos.permission.KEEP_BACKGROUND_RUNNING	切换到后台后仍可采集和播放。	初始化SDK时。	是

	ohos.permission.INTERNET	通过网络进行音视频数据传输。	初始化SDK时。	否
	ohos.permission.GET_NETWORK_INFO	获取网络状态。	初始化SDK时。	否
	ohos.permission.MODIFY_AUDIO_SETTINGS	修改系统音频设置。	初始化SDK时。	否
	ohos.permission.MICROPHONE	使用音频通话功能，需要开启麦克风。	使用音频通话功能，需要开启麦克风。	打开麦克风时

请注意，在不同设备和系统中，权限显示方式及关闭方式可能有所不同，请终端用户参考其使用的设备及操作系统开发方的说明或指引。当终端用户关闭权限即代表其取消了相应的授权，我们和开发者将无法继续收集和使用对应的个人信息，也无法为终端用户提供上述与该等授权所对应的功能。

2、配置可按照不同频次、精度收集个人信息

SDK 的数据采集仅在 App 调用/最终用户触发相关功能时触发，不涉及定时逻辑等频次控制选项。

3、指导建议

请您重点关注，在 APP 安装、运行和使用相关功能时，您应遵从国家相关法律法规、监管政策及标准的要求，收集用户个人信息或申请敏感权限，不得存在以下违规行为：
（1）未经用户同意不得收集任何个人信息。
（2）非服务所必需或无合理应用场景下，用户拒绝相关授权申请后，应用不得自动退出或关闭。
（3）在用户明确拒绝权限申请后，APP 不应向用户频繁弹窗或反复申请开启与当前服务场景无关的权限、影响用户正常使用，建议掌握合适时机申请敏感权限，不得影响其他功能可用。
（4）不得未明确告知用户索取权限的目的和用途。
（5）APP首次打开或运行中，未见使用权限对应的相关功能或服务时，不应提前向用户弹窗申请开启敏感权限。
（6）不得超出业务功能实际需要过度收集个人信息。

五、SDK扩展业务功能配置方式

本产品不涉及拓展业务功能。

六、用户权利保障机制

本 SDK 提供以下接口配置，以便您帮助终端用户实现个人信息主体权利请求。

1. 终端用户撤销同意处理其个人信息的授权时，您可通过调用 Uninit 接口停止使用 SDK 功能并停止采集与关闭功能相应的用户数据，点击此处查看接口使用的操作指导。
2. 如果您需要我们协助来实现您最终用户的其他个人信息主体权利请求，您可以通过“联系方式”来申请协助。

七、联系方式

我们设立了专门的个人信息保护团队和个人信息保护负责人，如果您和/或终端用户对本规则或个人信息保护相关事宜有任何疑问或投诉、建议时，可以通过以下方式与我们联系：

(i) 通过 腾讯客服（ https://kf.qq.com/ ）或 填写工单 与我们联系。

(ii) 将问题发送至 Dataprivacy@tencent.com。

(iii) 邮寄信件至：中国广东省深圳市南山区海天二路33号腾讯滨海大厦 数据隐私保护部(收)邮编：518054。

我们将尽快审核所涉问题，并在15个工作日或法律法规规定的期限内予以反馈。

八、注意事项

1. 您接入 GME SDK 前的合规自查

   为确保您就本 SDK 的使用获得终端用户的授权，且遵守个人信息保护要求和合规流程，我们建议您在接入 GME SDK 前进行合规自查。

   1）请仔细阅读并按本说明文档提示对您 App 的《隐私政策》进行合规自查。

   2）请务必做延迟初始化配置，确保获得用户同意后再初始化 SDK。

   3）当 GME SDK 基于最新的法律法规或监管要求进行更新后，请您在收到版本更新通知时及时将您 App 集成的 GME SDK 升级到最新版本。

   4）其他国家相关法律法规、监管政策及标准的要求。

2. 以下合规文件供开发者参考：

（1）《个人信息保护法》
（2）《工业和信息化部关于进一步提升移动互联网应用服务能力的通知》
（3）《工业和信息化部关于开展信息通信服务感知提升行动的通知》
（4）《工业和信息化部关于开展纵深推进APP侵害用户权益专项整治行动的通知》
（5）《工业和信息化部关于开展APP侵害用户权益专项整治工作的通知》
（6）《App违法违规收集使用个人信息行为认定方法》
（7）《网络安全标准实践指南—移动互联网应用程序(App)收集使用个人信息自评估指南》
（8）《常见类型移动互联网应用程序必要个人信息范围规定》
（9）《GB/T 35273-2020信息安全技术 个人信息安全规范》
（10）《网络安全标准实践指南—移动互联网应用程序（App）使用软件开发工具包（SDK）安全指引》
（11）《网络安全标准实践指南—移动互联网应用程序（App）系统权限申请使用指南》

服务端录制
全量录制

最近更新时间：2024-10-15 14:45:41

本文帮助您通过全量录制的方式快速接入**GME 服务端录制**功能。

使用场景

GME 对实时语音流提供**服务端录制**能力，帮助开发者实现内容留存、内容管理、内容再生产等场景。

- 全量录制：支持录制应用内全量语音房间按房间维度混流、按用户维度单流。
- 自定义录制：支持录制用户指定房间按房间维度混流、按用户维度单流。录制后的音频文件将存储在您账号下的**对象存储(COS)** 服务中。

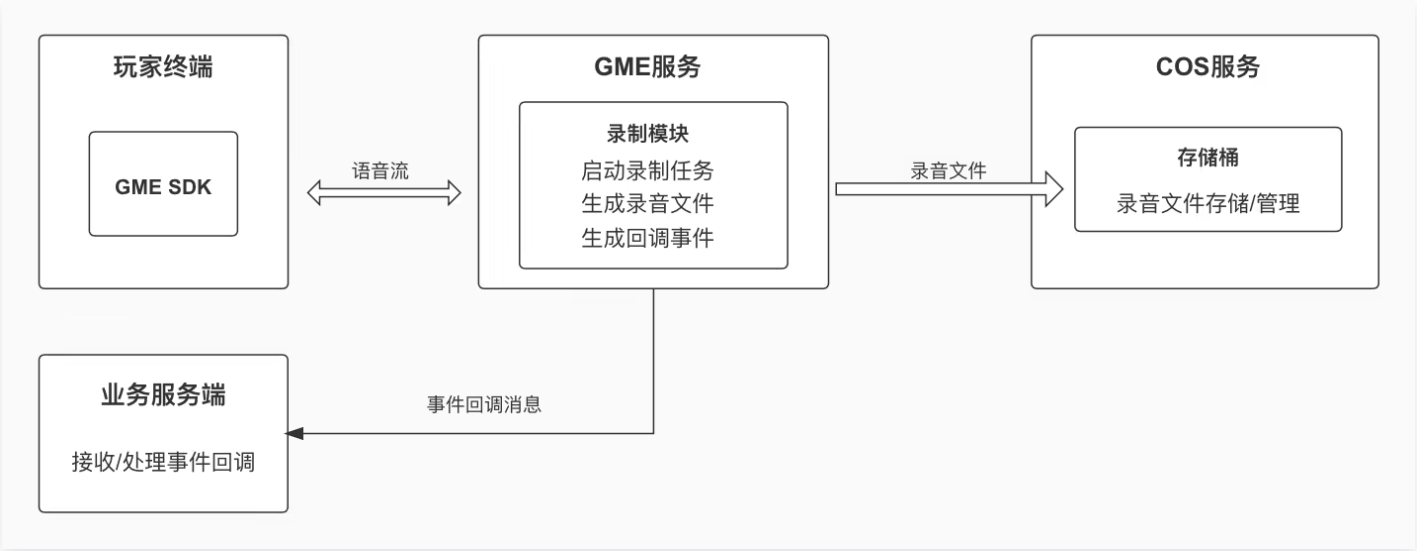
本文仅针对**全量录制**的开发接入方法进行说明。若您需要对应用开启自定义录制，请参见 自定义录制。

> ⚠ **注意：**
> - 使用 GME 服务端录制功能，录制过程将在 GME 产生录制服务费用。GME 录制服务将从2023年4月1日起正式计费，详细计费信息将提前公示在 GME 购买指南 。
> - 录制后的文件将存储在您的腾讯云账号下的**对象存储(COS)**服务中，将会根据您的存储量、存储时长、访问频次等具体使用方式产生**对象存储(COS)**账单。详细计费信息请参见 COS 计费说明 。

前提条件

- **已开通实时语音服务**：可参见 服务开通指引。
- **已开通服务端录制服务**：目前服务端录制功能针对白名单用户提供，请 提交工单申请 开通白名单。
- **已接入 GME SDK**：包括核心接口和实时语音接口的接入，详情可参见 Native SDK 快速接入 、Unity SDK 快速接入 、Unreal SDK 快速接入 。

服务架构



功能说明

录制范围

开启全量录制，将对所有实时语音房间进行录制。您可以配置仅录制房间混流，或仅录制用户单流，或同时录制单流和混流。

录制机制

机制项	描述
录制任务启动机制	第一个用户进房后触发录制任务启动
录制任务终止机制	最后一个用户退房后触发录制任务终止
录制任务分片机制	• 当单个音频文件时长达到两个小时，将自动进行音频文件分片 • 当用户关麦，用户单流录制音频将自动分片，用户开麦后启动新的分片 • 若录制中的任务异常中断，任务自动重连后将启动新的分片
录制任务事件通知机制	• 录制任务事件通过回调机制通知到您配置的回调地址。当发生这些事件时，您会收到回调通知：**录制开始、录制停止、录制文件上传完成** • 关于回调信息的详情，请参见 录制回调说明

存储位置

GME 服务端录制，录制完成后的音频文件将存储在您账号下的**对象存储 COS**服务中的指定存储桶。存储桶地域由您指定，可选的地域列表请参见 对象存储地域说明。

录制文件格式

文件格式以 `.mp3` 格式进行录制。

录制文件命名规则

- 用户单流录制文件：`bizid_roomid_userid/${任务开始时间}_${id}_audio.mp3`
- 房间混流录制文件：`bizid_roomid/${taskid}_${任务开始时间}_${id}_audio.mp3`

参数说明：

- **bizid:** GME 应用 ID。可在 GME 控制台 获取
- **roomid:** 语音房间 ID。是您在使用实时语音服务时定义并传入至 GME SDK
- **userid:** 玩家 ID。是您在使用实时语音服务时定义并传入至 GME SDK
- **taskid:** 录制任务 ID，由 GME 录制服务生成。每一个录制任务都具有一个独特的任务 ID
- **id:** 针对一个录制任务的分片的序列号。序列号初始序号为0

接入步骤

1. 在控制台完成录制服务配置
2. 接收录制任务回调（可选）
3. 查看/管理录制文件

步骤1：在控制台完成录制服务配置

1. 开通/关闭录制服务

登录 GME 控制台，进入**服务管理**菜单，对希望启用录制服务的应用单击**设置**，进入应用详情页。在页面中对**语音录制服务**单击 **修改** 。将录制开关置为 **开启** 。

首次开启录制服务时，GME 将会向您申请服务授权，以便于访问您的**对象存储 COS 服务**，您需要在弹窗页面需要同意授权后，才能正常开启服务端录制服务。

2. 录音文件存储配置

1. 进入应用详情页面，在页面中对**语音录制服务**单击 **修改** ，在 **录制文件存储桶** 单击 **绑定** 。
2. 在 **绑定存储桶**弹窗中，您可以绑定一个已有的 COS 存储桶（已有的存储桶需要您在 COS 控制台 先行创建），或新建一个存储桶。

3. 录制事件回调配置（可选）

如果您希望接收录制服务的事件回调，可以配置回调地址。操作路径：进入应用详情页面，在页面中对**语音录制服务**单击 **修改** ，在 **回调地址** 单击 **修改** ，在弹窗中输入接收回调的 URL 地址。目前仅针对录制任务完成状态推送事件回调消息。

4. 录制范围配置

录制范围可选自定义录制或全量录制。您此时应选中全量，并在复选框中指定是否录制单流、是否录制混流。完成上述配置后，单击 **保存**，录制服务即可开启。若您不再需要使用录制服务，请及时在控制台将录制服务开关置为 **关闭**， 避免产生预期之外的费用。

步骤2：接收录制任务回调（可选）

如果您在 步骤1 中配置了回调地址，则需要接收录制任务事件回调。

步骤3：查看/管理录制文件

一个录制任务结束后的数分钟内即可生成完成的音频文件。您需要登录您的 对象存储COS 控制台 对录制文件进行查看和管理。

自定义录制

最近更新时间：2025-03-27 18:05:02

本文帮助您通过自定义录制的方式接入**GME 服务端录制**功能。

使用场景

GME 对实时语音流提供**服务端录制**能力，帮助开发者实现内容留存、内容管理、内容再生产等场景。

- 全量录制：支持录制应用内全量语音房间按房间维度混流、按用户维度单流。
- 自定义录制：支持录制用户指定房间按房间维度混流、按用户维度单流。录制后的音频文件将存储在您账号下的**对象存储(COS)**服务中。

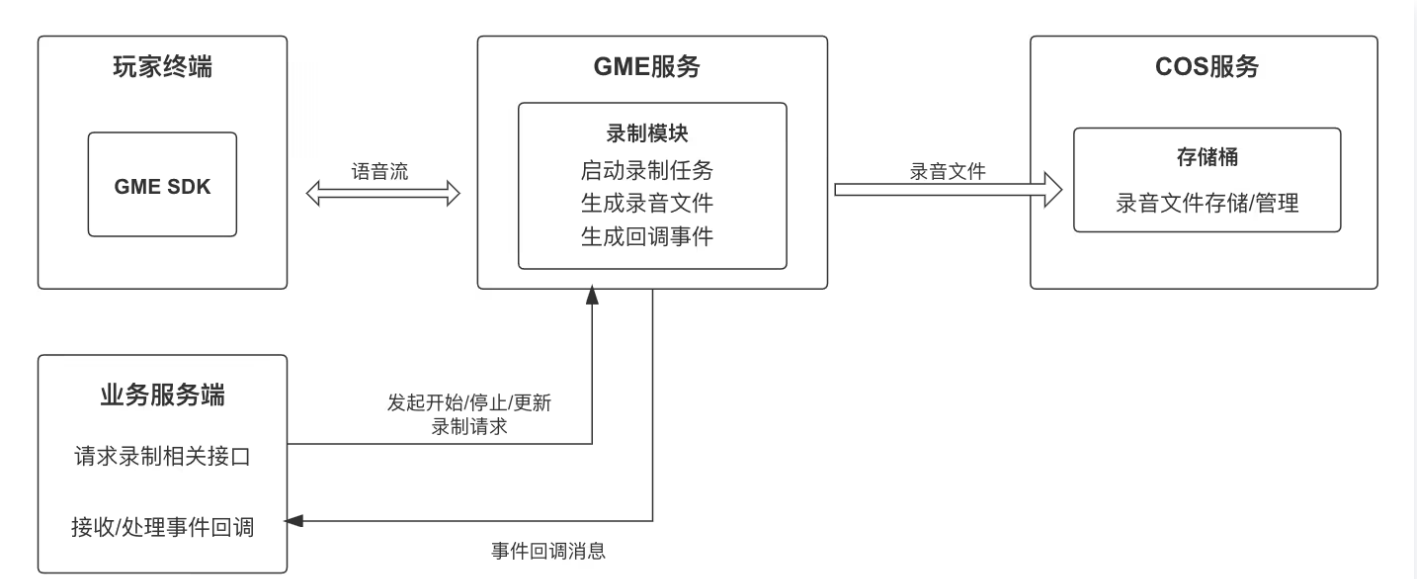
本文仅针对**自定义录制**的开发接入方法进行说明。若您需要对应用开启全量录制，请参见 开发指南-全量录制

> ⚠ **注意：**
>
> - 使用 GME 服务端录制功能，录制过程将在GME产生录制服务费用。GME 录制服务将从2023年4月1日起正式计费，计费详情请参见 GME 购买指南。
> - 录制后的文件将存储在您的腾讯云账号下的**对象存储(COS)**服务中，将会根据您的存储量、存储时长、访问频次等具体使用方式产生**对象存储(COS)**账单。详细计费信息请参见 COS 计费说明。

前提条件

- **已开通实时语音服务**：可参见 服务开通指引。
- **已开通服务端录制服务**：目前服务端录制功能针对白名单用户提供，请 提交工单申请 开通白名单。
- **已接入GME SDK**：包括核心接口和实时语音接口的接入，详情可参见 Native SDK 快速接入 、Unity SDK 快速接入 、Unreal SDK 快速接入 。

服务架构



功能说明

录制范围

您可通过服务端接口指定需要录制的房间 ID 混流或房间内用户单流。针对指定录制的房间ID，您可以通过服务端接口参数指定需要录制的玩家白名单，或无需录制的玩家黑名单。

相关接口

- StartRecord() **开始录制**：您可在这个接口参数中定义录制单流或混流、指定需要录制的 RoomID、指定订阅的白名单用户 ID 或黑名单用户 ID。
- StopRecord() **结束录制**：您可以通过这个接口，对某个taskid停止录制。若您没有记录taskid，则需要先通过 `DescribeTaskInfo()` 获取指定房间正在进行的录制任务 taskid。
- ModifyRecordInfo() **更新录制信息**：您可以通过这个接口，对某个 taskid 更新录制信息。包括更新录制类型、更新订阅的白名单用户 ID 或黑名单用户 ID。若您没有记录taskid，则需要先通过 `DescribeTaskInfo()` 获取指定房间正在进行的录制任务 taskid。
- DescribeTaskInfo() **查询房间录制信息**：您可以通过这个接口，查询指定房间的录制信息。包括进行中的录制任务 taskid、订阅的白名单/黑名单用户 ID。
- DescribeRecordInfo() **查询录制任务信息**：您可以通过这个接口，查询某个 taskid 的任务信息。包括录制类型、录制的房间 ID、录制的用户 ID、录制文件信息

录制机制

机制项	描述
录制任务启动机制	调用 StartRecord() 接口后，指定的房间录制任务将会启动。

录制任务终止机制	调用 StopRecord() 接口后，指定的房间录制任务将会结束。启动任务后，任务最长持续24小时，24小时之后自动终止录制，需要重启录制任务。
录音文件生成时机	• 房间混流录音文件：房间录制任务启动后，若房间已有用户，则混流音频文件立刻开始生成；若房间内无用户，则第一个用户进房后，混流音频文件立刻开始生成。 • 用户单流录音文件：房间录制任务启动后，在录制范围内的用户进房，则该用户的单流音频文件立刻开始生成。
录制任务分片机制	• 当单个音频文件时长达到两个小时，将自动进行音频文件分片。 • 当用户关麦，用户单流录制音频将自动分片，用户开麦后启动新的分片。 • 若录制中的任务异常中断，任务自动重连后将启动新的分片。
录制任务事件通知机制	• 录制任务事件通过回调机制通知到您配置的回调地址。当发生这些事件时，您会收到回调通知：**录制开始、录制停止、录制文件上传完成。** • 关于回调信息的详情，请参见 录制回调说明 。

存储位置

GME 服务端录制，录制完成后的音频文件将存储在您账号下的**对象存储 COS** 服务中的指定存储桶。存储桶地域由您指定，可选的地域列表请参见 对象存储地域说明。

录制文件格式

文件格式以 `.mp3` 格式进行录制。

录制文件命名规则

- **用户单流录制文件：** `bizid_roomid_userid/${任务开始时间}_audio.mp3`
- **房间混流录制文件：** `bizid_roomid/${taskid}_${任务开始时间}_audio.mp3`

> **参数说明：**
> - **bizid:** GME 应用 ID。可在 GME 控制台 获取。
> - **roomid:** 语音房间 ID。是您在使用实时语音服务时定义并传入至 GME SDK。
> - **userid:** 玩家 ID。是您在使用实时语音服务时定义并传入至 GME SDK。
> - **taskid:** 录制任务 ID，由 GME 录制服务生成。每一次成功调用 录制任务都具有一个独特的任务 ID。
> - **id:** 针对一个录制任务的分片的序列号。序列号初始序号为0。

接入步骤

1. 在控制台完成录制服务配置
2. 调用服务端接口，定义录制范围

3. 接收录制任务回调（可选）
4. 查看/管理录制文件

步骤1：在控制台完成录制服务配置

登录 GME 控制台，进入**服务管理**菜单，对希望启用录制服务的应用单击**设置**，进入应用详情页。

1. 开通/关闭录制服务

在页面中对**语音录制服务**单击 **修改** ，将录制开关置为 **开启** 。

首次开启录制服务时，GME 将会向您申请服务授权，以便于访问您的**对象存储 COS 服务**，您需要在弹窗页面需要同意授权后，才能正常开启服务端录制服务。

2. 录音文件存储配置

1. 在 **录制文件存储桶** 单击 **绑定** 。

2. 在 **绑定存储桶**弹窗中，您可以绑定一个已有的 COS 存储桶（已有的存储桶需要您在 COS 控制台 先行创建），或新建一个存储桶。

3. 录制事件回调配置（可选）

如果您希望接收录制服务的事件回调，可以配置回调地址。操作路径：进入应用详情页面，在页面中对**语音录制服务**单击 **修改**，在 **回调地址** 单击 **修改**，在弹窗中输入接收回调的 URL 地址。目前仅针对录制任务完成状态推送事件回调消息。

4. 录制范围配置

录制范围可选自定义录制或全量录制。您此时应选中自定义录制，否则调用服务端相关接口将会失败。完成上述配置后，单击 **保存**，录制服务即可开启。若您不再需要使用录制服务，请及时在控制台将录制服务开关置为 **关闭**， 避免产生预期之外的费用。

步骤2：接收录制任务回调（可选）

如果您在 步骤1 中配置了回调地址，则可以接收录制任务事件回调。

步骤3：调用服务端接口，定义录制范围

您需要根据业务场景实际需求来调用相关接口。在调用时序和流程的设计上，请注意：

- 仅支持对已存在的 RoomId 发起开始录制请求。若 RoomId 不存在，录制任务将创建失败。
- 若您没有主动调用停止录制，在房间内有用户的情况下，录制进程将一直保持。若房间内用户已全部退房，原录制任务进程将保持12小时。在这段保持的时间内，若用户重新进房，将自动开始录制。若已超过录制进程的保持时间，用户进房将不会自动开始录制。

步骤4：查看/管理录制文件

一个录制任务结束后的数分钟内即可生成完成的音频文件。您需要登录您的 对象存储 COS 控制台 对录制文件进行查看和管理。

回调说明

最近更新时间：2025-06-18 14:16:01

服务端录制回调说明

说明

受网络影响，您的服务器收到的通知顺序和事件发生的顺序可能不完全一致。
我们的服务有重试机制，但仍不能保证所有的消息都能到达。
考虑到以上两点，不建议您业务的核心业务逻辑依赖消息通知服务。

网络协议

如果在控制台配置了回调地址，即一个 HTTP(S) 协议接口的 URL，则需要支持 POST 方法，传输数据编码采用 UTF-8。

HTTP 头参数

名称	类型	是否必需	描述
Signature	string	是	签名，具体见下方 **签名生成** 说明

签名生成

Signature = HMAC-SH1 (strContent, SecretKey)

- **strContent**：签名原文串，为 body 的整个 JSON 内容（长度以 Content-Length 为准）。
- **body**：回调给业务的 JSON 内容，下方 回调示例 中的全部内容即为 body。
- **SecretKey**：密钥，为应用的权限密钥，可通过 控制台 > 应用详情 查看。
- **HMAC-SH1**：签名算法。

回调参数

名称	类型	描述
BizID	Integer	应用的 AppID，可通过 控制台 > 应用详情 查看。
RoomID	String	房间 ID
UserID	String	用户 ID
RecordMode	Integer	录制模式 • 0: 单流 • 1: 混流

Timestamp	Integer	发送回调时的时间戳（s）
TaskID	Integer	云录制服务分配的任务 ID。任务 ID 是对一次录制生命周期过程的唯一标识，结束录制时会失去意义。当您使用自定义录制模式时， 任务 ID 可以在开始录制时通过响应参数获取，需要业务保存下来，作为下次针对这个录制任务操作的请求参数。
EventType	Integer	事件类型
Detail	EventDetail	事件详情，由 EventType 决定格式

EventDetail 事件详情说明

EventType	说明	Detail
1	音频文件启动录制	• SeqNo: Number，分片序号 • FileName: String，文件名
2	音频文件完成录制	• SeqNo: Number，分片序号 • FileName: String，文件名
3	音频文件上传完成	• SeqNo: Number，分片序号 • FileName: String，文件名

回调示例

```
{
    "BizID":1400000000,
    "RoomID":"100",
    "UserID":"999",
    "TaskID":446946705284000000,
    "RecordMode":1,
    "Timestamp":1675930605,
    "EventType":1,
    "Detail":{
        "SeqNo":0,
        "FileName":"1400000000_100_999/2023-02-09-16-16-
45_446946705284000000_audio.mp3"
    }
```

```
}
```

客户端录制
K 歌录制

最近更新时间：2024-10-15 14:45:41

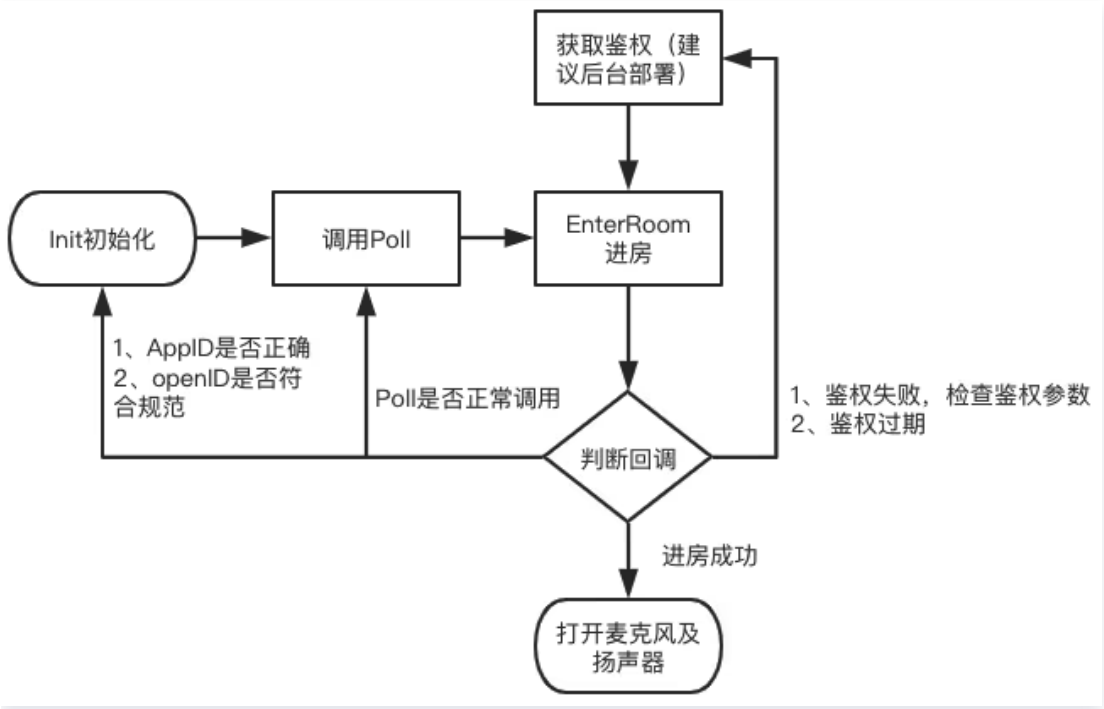
为方便开发者调试和接入腾讯云游戏多媒体引擎产品 API，这里向您介绍游戏多媒体引擎实时语音 K 歌功能的接入技术文档。

使用前提

- 如需使用实时语音 K 歌功能，需要在接入 GME SDK 且能在进行实时语音通话的情况下，才可以使用此功能。
- 进房时候需要填入房间类型参数，推荐使用 RoomType = 2 进入房间（请使用2或者3）。
- 使用中如果有错误码提示，可参考 错误码文档 进行解决。
- 技术方案及相关实践可以观看 零基础快速搭建 K 歌应用 。

流程图

进房流程参考图如下：



实时语音进房接口：

```
ITMGContext.GetInstance(this).Init(String.valueOf(mAppId), mUserId);//初始化sdk
ITMGContext.GetInstance(this).SetTMGDelegate(new MyDelegate());//设置代理类，用来接受各种回调和事件
```

```
EnginePollHelper.createEnginePollHelper();//周期性调用Poll函数，触发回调

byte[] authbuff = AuthBuffer.getInstance().genAuthBuffer(mAppId,
mRoomId, mUserId,mAppKey);//获得鉴权信息
ITMGContext.GetInstance(this).EnterRoom(mRoomId, 2, authbuff);//进入房间
```

> **说明**
> 详细调用流程和接口详情请参考 各端 SDK 接口文档 。

K 歌接口获取

1. 您需要在 下载指引 中下载标准 SDK 文件。
2. 根据不同的平台，下载 K 歌功能接口 并导入相应接口文件。

> **说明**
> 此功能支持 mp3、ogg 一共两种格式。

- 如果音乐文件为 ogg 格式，请 单击下载ogg动态库（iOS 端已包含 ogg 动态库，无需额外导入）。
- 如果音乐文件为 mp3 格式，请 单击下载mp3动态库（只有 iOS 平台才需导入此动态库，其他平台无需额外导入）。

Android 端配置

Android 对应的接口，已经包含在标准 jar 包内，不需要下载额外的接口文件。

iOS 端配置

1. 在 iOS 端使用 K 歌功能，需要将相关动态库引入工程中，单击下载 mp3 动态库 。
2. 将下载好的文件引入到工程文件中。并在 Link Binary With Libraries 中添加此动态库。
3. 将头文件 TMGEngine_adv.h 加入工程中，与其他 SDK 头文件同目录下。

Windows 端配置

Windows 端使用 K 歌功能，需要下载头文件后，将头文件 tmg_sdk_adv.h、tmg_type_adv.h 导入工程中。

Unity 引擎配置

在 Unity 引擎中使用 K 歌功能，需要下载头文件，将 Unity 文件夹下的代码文件 TMGEngine_Adv.cs、ITMGEngine_Adv.cs 拷贝后导入工程中。
如需导出 iOS 平台，请参考上文进行 mp3 动态库导入。

录制相关接口

开始录制

调用 StartRecord 接口开始录制。录制完成会有回调函数，需要监听 ITMG_MAIN_EVENT_TYPE_RECORD_COMPLETED。
录制时候请保证麦克风已经打开（设备及上行都需要打开），另外保证文件路径是可访问的，SDK 不会主动创建文件夹。

函数原型

```
int StartRecord(int type, String dstFile, String accMixFile, String accPlayFile)
```

参数	类型	意义
type	int	K歌场景下，此参数传 ITMG_AUDIO_RECORDING_KTV。如果是纯录制 MP3 文件，请使用 ITMG_AUDIO_RECORDING_SELF
dstFile	String	目标文件路径，用于保存录制完成的音乐。
accMixFile	String	一般为没有原声的伴奏，用于和人声合成音乐文件。
accPlayFile	String	用于播放的音乐文件，正常情况下与 accMixFile 为同一个文件。但在用户不熟悉歌曲时，可以填入带原唱的音乐文件路径，此时播放内容为带原唱的音乐，而合成为不带原声的伴奏。

示例代码

```
//Android
ITMGAudioRecordCtrl.GetInstance().StartRecord(ITMGAudioRecordCtrl.ITMG_AUDIO_RECORDING_KTV, dstFile, accMixFile, accPlayFile);
//iOS
#import "GMESDK/TMGEngine_adv.h"
[[ITMGAudioRecordCtrl GetInstance]StartPreview]
```

停止录制

调用 StopRecord 接口停止录制。

函数原型

```
int StopRecord()
```

暂停录制

调用 PauseRecord 接口暂停录制。

函数原型

```
int PauseRecord()
```

恢复录制

调用 ResumeRecord 接口恢复录制。

函数原型

```
int ResumeRecord()
```

录制回调

ITMG_MAIN_EVENT_TYPE_RECORD_COMPLETED 录制完成的回调。伴奏播放结束或者调用 StopRecord 触发此回调。

回调参数

参数	类型	意义
result	int	录制结果，0为成功。如果有其他错误码，可参考 错误码文档 进行解决。
filepath	String	目标文件的路径，StartRecord 传入的参数 dstFile。
duration	String	录制文件的长度，单位为毫秒。

设置播放文件

在调用 StartRecord 接口进行录制的时候，会设置播放的音乐文件。如果想要重新设置，可调用此接口重新设置播放文件。一般用于在原唱和纯音伴奏之间切换。

函数原型

```
int SetAccompanyFile(String accPlayFile)
```

参数	类型	意义
accPlayFile	String	用于播放的音乐文件。

获取伴奏时长

调用此参数获取伴奏文件 accMixFile 的时长，返回的时长单位为毫秒。

函数原型

```
int GetAccompanyTotalTimeByMs()
```

获取当前录制时长

调用此参数获取当前录制时长，返回的时长单位为毫秒。

函数原型

```
int GetRecordTimeByMs()
```

录制跳转

将录制时间跳转到指定时刻。如果参数比当前时间靠前，则重复的地方重新录制；相比当前时间靠后，则用静音数据填充没有录制的部分。

函数原型

```
int SetRecordTimeByMs(int timeMs)
```

参数	类型	意义
timeMs	int	跳转的时刻，单位为毫秒。

K 歌文件预览

获取录制文件的时长

调用此参数获取录制文件的时长。

函数原型

```
int GetRecordFileDurationByMs()
```

开始预览录制文件

调用此参数开始预览录制文件。

函数原型

```
int StartPreview()
```

停止预览录制文件

调用此参数停止预览录制文件。

函数原型

```
int StopPreview()
```

暂停预览录制文件

调用此参数暂停预览录制文件。

函数原型

```
int PausePreview()
```

恢复预览录制文件

调用此参数恢复预览录制文件。

函数原型

```
int ResumePreview()
```

设置当前预览的时间点

调用此参数设置当前预览的时间点。

函数原型

```
int SetPreviewTimeByMs(int time)
```

参数	类型	意义

time	int	预览文件的时间点，单位毫秒。

获取当前预览的时间点

调用此参数获取当前预览的时间点。

函数原型

```
int GetPreviewTimeByMs()
```

播放预览回调

ITMG_MAIN_EVENT_TYPE_RECORD_PREVIEW_COMPLETED 预览完成的回调。预览文件播放结束或者调用 StopPreview 接口触发

回调参数

参数	类型	意义
result	int	播放结果，0为成功。

文件合成接口

合并文件

调用此参数将录制好的人声和伴奏合并成一个文件。

函数原型

```
int MixRecordFile();
```

取消合并

调用此参数将取消合并操作。

函数原型

```
int CancelMixRecordFile();
```

合并文件回调

ITMG_MAIN_EVENT_TYPE_RECORD_MIX_COMPLETED 预览完成的回调。预览文件播放结束或者调用 StopPreview 接口触发。

回调参数

参数	类型	意义
result	int	合成结果，0为成功。
filepath	String	目标文件的路径，由 StartRecord 接口中传入的 dstFile。
duration	String	录制文件的长度，单位为毫秒。

高级设置

设置伴奏缩放

调用此接口设置人声和伴奏的缩放比例。录制完成后可进行调整。

函数原型

```
int SetMixWieghts(float mic, float acc)
```

参数	类型	意义
mic	float	人声的缩放比例，1.0为原来音量，小于1.0为缩小，大于1.0为放大，范围为0到2。
acc	float	伴奏的缩放比例，1.0为原来音量，小于1.0为缩小，大于1.0为放大，范围为0到2。

设置偏移量

调用此接口设置人声相对于伴奏的偏移，一般用于调整声音跟不上节拍的问题。录制完成后可进行调整。

函数原型

```
int AdjustAudioTimeByMs(int time)
```

参数	类型	意义
time	int	人声相对于伴奏的偏移时间，单位ms。大于0为向后移动，小于0为向前移动。

设置音效

设置音效类型，K 歌音效可参考 实时语音音效文档 。录制完成后可进行调整，录制时也可使用。

函数原型

```
int SetRecordKaraokeType(int type)
```

参数	类型	意义
type	int	此类型同等于实时语音音效中的K歌音效特效类型，具体可参考 K 歌音效特效 。

万人范围语音

最近更新时间：2025-11-24 16:14:52

功能简介

在一个语音房间内，用户可以与一定距离内的其他用户进行实时语音通话。此功能通过业务层调用 GME 客户端接口更新声源方位，目的是告诉服务器本端位置，通过**本端世界坐标+本端接收音频的范围**，与**其他端世界坐标 + 其他端接收音频的范围**进行判断后，服务器对玩家范围内音频流进行转发，默认转发距离玩家最近的20路音频流，**最高可支持上万人同时在房间内开麦**。

应用场景

大逃杀游戏	提供**大逃杀类型游戏、生存射击类手游**中特有的“仅小队”或“所有人”的语音模式。在游戏设置中，根据玩家选择： 1. 如果使用“仅小队”模式，则只能听见同队伍中队友说话的声音。 2. 如果使用“所有人”模式，则可以听见队伍中队友声音，以及一定范围内其他对局玩家说话的声音。
沉浸式虚拟场景	例如**游戏演唱会**等虚拟场景中，可以通过范围语音实现演唱者在虚拟房间内唱歌，台下所有听众可以听到歌声的同时，能与自身一定范围内其他听众进行交流，最高可支持同房间内上万人同时开启麦克风。

体验效果

可以到 Demo 体验 中下载体验程序，体验 3D 音效及范围语音的效果。

基本概念

使用范围语音功能，涉及到**语音模式**、**语音接收范围**以及 TeamID 这三个概念。

语音模式

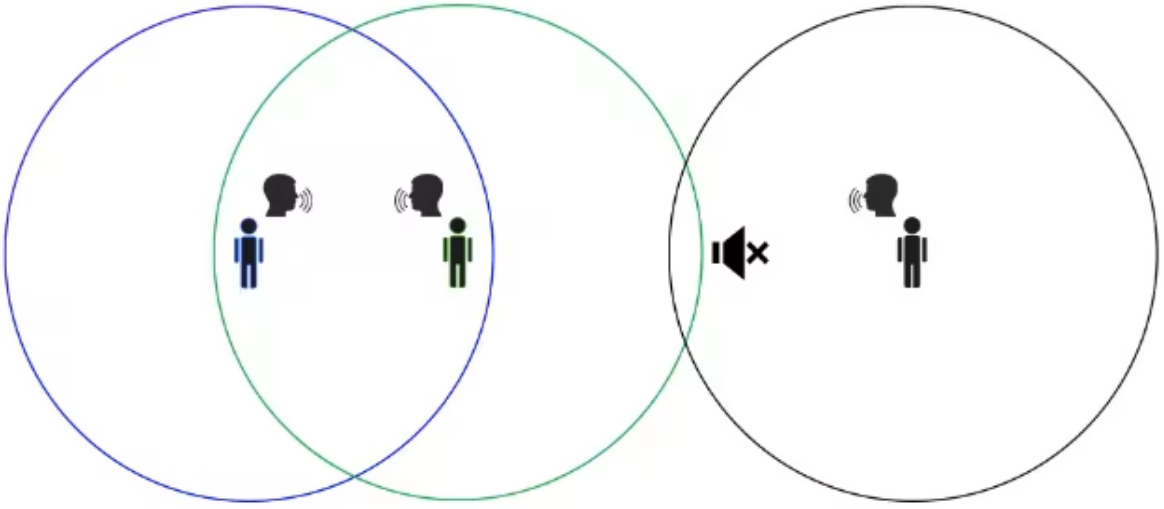
当进入范围语音房间时，有两种语音模式可供选择：

语音模式	参数名称	功能
所有人	RANGE_AUDIO_MODE_WORLD	• 设置后玩家附近一定范围的人都能听到该玩家讲话，如果范围内也有玩家设置为此模式，则也可以互相通话。 • 队友可以互相听到
仅小队	RANGE_AUDIO_MODE_TEAM	仅队友可以互相听到

> **注意：**
> 队友之间的通话不受距离以及语音模式的影响。

语音接收范围

RANGE_AUDIO_MODE_WORLD:



如果设置的语音模式为**所有人(RANGE_AUDIO_MODE_WORLD)**，此时的语音接收范围受 UpdateAudioRecvRange 接口影响。
假设以下两个玩家 A 与 B 为不同的小队，且设置的语音模式为**所有人(RANGE_AUDIO_MODE_WORLD)**：

玩家坐标	语音接收范围	与另一玩家声音可达情况	与队友声音可达情况
A（0,0,0）	10米	可以听到玩家 B 的声音，因为 B 玩家距离 A 玩家在10米内	不影响同一小队成员互相通话
B（0,8,0）	5米	不可以听到玩家 A 的声音，因为 A 玩家与 B 玩家距离超过5米	不影响同一小队成员互相通话

> **说明：**
> 具体玩家声音可达情况请参见 附录 。

TeamID

使用范围语音，需要调用 SetRangeAudioTeamID 接口设置小队号 TeamID，再调用 EnterRoom 接口进入语音房间。

- 当进入语音房间时指定的 TeamID != 0 时，进入范围语音房间模式。如果某成员使用 TeamID = 1 进入语音房间，当他设置语音模式为 RANGE_AUDIO_MODE_TEAM，则只有 TeamID = 1 的成员能听到他的声音；如果他设置的语音模式为 RANGE_AUDIO_MODE_WORLD，则除了 TeamID = 1 的成员，一定范围内的玩家也能听到他的声音。

TeamID 情况	语音模式	范围	声音可达情况
TeamID != 0，假设 TeamID = 1	RANGE_AUDIO_MODE_TEAM	10 米	声音只能和 TeamID = 1 的成员互通
	RANGE_AUDIO_MODE_WORLD	10 米	声音能和 TeamID = 1 的成员、以及语音模式设置为 RANGE_AUDIO_MODE_WORLD 的同房间10米范围内成员互通

- 如果某成员使用 TeamID = 0 进入语音房间，则为范围语音主持人模式，房间内所有人（不论语音模式是所有人还是仅小队）都可以听到该成员的声音。

TeamID 情况	TeamID 修改时机	范围	声音可达情况
TeamID = 0	进房前 TeamID != 0，进房后修改 TeamID = 0	10 米	• 说话声音全房间成员（不论语音模式是所有人还是仅小队）都能听到 • 能与 TeamID = 0 的成员互相沟通 • 能听到设置为 RANGE_AUDIO_MODE_WORLD 的同房间10米范围内成员声音
	进房前 TeamID = 0，以 TeamID = 0 进入房间	10 米	• 说话声音全房间成员（不论语音模式是所有人还是仅小队）都能听到 • 能与 TeamID = 0 的成员互相沟通 • 不能听到房间内其他人说话声音

示例场景

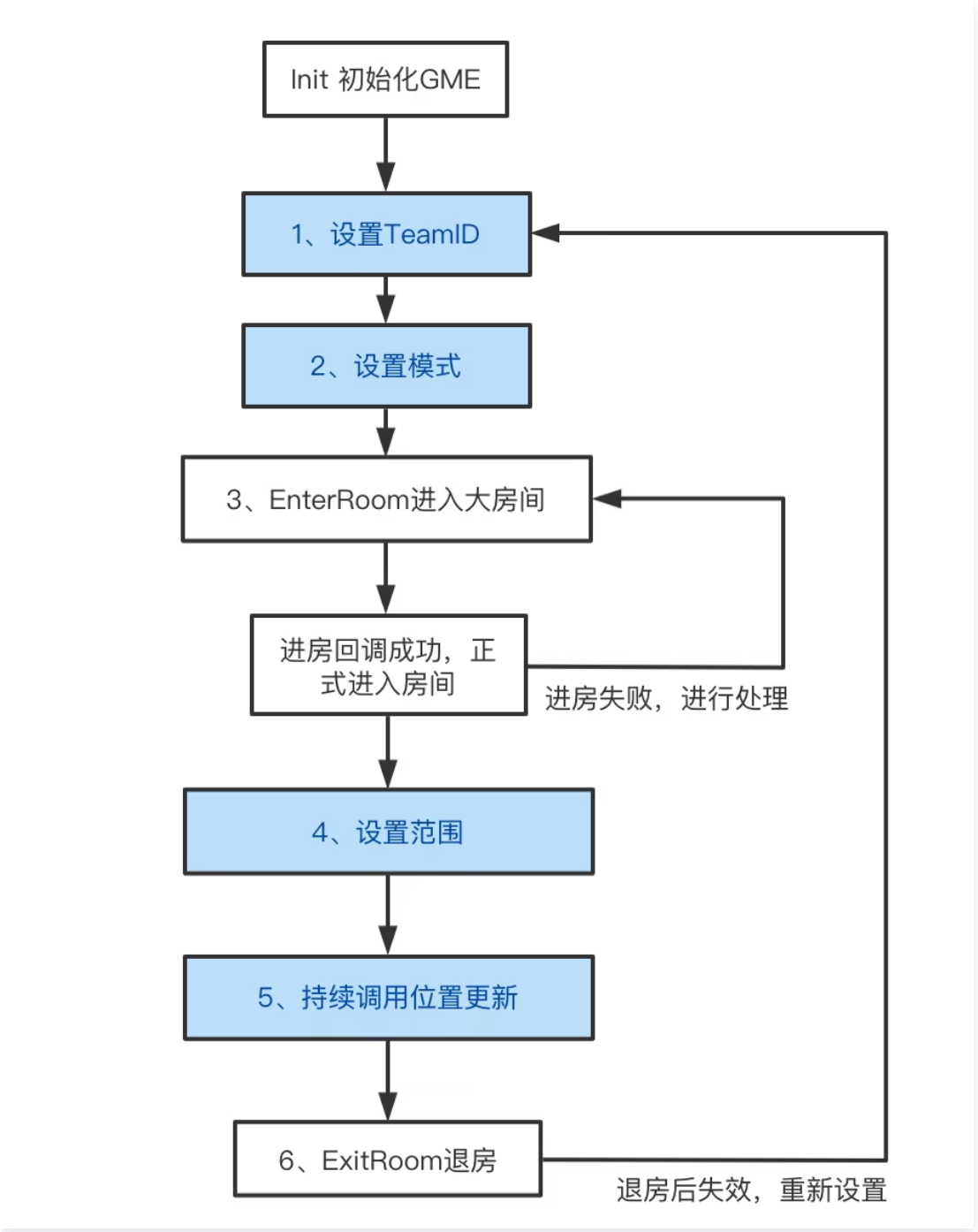
- **大逃杀游戏**：例如一个大逃杀类型的游戏，每4个人为一个队伍，则这4个人需要设置一个小队号 TeamID，每100人为一个对局房间，一个对局共25个小队，则25个小队都进去一个语音房间。在对局中，如果某玩家想和10米范围内的陌生人沟通，则将语音距离范围设置为10，将语音模式设置为 RANGE_AUDIO_MODE_WORLD，同时打开麦克风及扬声器。如果他只想和小队成员沟通，不和非小队的成员沟通，则只需要将语音模式设置为 RANGE_AUDIO_MODE_TEAM。

- **游戏演唱会**：在游戏中如果需要举办演唱会，歌手不需要和游戏玩家有互动，可以让游戏玩家通过 TeamID = OpenID 进入范围语音房间，设置语音模式为 RANGE_AUDIO_MODE_WORLD，依照游戏玩法设置语音距离范围，这样游戏玩家可以和附近玩家沟通交流；歌手将 TeamID 设置为 0 后进入房间，歌手的声音整个房间的人都能听见，但歌手听不见其他人的声音。
- **主持人模式**：在游戏中例如虚拟桌游场景，主持人说话声音即要房间内所有人听见，也要听见范围内玩家说话的声音，可以让主持人先以 TeamID != 0 的形式进入房间，进房后将 TeamID 设置为 0，此时主持人说话全房间的人都能听见，主持人也能听见范围内玩家说话的声音。

前提条件

- **已开通实时语音服务**：可参见 语音服务开通指引 。
- **已接入GME SDK**：包括核心接口和实时语音接口的接入，详情可参见 Native SDK 快速接入 、Unity SDK 快速接入 、Unreal SDK 快速接入 。
- **万人范围语音**：如果同房间内使用范围语音人数超过1千人，请 提交工单 联系 GME 开发者。

使用流程



注意：

- 请务必参照此流程调用接口。
- 流程图中蓝色部分为范围语音所需流程。
- 在进房成功后，调用 UpdateAudioRecvRange（至少一次），及每帧调用 UpdateSelfPosition。

1. 设置 TeamID

- 进房前，通过此接口设置队伍号，影响下一次进房。

- 进房后，可通过此接口修改队伍号，设置后立即生效。
- 退房后，TeamID 不会自动重置为0，所以一旦决定调用此语音模式，请在每次 EnterRoom 之前都调用此方法设置 TeamID。
- 退房后再进房，请在退房成功回调回来之后再调用设置队伍号接口。

函数原型

```
ITMGContext SetRangeAudioTeamID(int teamID)
```

参数	类型	意义
teamID	int	队伍号，专供范围语音模式中使用。当 TeamID 为0时，通话模式为普通小队语音，默认0。

2. 设置语音模式

- 进房前，通过调用此接口修改语音模式，影响下一次进房。
- 进房后，通过调用此接口修改语音模式，将直接改变当前用户的语音模式。
- 退房后，此参数不会自动重置为 MODE_WORLD，所以一旦决定调用此方法，请在每次 EnterRoom 之前都调用此方法设置语音模式。

函数原型

```
ITMGRoom int SetRangeAudioMode(RANGE_AUDIO_MODE rangeAudioMode)
```

参数	类型	意义
rangeAudioMode	int	0(MODE_WORLD) 代表“所有人”，1(MODE_TEAM) 代表"仅小队"

3. 进入语音房间

在调用 EnterRoom 之前，需调用以下两个 API：SetRangeAudioTeamID、SetRangeAudioMode。

函数原型

```
ITMGContext.GetInstance(this).EnterRoom(roomId,ITMG_ROOM_TYPE_FLUENCY,
authBuffer);
```

进入语音房间后，随后监听进房的回调并进行处理。

```
public void OnEvent(ITMGContext.ITMG_MAIN_EVENT_TYPE type, Intent data)
{
    if (ITMGContext.ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVENT_TYPE_ENTER_ROOM
== type)
        {
            //对事件返回的 Data 进行解析
            int nErrCode = data.getIntExtra("result" , -1);
            String strErrMsg = data.getStringExtra("error_info");

            if (nErrCode == AVError.AV_OK)
            {
                //收到进房信令，进房成功，可以操作设备
                ScrollView_ShowLog("EnterRoom success");
                Log.i(TAG,"EnterRoom success!");
            }
            else
            {
                //进房失败，需分析返回的错误信息
                ScrollView_ShowLog("EnterRoom fail :" + strErrMsg);
                Log.i(TAG,"EnterRoom fail!");
            }
        }
    }
```

在进房成功后，调用 UpdateAudioRecvRange（至少调用一次），及每帧调用 UpdateSelfPosition。

4. 设置接收语音距离范围

- 通过此方法用于设置接收的语音范围（距离以游戏引擎为准），只支持在**进房成功后调用**。
- 此方法必须配合 UpdateSelfPosition 更新声源方位联合使用。
- 此方法只需调用一次即可生效，支持修改。

函数原型

```
ITMGRoom int UpdateAudioRecvRange(int range)
```

参数	类型	意义
range	int	最大可以接收音频的范围，单位为引擎距离单位

示例代码

```
ITMGContext.GetInstance().GetRoom().UpdateAudioRecvRange(300);
```

5. 更新声源方位

更新声源方位，目的是告诉服务器本端位置，通过**本端世界坐标+本端接收音频的范围**，与**其他端世界坐标+其他端接收音频的范围**进行判断，达到范围语音效果。

- 此函数用于更新声源位置信息，只支持在**进房成功后调用**，且需要**每帧调用**，以 Unity 引擎为例，此接口需要在 Update 中调用。
- 使用范围语音一定要更新声源方位，如果不需要范围判断能力，**也需要进房后调用此接口一次**。
- 如果需要同时使用 3D 音效效果，此接口中的参数 axisForward、axisRight 及 axisUp 需要按照下文**范围语音结合 3D 音效**进行设置。

函数原型

```
public abstract int UpdateSelfPosition(int position[3], float
axisForward[3], float axisRight[3], float axisUp[3])
```

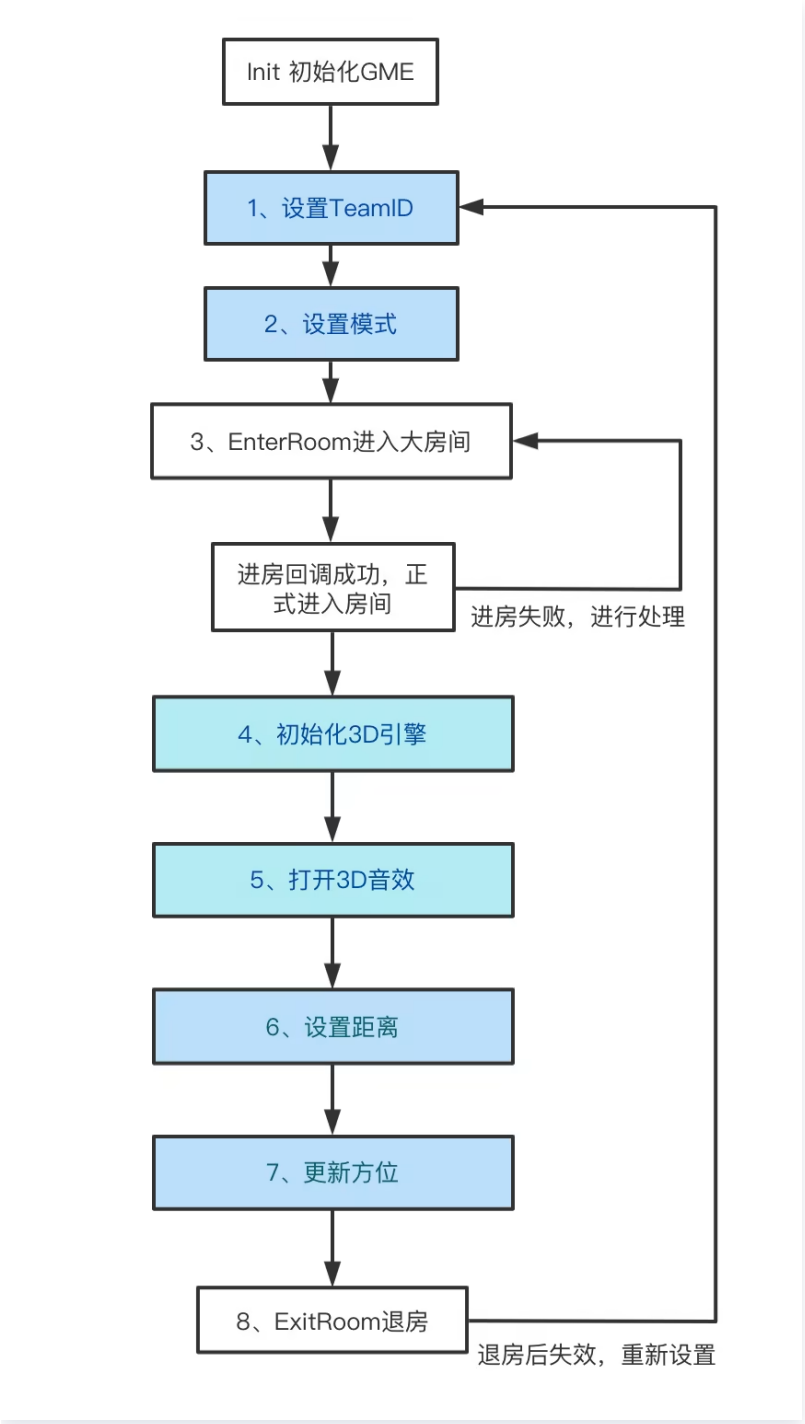
参数	类型	意义
position	int[]	自身在世界坐标系中的坐标，顺序是前、右、上
axisForward	float[]	在本产品中无需关注
axisRight	float[]	在本产品中无需关注
axisUp	float[]	在本产品中无需关注

范围语音结合 3D 音效

上文所介绍的范围语音功能，即通过距离来进行控制声音的可达性，如果需要更加沉浸式的体验，建议配合 3D 音效一起使用。

使用流程

使用范围语音的同时，如果需要同时使用 3D 音效功能，需要以下几个步骤，在进房后使用范围语音步骤1、2、3完成后，初始化 3D 引擎以及打开3D音效。



说明：
流程图中绿色部分为3D语音所需流程。

前提条件

参考 范围语音使用流程，完成步骤1、2、3。

1. 初始化 3D 音效引擎

此函数用于初始化 3D 音效引擎，在进房后调用。在使用 3D 音效之前必须先调用此接口，只接收 3D 音效而不发出 3D 音效的用户也需要调用此接口。

函数原型

```
public abstract int InitSpatializer(string modelPath)
```

参数	类型	意义
modelPath	string	填空即可

2. 开启或关闭 3D 音效

此函数用于开启或关闭 3D 音效。开启之后可以听到 3D 音效。

函数原型

```
public abstract int EnableSpatializer(bool enable, bool applyToTeam)
```

参数	类型	意义
enable	bool	开启之后可以听到 3D 音效
applyToTeam	bool	3D 语音是否作用于小队内部，仅 enable 为 true 时有效

通过 IsEnableSpatializer 接口获取 3D 音效状态。

3. 设置接收语音距离范围（3D）

- 通过此方法用于设置接收的语音范围（距离以游戏引擎为准），只支持在**进房成功后调用**。
- 此方法必须配合 UpdateSelfPosition 更新声源方位联合使用。
- 此方法只需调用一次即可生效。
- 3D 音效中，音源音量的大小与音源距离有一定的衰减关系。单位距离超过 range 之后，音量衰减到几乎为零。
- 距离与声音衰减的关系参考文档附录。

函数原型

```
ITMGRoom int UpdateAudioRecvRange(int range)
```

参数	类型	意义

range	int	最大可以接收音频的范围，单位为引擎距离单位

示例代码

```
ITMGContext.GetInstance().GetRoom().UpdateAudioRecvRange(300);
```

4. 更新声源方位（3D）

更新声源方位，目的是告诉服务器本端位置，通过本端世界坐标+本端接收音频的范围，与其他端世界坐标+其他端接收音频的范围进行判断，达到范围语音效果。

- **此函数用于更新声源位置信息，只支持在进房成功后调用，且需要每帧调用，以 Unity 引擎为例，此接口需要在 Update 中调用。**
- **请直接复制并调用示例代码使用此功能。**

函数原型

```
public abstract int UpdateSelfPosition(int position[3], float
axisForward[3], float axisRight[3], float axisUp[3])
```

参数	类型	意义
position	int[]	自身在世界坐标系中的坐标，顺序是前、右、上
axisForward	float[]	自身坐标系前轴的单位向量
axisRight	float[]	自身坐标系右轴的单位向量
axisUp	float[]	自身坐标系上轴的单位向量

示例代码

Unreal

```
FVector cameraLocation =
UGameplayStatics::GetPlayerCameraManager(GetWorld(), 0)-
>GetCameraLocation();
FRotator cameraRotation =
UGameplayStatics::GetPlayerCameraManager(GetWorld(), 0)-
>GetCameraRotation();
int position[] = {
```

```
    (int)cameraLocation.X,
    (int)cameraLocation.Y,
    (int)cameraLocation.Z };
FMatrix matrix = ((FRotationMatrix)cameraRotation);
float forward[] = {
    matrix.GetColumn(0).X,
    matrix.GetColumn(1).X,
    matrix.GetColumn(2).X };
float right[] = {
    matrix.GetColumn(0).Y,
    matrix.GetColumn(1).Y,
    matrix.GetColumn(2).Y };
float up[] = {
    matrix.GetColumn(0).Z,
    matrix.GetColumn(1).Z,
    matrix.GetColumn(2).Z};
ITMGContextGetInstance()->GetRoom()->UpdateSelfPosition(position,
forward, right, up);
```

Unity

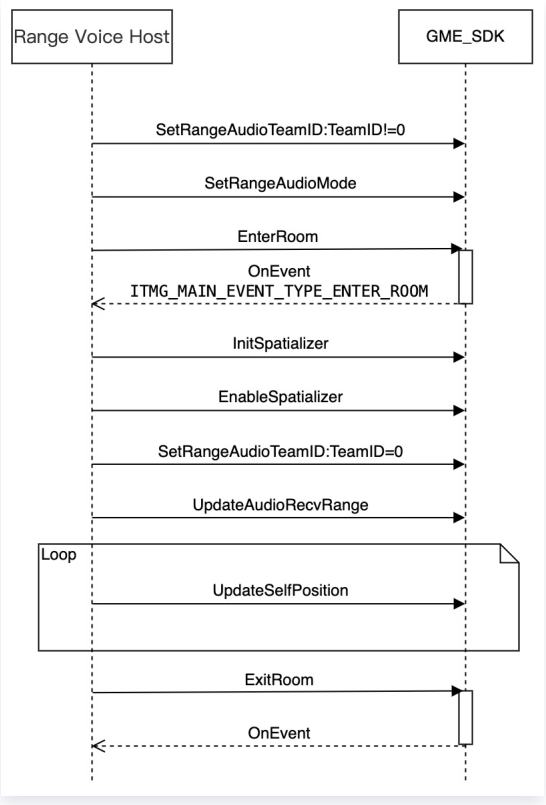
```
Transform selftrans = currentPlayer.gameObject.transform;
Matrix4x4 matrix = Matrix4x4.TRS(Vector3.zero, selftrans.rotation,
Vector3.one);
int[] position = new int[3] {
    selftrans.position.z,
    selftrans.position.x,
    selftrans.position.y};
float[] axisForward = new float[3] {
    matrix.m22,
    matrix.m02,
    matrix.m12 };
float[] axisRight = new float[3] {
    matrix.m20,
    matrix.m00,
    matrix.m10 };
float[] axisUp = new float[3] {
    matrix.m21,
    matrix.m01,
```

```
    matrix.m11 };
ITMGContext.GetInstance().GetRoom().UpdateSelfPosition(position,
axisForward, axisRight, axisUp);
```

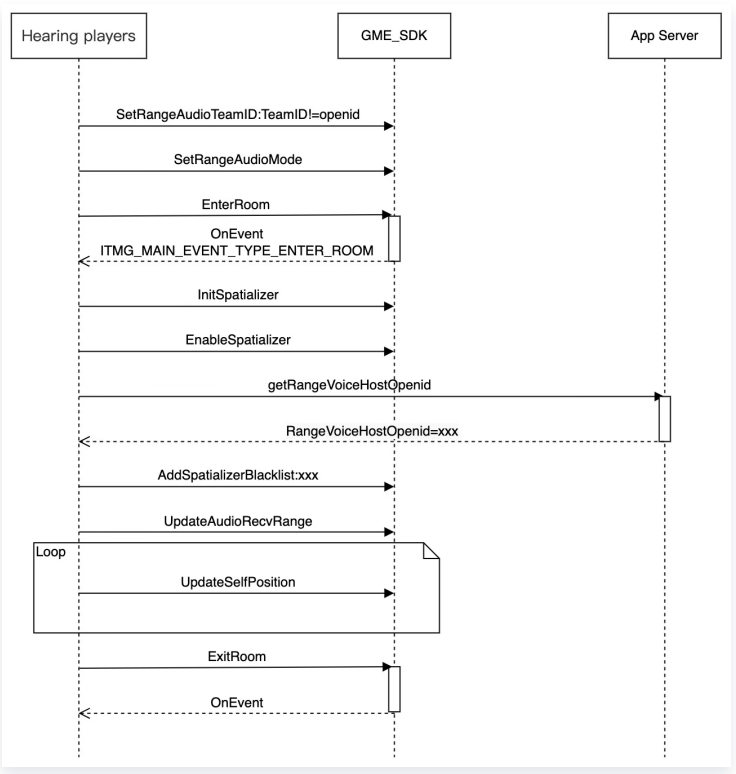
对主持人禁止使用 3D 音效

如果场景中有玩家使用范围语音主持人模式，即他的声音需要全房间玩家都能收听到，需要参考 3D 语音黑名单接口，在所有收听端将主持人加入到收听端的 3D 语音黑名单中，避免出现 3D 语音功能衰减效果对主持人声音可达性的影响。
不同角色的 API 调用时序如下：

主持人 API 时序



听众 API 时序



附录

不同语音模式

不同语音模式下，玩家声音可达情况：

- 假设 A 玩家状态为“所有人”，对应 B 玩家在不同语音模式下声音可达情况：

是否同一小队	是否范围内	语音模式	A与B 是否能相互听到对方的声音
同一小队	是	MODE_WORLD	是

		MODE_TEAM	是
	否	MODE_WORLD	是
		MODE_TEAM	是
不同小队	是	MODE_WORLD	是
		MODE_TEAM	否
	否	MODE_WORLD	否
		MODE_TEAM	否

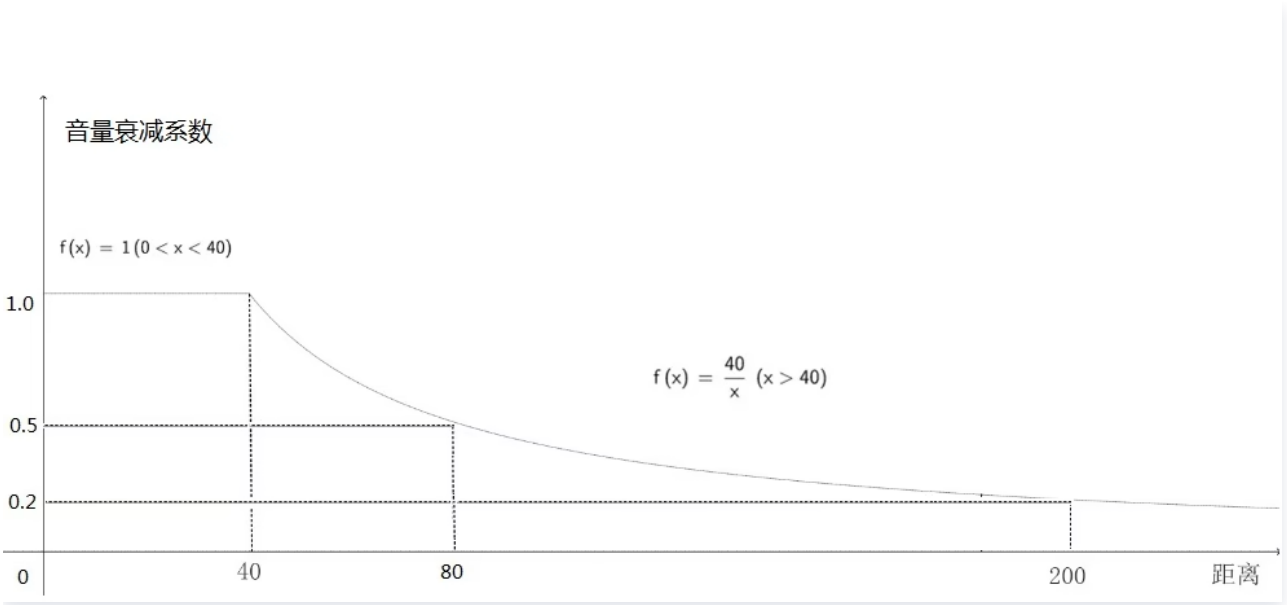
- 假设 A 玩家状态为“仅小队”，对应 B 玩家在不同语音模式下声音可达情况：

是否同一小队	是否范围内	语音状态	A与B 是否能相互听到对方的声音
同一小队	是	MODE_WORLD	是
		MODE_TEAM	是
	否	MODE_WORLD	是
		MODE_TEAM	是
不同小队	是	MODE_WORLD	否
		MODE_TEAM	否
	否	MODE_WORLD	否
		MODE_TEAM	否

距离与声音衰减的关系

3D 音效中，音源音量的大小与音源距离有一定的衰减关系。单位距离超过 range 之后，音量衰减到几乎为零。

距离范围（引擎单位）	衰减公式
0 < N < range/10	衰减系数：1.0（音量无衰减）
N ⩾ range/10	衰减系数：range/10/N


音量衰减系数
f(x) = 1 (0 < x < 40)
1.0
0.5
0.2
0
40
80
200
距离
f(x) = 40/x (x > 40)

3D 音效

最近更新时间：2025-11-24 16:14:52

为方便开发者调试和接入腾讯云游戏多媒体引擎产品 API，本文向您介绍游戏多媒体引擎 3D 音效的接入技术。

使用场景

在普通的进房实时语音中，玩家的声音不具有 3D 音效的效果，玩家之间只能进行很简单的互动；而引入 3D 位置语音之后，玩家在喊话过程中会暴露自己的方位和位置信息，玩家的声音也会根据位置变化而实时改变。可以说，3D 音效让《大逃杀》这种玩家间的沟通和战斗体验更真实，感受更加沉浸式、更加身临其境的吃鸡玩法。
点击 下载demo ，体验 3D 音效效果。

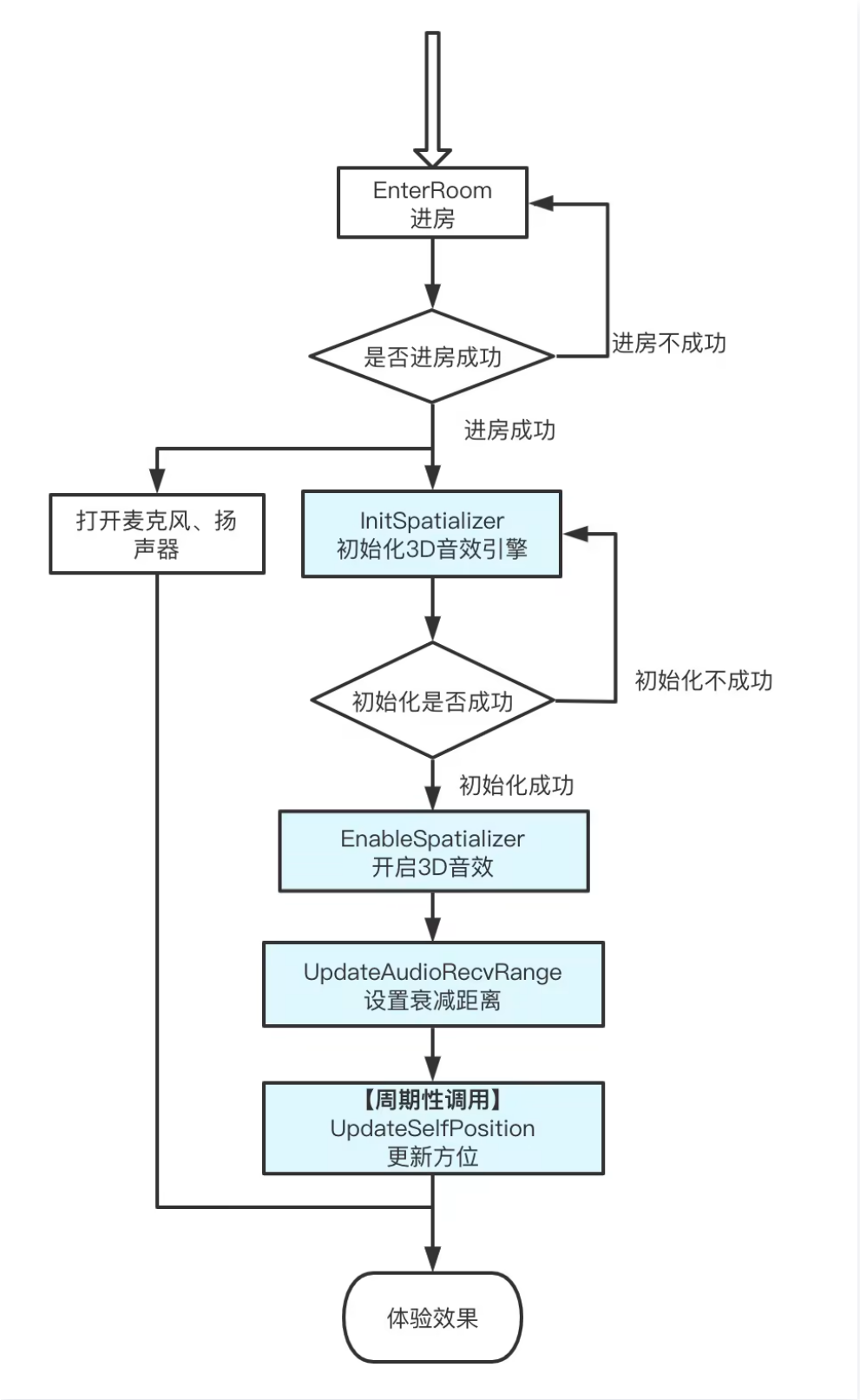
前提条件

- **已开通实时语音服务**：可参见 语音服务开通指引 。
- **已接入 GME SDK**：包括核心接口和实时语音接口的接入，详情可参见 Native SDK 快速接入 、Unity SDK 快速接入 、Unreal SDK 快速接入 。

实现流程

实现流程图

下图为 3D 音效实现的流程图，其中蓝色部分为对比普通进房实时语音需要接入的步骤。



初始化 3D 音效引擎

此函数用于初始化 3D 音效引擎，在进房后调用。在使用 3D 音效之前必须先调用此接口，只接收 3D 音效而不发出 3D 音效的用户也需要调用此接口。

函数原型

```
public abstract int InitSpatializer(string modelPath)
```

参数	类型	意义
modelPath	string	3D 音效资源文件的绝对路径

参数中的 3D 音效资源文件，需要另外下载到本地，根据接入 SDK 的版本进行区分：

- 如果是v2.8以下版本，请单击 下载，md5: d0b76aa64c46598788c2f35f5a8a8694。
- 如果是v2.8-v2.9.5版本，请单击 下载，md5: 3d4d04b3949e267e34ca809e8a0b9243。
- 如果是v2.9.6及以上版本，由于3D音效资源文件已内置，此处 modelPath 可填空。

SDK 的版本发布历史请参见 产品动态 。

> **关于资源路径**
>
> - 以 Unity 为例，建议将 3D 文件放在项目的 StreamingAssets 目录下，并参考 SampleCode 中 **copyFileFromAssetsToPersistent** 函数写法，将资源文件拷贝到不同平台的相应目录中。
> - 以 Unreal 为例，参考 SampleCode 中 **CopyAllAssetsToExternal** 函数写法，将 3D 模型文件拷贝后再读取路径。

开启或关闭 3D 音效

此函数用于开启或关闭 3D 音效。开启之后可以听到 3D 音效。

函数原型

```
public abstract int EnableSpatializer(bool enable, bool applyToTeam)
```

参数	类型	意义
enable	bool	开启之后可以听到 3D 音效
applyToTeam	bool	3D 语音是否作用于小队内部，仅 enable 为 true 时有效

获取当前 3D 音效状态

此函数用于获取当前 3D 音效状态。

函数原型

```
public abstract bool IsEnableSpatializer()
```

返回值	意义
true	开启状态
false	关闭状态

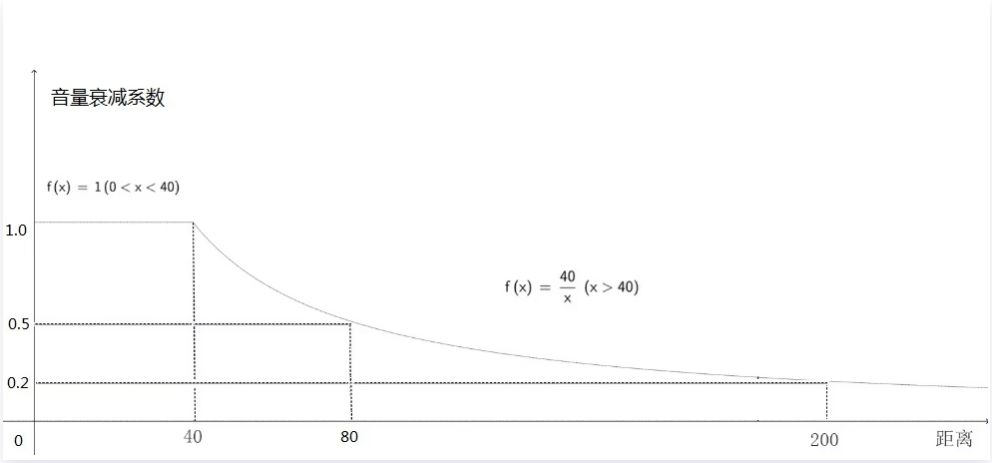
设置 3D 音效衰减距离

需要设置衰减距离，建议设置为 100。

距离与声音衰减的关系

3D 音效中，音源音量的大小与音源距离有一定的衰减关系。单位距离超过 range 之后，音量衰减到几乎为零。

距离范围（引擎单位）	衰减公式
0 < N < range/10	衰减系数：1.0（音量无衰减）
N ⩾ range/10	衰减系数：range/10/N



函数原型

```
public abstract void UpdateAudioRecvRange(int range)
```

参数	类型	意义
range	int	设定音效可接收的范围，建议设置100，此处的距离单位为游戏引擎中的距离单位

更新声源方位（包含朝向）

此函数用于更新声源方位角信息，每帧调用便可实现 3D 音效效果。

函数原型

```
public abstract int UpdateSelfPosition(int position[3], float
axisForward[3], float axisRight[3], float axisUp[3])
```

参数	类型	意义
position	int[]	自身在世界坐标系中的坐标，顺序是前、右、上
axisForward	float[]	自身坐标系前轴的单位向量
axisRight	float[]	自身坐标系右轴的单位向量
axisUp	float[]	自身坐标系上轴的单位向量

示例代码

Unreal

```
FVector cameraLocation =
UGameplayStatics::GetPlayerCameraManager(GetWorld(), 0)-
>GetCameraLocation();
FRotator cameraRotation =
UGameplayStatics::GetPlayerCameraManager(GetWorld(), 0)-
>GetCameraRotation();
int position[] = { (int)cameraLocation.X,(int)cameraLocation.Y,
(int)cameraLocation.Z };
FMatrix matrix = ((FRotationMatrix)cameraRotation);
float forward[] = {
matrix.GetColumn(0).X,matrix.GetColumn(1).X,matrix.GetColumn(2).X };
float right[] = {
matrix.GetColumn(0).Y,matrix.GetColumn(1).Y,matrix.GetColumn(2).Y };
float up[] = {
matrix.GetColumn(0).Z,matrix.GetColumn(1).Z,matrix.GetColumn(2).Z};
ITMGContextGetInstance()->GetRoom()->UpdateSelfPosition(position,
forward, right, up);
```

Unity

```
Transform selftrans = currentPlayer.gameObject.transform;
```

```
Matrix4x4 matrix = Matrix4x4.TRS(Vector3.zero, selftrans.rotation,
Vector3.one);
int[] position = new int[3] { selftrans.position.z,
selftrans.position.x, selftrans.position.y };
float[] axisForward = new float[3] { matrix.m22, matrix.m02, matrix.m12
};
float[] axisRight = new float[3] { matrix.m20, matrix.m00, matrix.m10 };
float[] axisUp = new float[3] { matrix.m21, matrix.m01, matrix.m11 };
ITMGContext.GetInstance().GetRoom().UpdateSelfPosition(position,
axisForward, axisRight, axisUp);
```

本地方位接口（VR 场景）

本接口适用于 VR 设备中，对 3D 位置变化有极高要求的场景。此功能为高级接口，需要在 GME 2.9.2 以上版本才可使用。

- 在一般 3D 的场景中，用户只需要通过函数 UpdateSelfPosition 更新自己的位置信息，然后经网络发送给其他用户。UpdateOtherPosition支持本地传入其他玩家的位置信息，不经过网络发送，适用于 VR 游戏场景。
- 为了避免和远程更新的坐标发生冲突，只要您调用了UpdateOtherPosition，远程坐标便会被丢弃，这种影响持续到再一次进房。所以，**您如果要通过本地方式更新玩家坐标，请更新所有玩家的坐标**。

函数原型

```
public abstract int UpdateOtherPosition(string openID, int position[3])
```

参数	类型	意义
openID	string	用户标识。
position	int[]	其他玩家在世界坐标系中的坐标，顺序是前、右、上

> **注意**
> 您如果要通过本地方式更新玩家坐标，请遍历**所有的玩家坐标**，通过此接口将坐标进行传入。

3D 语音黑名单接口

> **注意**
> 此接口在 GME 2.9.3 及以上版本 SDK 生效。

目前 3D 音效调用后对房间内所有人都会生效。由于某些特定场景下，不希望接受到的某个人声音会有因 3D 音效而产生的衰减，可以通过调用此接口，将某个人加入 3D 语音黑名单中，加入后此 openid 的声音不再有 3D 音效效果。

```
virtual int AddSpatializerBlacklist(const char* openId);
```

如果需要将此 openid 从黑名单中移除，需要调用以下接口：

```
virtual int RemoveSpatializerBlacklist(const char* openId);
```

如果需要清空黑名单，需要调用以下接口：

```
virtual int ClearSpatializerBlacklist();
```

问题排查

如果接入后测试语音，没有 3D 音效效果，排查路径如下：

1. 确定是否已经进房成功及开启麦克风？双方能否听到声音？
2. 是否适用双声道耳机进行收听？
3. InitSpatializer 接口返回值是否为0？
4. UpdateAudioRecvRange 设置是否过小？
5. 是否有周期性的调用 UpdateSelfPosition 接口？
6. 通过 错误码文档 进行判断并解决。

实时语音转文本

最近更新时间：2025-06-18 11:49:31

为方便开发者快速实现**实时语音流转文本**（以下用 **实时 ASR** 指代）功能，这里向您介绍实时语音转文本的使用场景以及接入流程。

使用场景

GME 语音转文本服务，可以将 GME SDK 采集的音频流实时识别成文字。该功能可应用于线上桌游场景，如 **“你说我猜”** 、**“你画我猜”** 等猜词游戏场景，玩家开始猜词后，调用 **GME SDK** 的 **实时 ASR** 接口，即可将玩家的实时语音流送入 **GME** 进行实时识别。业务客户端或业务服务端使用 **GME** 返回的语音转文本识别结果，可进行相关游戏玩法的逻辑实现。

> **说明：**
> 实时语音流转文本目前仅支持中文识别。

前提条件

- **已开通实时语音服务**：可参见 服务开通指引 。
- **已开通语音转文本服务**：可参见 服务开通指引 。
- **已接入 GME SDK**：包括核心接口和实时语音接口的接入，详情可参见 Native SDK 快速接入 、Unity SDK 快速接入 、Unreal SDK 快速接入 。

> **说明：**
> 实时语音流转文本功能将产生实时语音服务费用、语音转文本服务费用。详情请参见 计费文档 。

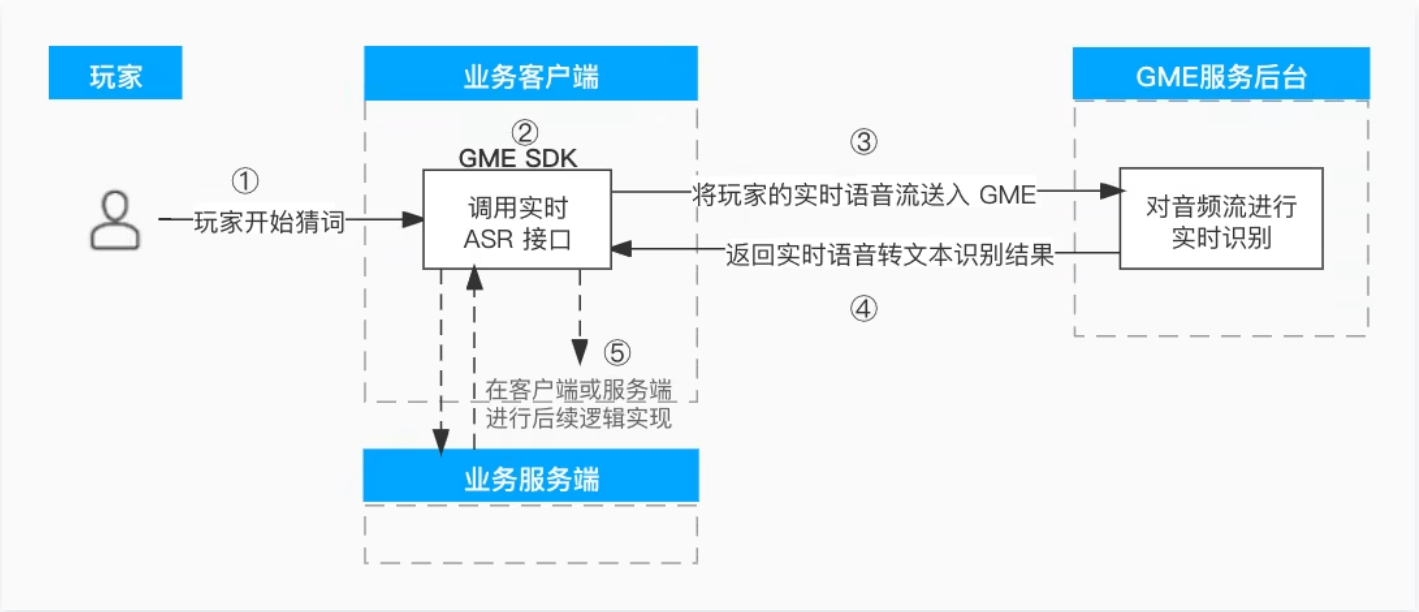
使用限制

实时语音流转文本默认单账号限制并发连接数为50路，如您有提高并发限制的需求， 请提工单 进行咨询。

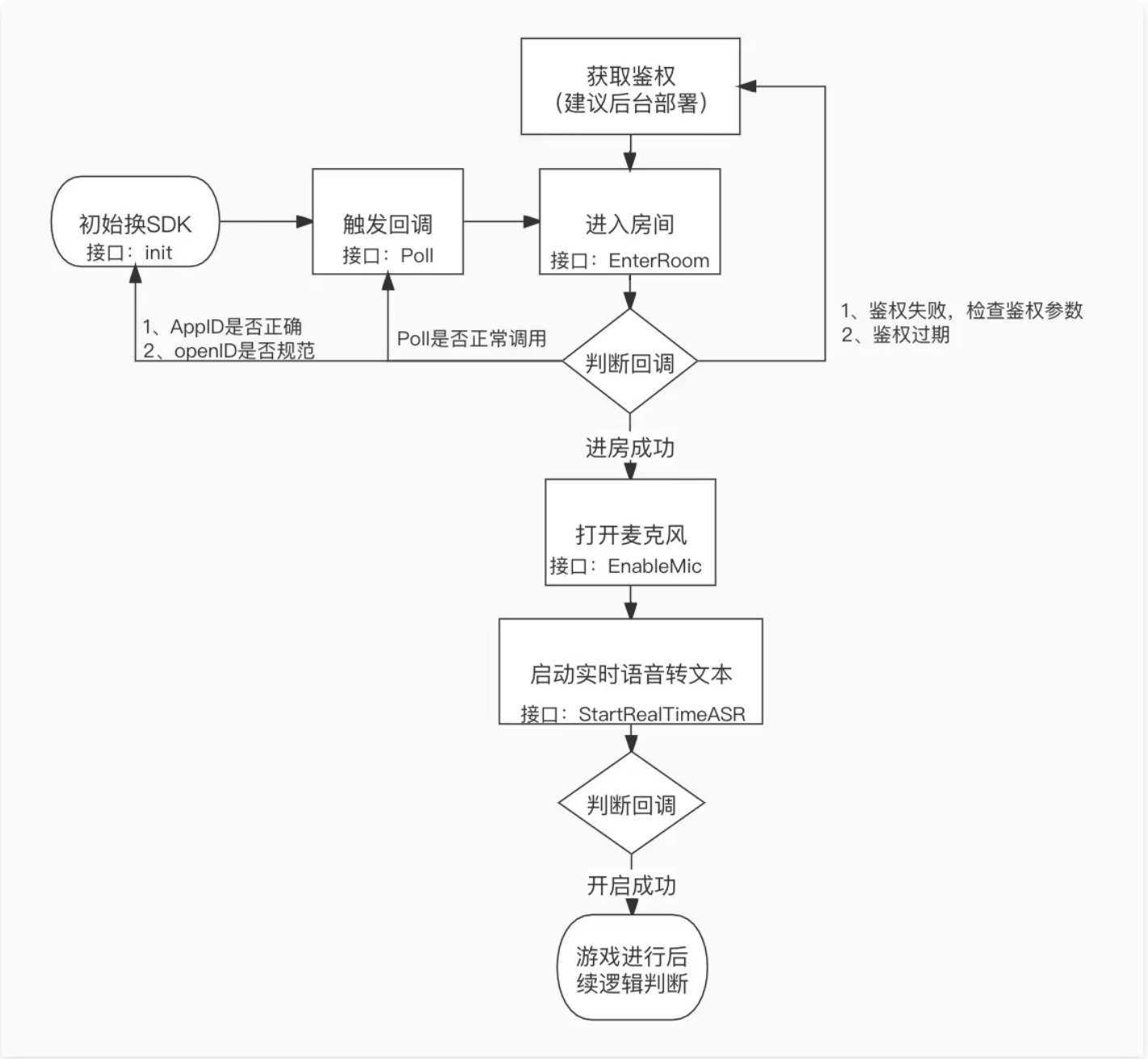
实现流程

整体流程

实时语音流转文本整体流程如下：



接入流程



接入步骤

1. 启动实时语音流转文本
2. 实时语音流转文本回调
3. 业务进行后续逻辑判断
4. 停止实时语音流转文本

步骤1： 启动实时语音流转文本

在成功进房之后，开启麦克风采集及上行的情况下，调用 StartRealTimeASR 接口可以启动实时语音流转文本功能。

iOS

```
//启动实时语音流转文本
ret = (int)[[ITMGContext GetInstance] StartRealTimeASR];
```

Android

```
//启动实时语音流转文本
nRet = ITMGContext.GetInstance(getActivity()).StartRealTimeASR();
```

步骤2： 实时语音流转文本回调

启动实时语音流转文本功能之后，会在回调 ITMG_MAIN_EVENT_TYPE_REALTIME_ASR 中返回对应的信息。

字段	含义
sub_type	• ITMG_REALTIME_ASR_START：调用 StartRealTimeASR 接口返回值为0 • ITMG_REALTIME_ASR_CONTENT：转文本内容，可根据不同的slice_type获取不同状态的转文本结果 • ITMG_REALTIME_ASR_END：调用StopRealTimeASR 函数接口返回值为0，或者在成功启动实时语音转文本，如果中间发生错误，例如服务器返回异常，长时间断网也会返回此回调
index	前一段话的结果在整个音频流中的序号，从0开始逐句递增
result	执行结果，如果为 0 则表示成功
slice_type	转文本状态，业务层可根据需求关注不同状态的返回文本： • 0：一段话开始转文本 • 1：一段话转文本中，voice_text_str 为非稳态结果（结果还可能变化） • 2：一段话转文本结束，voice_text_str 为稳态结果（结果不再变化） 识别过程中可能返回的 slice_type 序列可能有： • 0-1-2：一段话开始识别（slice_type=0）、识别中（可能有多次slice_type=1返回）、识别结束（slice_type=2） • 0-2：一段话开始识别（slice_type=0）、识别结束（slice_type=2） • 2：直接返回一段话完整的转文本结果（slice_type=2）
voice_text_str	转文本内容

回调示例代码：

iOS

```
-(void)OnEvent:(ITMG_MAIN_EVENT_TYPE)eventType data:(NSDictionary
*)data{
    NSString* log =[NSString stringWithFormat:@"OnEvent:%d,data:%@",
(int)eventType, data];
    [self showLog:log];
    NSLog(@"====%@====",log);
    switch (eventType) {
            case ITMG_MAIN_EVENT_TYPE_REALTIME_ASR :{
            if (ITMG_REALTIME_ASR_START == ((NSNumber
*)data[@"sub_type"]).intValue) {
                UIAlertView *alert = [[UIAlertView alloc]
initWithTitle:@"实时语音流转文本" message:[NSString stringWithFormat:@"开始
结果:%@ 错误信息:%@",data[@"result"],data[@"error_info"]] delegate:NULL
cancelButtonTitle:@"OK" otherButtonTitles:nil];
                [alert show];
                if (((NSNumber *)data[@"result"]).intValue == 0) {
                    isRealTimeASRRunnig = true;
                    asrResult = @"";
                }
            }
            else if (ITMG_REALTIME_ASR_CONTENT == ((NSNumber
*)data[@"sub_type"]).intValue){

                asrResult = [NSString stringWithFormat:@"index:%@ \n
slice_type:%@ \n
voice_text_str:%@",data[@"index"],data[@"slice_type"],data[@"voice_text_
str"]];
                NSString *totalString = [NSString stringWithFormat:@" \n
\n ASR: \n %@  \n sendType:%d \n sendList:%@ \n recvType:%d
\nrecvlist:%@ \n
%@",asrResult,sendType,sendArray,recvType,recvArray,tips];
                [RunningTipsView Show:totalString
parentView:_tipsScrollView];
            } else if (ITMG_REALTIME_ASR_END == ((NSNumber
*)data[@"sub_type"]).intValue){
```

```
                UIAlertView *alert = [[UIAlertView alloc]
initWithTitle:@"实时语音流转文本" message:[NSString stringWithFormat:@"结束
结果:%@ 错误信息:%@",data[@"result"],data[@"error_info"]] delegate:NULL
cancelButtonTitle:@"OK" otherButtonTitles:nil];
                [alert show];
                isRealTimeASRRunnig = false;
                asrResult = @"";
            } else {
                UIAlertView *alert = [[UIAlertView alloc]
initWithTitle:@"实时语音流转文本" message:[NSString stringWithFormat:@"未知
操作 结果:%@ 错误信息:%@",data[@"result"],data[@"error_info"]]
delegate:NULL cancelButtonTitle:@"OK" otherButtonTitles:nil];
                [alert show];
            }
```

```
    }
 }
```

```
 }
```

Android

```
...
//判断回调
if(ITMGContext.ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVENT_TYPE_REALTIME_ASR==
type) {
    int code = data.getIntExtra("result", 0);
    int sub_type = data.getIntExtra("sub_event", 0);

    if (sub_type == ITMGContext.ITMG_REALTIME_ASR_CONTENT){

        String totaString = " \n \n ASR: \n ";
        totaString +=  data.getIntExtra("index", -1);
        totaString += " \n slice_type:";
        totaString +=  data.getIntExtra("slice_type", -1);
        totaString += " \n voice_text_str";
        totaString += data.getStringExtra("voice_text_str");
        totaString += "\nAudioRoute :" + "\n";
```

```
    }

    String errorInfo =  data.getStringExtra("error_info");
    String typeInfo ;
    if (sub_type == ITMGContext.ITMG_REALTIME_ASR_START) {
        typeInfo = "开始 结果:";
        if (code ==0) {
            isRealTimeASRRunning = true;
        }
    } else  if(sub_type == ITMGContext.ITMG_REALTIME_ASR_END) {
        typeInfo = "结束 结果:";
        isRealTimeASRRunning = false;
    } else {
        typeInfo = "操作unknown 结果:";
    }
    Toast.makeText(getActivity(), String.format("%s %d: 错误信息: %s
",typeInfo,code,errorInfo), Toast.LENGTH_SHORT).show();
}
```

步骤3： 业务进行后续逻辑判断

业务客户端在收到 GME 返回的实时语音识别结果后，即可在客户端或服务端根据玩法逻辑进行后续判断。

步骤4： 停止实时语音转文本

通过 StopRealTimeASR 接口可以停止实时语音转文本。
示例代码：

iOS

```
//停止实时语音ASR
ret = (int)[[ITMGContext GetInstance] StopRealTimeASR];
```

Android

```
//停止实时语音ASR
int nRet = ITMGContext.GetInstance(getActivity()).StopRealTimeASR();
```

实践教程

在一局“你说我猜”游戏中，将所有玩家加入房间。以每一轮新词竞猜作为一个实时语音转文本的会话周期，即：在每一次更换新词时调用“启动实时语音转文本”接口；在当前词语竞猜时间截止时，或有玩家猜中时调用“停止实时语音转文本”接口。

房间管理功能

最近更新时间：2024-10-15 14:45:41

为方便开发者快速接入房间管理服务，这里向您介绍房间管理服务的使用场景以及接入流程。

功能简介

通过客户端房间管理接口，可以简单实现对房间内成员的管理、以及对房间内成员上下麦的管理。

使用场景

例如在狼人杀场景中，作为主持人可以通过 EnableMic 控制其他玩家打开麦克风发言；当某位玩家已经“死亡”，不需要听房间内声音或者操作麦克风说话，则通过 ForbidUserOperation 接口禁止该玩家操作设备。

前提条件

- **已开通实时语音服务**：可参见 语音服务开通指引 。
- **已接入GME SDK**：包括核心接口和实时语音接口的接入，详情可参见 Native SDK 快速接入 、Unity SDK 快速接入 、Unreal SDK 快速接入 。

> **说明：**
> 此功能不支持 H5 SDK。

接入流程

类名：ITMGRoomManager

GME 房间管理功能在进房后才可调用，且只能修改房间内成员的状态。
所有接口的结果会通过 `ITMG_MAIN_EVNET_TYPE_ROOM_MANAGEMENT_OPERATOR` 回调，回调详情请参考 回调处理 。

接口列表

类型	接口
控制采集	EnableMic、EnableAudioCaptureDevice、EnableAudioSend
控制播放	EnableSpeaker、EnableAudioPlayDevice、EnableAudioRecv
设备状态获取	GetMicState、GetSpeakerState
敏感接口	ForbidUserOperation

采集管理相关接口

采集管理接口包含**麦克风管理**、**音频上行管理**以及**采集硬件设备管理**。其中麦克风管理相当于音频上行管理加上采集硬件设备管理。

采集管理

调用此接口打开或关闭房间内某用户的麦克风，调用成功后，该用户的麦克风将被关闭或打开。
EnableMic 相当于同时调用 EnableAudioSend 及 EnableAudioCaptureDevice。

函数原型

Android

```
public abstract int EnableMic(boolean isEnabled,String receiverID);
```

iOS

```
-(QAVResult)EnableMic:(BOOL)enable Receiver:(NSString *)receiverID;
```

参数	类型	含义
enable	BOOL	YES：打开某用户麦克风，NO：关闭某用户麦克风
receiverID	NSString*	填入目标用户 OpenId

回调

回调参数为 ITMG_ROOM_MANAGEMENT_MIC_OP。

音频流发送管理

调用此接口打开或关闭房间内某用户的音频上行。调用成功后，该用户的音频上行将被关闭或打开，但不影响麦克风采集。

函数原型

Android

```
public abstract int EnableAudioSend(boolean isEnabled,String
receiverID);
```

iOS

```
-(QAVResult)EnableAudioSend:(BOOL)enable Receiver:(NSString
*)receiverID;
```

参数	类型	含义
enable	BOOL	YES：打开某用户上行，NO：关闭某用户上行
receiverID	NSString*	填入目标用户 OpenId

回调

回调参数为 ITMG_ROOM_MANAGEMENT_AUDIO_SEND_OP。

音频采集硬件管理

调用此接口打开或关闭房间内某用户的音频采集硬件设备。调用成功后，该用户的音频采集硬件设备将被关闭或打开，但不影响上行。

函数原型

Android

```
public abstract int EnableAudioCaptureDevice(boolean isEnabled,String
receiverID);
```

iOS

```
-(QAVResult)EnableAudioCaptureDevice:(BOOL)enabled Receiver:(NSString
*)receiverID;
```

参数	类型	含义
enable	BOOL	YES：打开某用户音频采集硬件设备，NO：关闭某用户音频采集硬件设备
receiverID	NSString*	填入目标用户 OpenId

回调

回调参数为 ITMG_ROOM_MANAGEMENT_CAPTURE_OP。

播放管理相关接口

播放管理接口包含**扬声器管理**、**音频下行管理**以及**播放硬件设备管理**。其中扬声器管理相当于音频下行管理加上播放硬件设备管理。

播放管理

调用此接口打开或关闭房间内某用户的扬声器。调用成功后，该用户的扬声器将被关闭或打开，听到房间内的音频声音。
EnableSpeaker 相当于同时调用 EnableAudioRecv 及 EnableAudioPlayDevice。

函数原型

Android

```
public abstract int EnableSpeaker(boolean isEnabled,String receiverID);
```

iOS

```
-(QAVResult)EnableSpeaker:(BOOL)enable Receiver:(NSString *)receiverID;
```

参数	类型	含义
enable	BOOL	YES：打开某用户扬声器，NO：关闭某用户扬声器
receiverID	NSString*	填入目标用户 OpenId

回调

回调参数为 ITMG_ROOM_MANAGEMENT_SPEAKER_OP。

音频流接收管理

调用此接口打开或关闭房间内某用户的音频下行。调用成功后，该用户的音频下行将被关闭或打开，但不影响播放设备。

函数原型

Android

```
public abstract int EnableAudioRecv(boolean isEnabled,String
receiverID);
```

iOS

```
-(QAVResult)EnableAudioRecv:(BOOL)enabled Receiver:(NSString
*)receiverID;
```

参数	类型	含义
enable	BOOL	YES：打开某用户音频下行，NO：关闭某用户音频下行
receiverID	NSString*	填入目标用户 OpenId

回调

回调参数为 ITMG_ROOM_MANAGEMENT_AUDIO_REC_OP。

音频播放硬件管理

调用此接口打开或关闭房间内某用户的音频播放硬件设备。调用成功后，该用户的音频播放硬件设备将被关闭或打开，但不影响下行。

函数原型

Android

```
public abstract int EnableAudioPlayDevice(boolean isEnabled,String
receiverID);
```

iOS

```
-(QAVResult)EnableAudioPlayDevice:(BOOL)enabled Receiver:(NSString
*)receiverID;
```

参数	类型	含义
enable	BOOL	YES：打开某用户音频播放硬件设备，NO：关闭某用户音频播放硬件设备

receiverID	NSString*	填入目标用户 OpenId

回调

回调参数为 ITMG_ROOM_MANAGEMENT_PLAY_OP。

获取成员状态接口

获取某人采集状态

调用此接口，获取房间内某成员的麦克风状态。

函数原型

Android

```
public abstract int GetMicState(String receiverID);
```

iOS

```
-(QAVResult)GetMicState:(NSString *)receiverID;
```

参数	类型	含义
receiverID	NSString*	填入目标用户 OpenId

回调

回调参数为 ITMG_ROOM_MANAGEMENT_GET_MIC_STATE。

获取某人播放状态

调用此接口，获取房间内某成员的扬声器状态。

函数原型

Android

```
public abstract int GetSpeakerState(String receiverID);
```

iOS

```
-(QAVResult)GetSpeakerState:(NSString *)receiverID;
```

回调

回调参数为 ITMG_ROOM_MANAGEMENT_GET_SPEAKER_STATE。

禁止某位成员操作采集及播放

成员进房默认为允许操作麦克风及扬声器，调用此接口可以禁止房间内的某位成员操作麦克风及扬声器，在该成员退出房间后此功能失效。

Android

```
public  abstract int ForbidUserOperation(boolean isEnabled,String
receiverID);
```

iOS

```
-(QAVResult)ForbidUserOperation:(BOOL)enable Receiver:(NSString
*)receiverID;
```

参数	类型	含义
enable	BOOL	YES：禁止某用户操作设备，NO：允许某用户操作设备
receiverID	NSString*	填入目标用户 OpenId

回调

回调参数为 ITMG_ROOM_MANAGERMENT_FOBIN_OP。

回调处理

与 GME 其他回调相同，房间管理的回调也在 OnEvent 中处理，事件名称为 `ITMG_MAIN_EVNET_TYPE_ROOM_MANAGEMENT_OPERATOR`，事件会返回一个结构体，具体包含的内容如下。

回调参数

参数	类型	含义

SenderID	NSString	事件发送者 ID，如果与自己 OpenId 相同，即为本端发送的命令
ReceiverID	NSString	事件接收者 ID，如果与自己 OpenId 相同，即为本端接收的命令
OperateType	NSNumber	事件类型
Result	NSNumber	事件结果，0为成功
OperateValue	NSNumber	命令详情

OperateType

数值	事件类型	含义
0	ITMG_ROOM_MANAGEMENT_CAPTURE_OP	控制采集设备硬件回调
1	ITMG_ROOM_MANAGEMENT_PLAY_OP	控制播放设备硬件回调
2	ITMG_ROOM_MANAGEMENT_AUDIO_SEND_OP	控制上行回调
3	ITMG_ROOM_MANAGEMENT_AUDIO_REC_OP	控制下行回调
4	ITMG_ROOM_MANAGEMENT_MIC_OP	控制麦克风回调
5	ITMG_ROOM_MANAGEMENT_PLAY_OP	控制扬声器回调
6	ITMG_ROOM_MANAGEMENT_GET_MIC_STATE	获取麦克风状态
7	ITMG_ROOM_MANAGEMENT_GET_SPEAKER_STATE	获取扬声器状态
8	ITMG_ROOM_MANAGERMENT_FOBIN_OP	禁止操作麦克风及扬声器事件

OperateValue

成员	含义
boolValue	0：关闭命令，1：打开命令

示例代码

Android

```
 public void OnEvent(ITMGContext.ITMG_MAIN_EVENT_TYPE type, Intent data)
{
 if
(ITMGContext.ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVNET_TYPE_ROOM_MANAGEMENT_O
PERATOR== type) {

            ArrayList<String> operatorArr = new ArrayList<String>();
            operatorArr.add("采集");
            operatorArr.add("播放");
            operatorArr.add("上行");
            operatorArr.add("下行");
            operatorArr.add("采集上行");
            operatorArr.add("播放下行");
            operatorArr.add("mic状态");
            operatorArr.add("spk状态");
            operatorArr.add("禁止操作mic/speak");

            String SenderID =  data.getStringExtra("SenderID");
            String ReceiverID = data.getStringExtra("ReceiverID");
            int OperateType = data.getIntExtra("OperateType",-1000);

            int Result =data.getIntExtra("Result",-1000);
            boolean OperateValue =
data.getBooleanExtra("OperateValue",false);
            if (OperateType == -1000 ||Result == -1000) {
                return;
            }
            if (SenderID.equals(identifier)) {
                if (OperateType ==
ITMGContext.ITMG_ROOM_MANAGEMENT_GET_MIC_STATE || OperateType ==
ITMGContext.ITMG_ROOM_MANAGEMENT_GET_SPEAKER_STATE) {
                    Toast.makeText(getActivity(), String.format("发送给
id:%s 的%s操作,结果:%s", ReceiverID, operatorArr.get(OperateType),
OperateValue ? "开" : "关"), Toast.LENGTH_LONG).show();
                } else  {
                    Toast.makeText(getActivity(), String.format("发送给
id:%s 的%s%s操作,结果:%d", ReceiverID, operatorArr.get(OperateType),
OperateValue ? "开" : "关", Result), Toast.LENGTH_LONG).show();
                }
```

```
            } else if
(ReceiverID.equals(identifier)||ReceiverID.equals("ALL")) {
                if (Result == 0) {
                    switch (OperateType) {
                        case
ITMGContext.ITMG_ROOM_MANAGEMENT_CAPTURE_OP:
                        {
                            if (!OperateValue) {
                                mSwitchCapture.setChecked(OperateValue);
                            } else  {
                                AlertDialog.Builder dialog = new
AlertDialog.Builder (getActivity());  //创建对象
                                dialog.setTitle("是否要打开设备采集");
                                dialog.setMessage("");
                                dialog.setCancelable(false);
                                dialog.setPositiveButton("开", new
DialogInterface.OnClickListener() {
                                    //设置确定按钮的点击事件
                                    @Override
                                    public void onClick(DialogInterface
dialog, int which) {
                                        mSwitchCapture.setChecked(true);

ITMGContext.GetInstance(getActivity()).GetAudioCtrl().EnableAudioCapture
Device(true);
                                    }
                                });
                                dialog.setNegativeButton("关", new
DialogInterface.OnClickListener() {
                                    //设置取消按钮的点击事件
                                    @Override
                                    public void onClick(DialogInterface
dialog, int which) {
                                    }
                                });
                                dialog.show();
                            }

                        }
```

```
                            break;
                        case ITMGContext.ITMG_ROOM_MANAGEMENT_PLAY_OP:
                        {
                            mSwitchPlayDevice.setChecked(OperateValue);
                        }
                            break;
                        case
ITMGContext.ITMG_ROOM_MANAGEMENT_AUDIO_SEND_OP:
                        {
                            if (!OperateValue) {
                                mSwitchSend.setChecked(OperateValue);
                            } else  {
                                AlertDialog.Builder dialog = new
AlertDialog.Builder (getActivity());  //创建对象
                                dialog.setTitle("是否要打开上行");
                                dialog.setMessage("");
                                dialog.setCancelable(false);
                                dialog.setPositiveButton("开", new
DialogInterface.OnClickListener() {
                                    //设置确定按钮的点击事件
                                    @Override
                                    public void onClick(DialogInterface
dialog, int which) {
                                        mSwitchSend.setChecked(true);

ITMGContext.GetInstance(getActivity()).GetAudioCtrl().EnableAudioSend(tr
ue);
                                    }
                                });
                                dialog.setNegativeButton("关", new
DialogInterface.OnClickListener() {
                                    //设置取消按钮的点击事件
                                    @Override
                                    public void onClick(DialogInterface
dialog, int which) {
                                    }
                                });
                                dialog.show();
                            }
                        }
```

```
                            break;
                        case
ITMGContext.ITMG_ROOM_MANAGEMENT_AUDIO_REC_OP:
                         {
                             mSwitchRecv.setChecked(OperateValue);
                        }
                            break;
                        case ITMGContext.ITMG_ROOM_MANAGEMENT_MIC_OP:
                         {
                             if (!OperateValue) {

mSwitchCapture.setChecked(OperateValue);
                                 mSwitchSend.setChecked(OperateValue);
                             } else {
                                 AlertDialog.Builder dialog = new
AlertDialog.Builder (getActivity());  //创建对象
                                 dialog.setTitle("是否要打开采集和上行");
                                 dialog.setMessage("");
                                 dialog.setCancelable(false);
                                 dialog.setPositiveButton("开", new
DialogInterface.OnClickListener() {
                                     //设置确定按钮的点击事件
                                     @Override
                                     public void onClick(DialogInterface
dialog, int which) {

mSwitchCapture.setChecked(true);
                                         mSwitchSend.setChecked(true);

ITMGContext.GetInstance(getActivity()).GetAudioCtrl().EnableMic(true);
                                     }
                                 });
                                 dialog.setNegativeButton("关", new
DialogInterface.OnClickListener() {
                                     //设置取消按钮的点击事件
                                     @Override
                                     public void onClick(DialogInterface
dialog, int which) {
                                     }
                                 });
```

```
                                dialog.show();
                            }
                        }
                            break;
                    case
ITMGContext.ITMG_ROOM_MANAGEMENT_SPEAKER_OP:
                    {
                        mSwitchPlayDevice.setChecked(OperateValue);
                        mSwitchRecv.setChecked(OperateValue);
                    }
                        break;

                }
            }
            if (OperateType ==
ITMGContext.ITMG_ROOM_MANAGEMENT_GET_MIC_STATE || OperateType ==
ITMGContext.ITMG_ROOM_MANAGEMENT_GET_SPEAKER_STATE)
            {
                Toast.makeText(getActivity(), String.format("收到来自
id:%s 的%s操作,结
果:%s",SenderID,operatorArr.get(OperateType),OperateValue?"开":"关"),
Toast.LENGTH_LONG).show();
            }
            else if (OperateType ==
ITMGContext.ITMG_ROOM_MANAGEMENT_SPEAKER_OP || OperateType ==
ITMGContext.ITMG_ROOM_MANAGEMENT_AUDIO_REC_OP|| OperateType ==
ITMGContext.ITMG_ROOM_MANAGEMENT_PLAY_OP|| OperateType ==
ITMGContext.ITMG_ROOM_MANAGERMENT_FOBIN_OP){
                Toast.makeText(getActivity(), String.format("收到来自
id:%s 的%s%s操作,结
果:%d",SenderID,operatorArr.get(OperateType),OperateValue?"开":"关",Resul
t), Toast.LENGTH_LONG).show();
            } else if (OperateValue == false) {
                Toast.makeText(getActivity(), String.format("收到来自
id:%s 的%s%s操作,结
果:%d",SenderID,operatorArr.get(OperateType),OperateValue?"开":"关",Resul
t), Toast.LENGTH_LONG).show();
            }
        }
```

```
}
```

iOS

```
-(void)OnEvent:(ITMG_MAIN_EVENT_TYPE)eventType data:(NSDictionary
*)data{
    NSString* log =[NSString stringWithFormat:@"OnEvent:%d,data:%@",
(int)eventType, data];
    [self showLog:log];
    NSLog(@"====%@====",log);
    switch (eventType) {
        case ITMG_MAIN_EVNET_TYPE_ROOM_MANAGEMENT_OPERATOR:
        {
            NSArray *operatorArr = @[@"采集",@"播放",@"上行",@"下行",@"采集
上行",@"播放下行",@"踢人",@"mic状态",@"spk状态",@"禁止操作mic/speak"];
            // _openId
            NSString *SenderID = [data objectForKey:@"SenderID"];
            NSString *ReceiverID = [data objectForKey:@"ReceiverID"];
            NSNumber *OperateType = [data objectForKey:@"OperateType"];
            NSNumber *Result = [data objectForKey:@"Result"];
            NSNumber *OperateValue = [data
objectForKey:@"OperateValue"];

            //自己发出去的命令
            if ([SenderID isEqualToString:_openId]) {
                if (OperateType.intValue ==
ITMG_ROOM_MANAGEMENT_GET_MIC_STATE || OperateType.intValue ==
ITMG_ROOM_MANAGEMENT_GET_SPEAKER_STATE) {
                        NSString *alterString = [NSString
stringWithFormat:@"发送给id:%@ 的%@操作,结
果:%@",ReceiverID,operatorArr[OperateType.intValue],OperateValue.boolValu
e?@"开":@"关"];
                                    UIAlertView *alert =
[[UIAlertView alloc] initWithTitle:@"房间管理操作" message:alterString
delegate:NULL cancelButtonTitle:@"OK" otherButtonTitles:nil];
                                    [alert show];
                }
                else
                {
```

```
                    NSString *alterString = [NSString
stringWithFormat:@"发送给id:%@ 的%@%@操作,结
果:%@",ReceiverID,OperateValue.boolValue?
@"开":@"关",operatorArr[OperateType.intValue],Result];
                                UIAlertView *alert = [[UIAlertView alloc]
initWithTitle:@"房间管理操作" message:alterString delegate:NULL
cancelButtonTitle:@"OK" otherButtonTitles:nil];
                                [alert show];
                }



            }
            else if([ReceiverID isEqualToString:_openId] ){ //别人发过来的
命令
                if (Result.intValue == 0) {
                    switch (OperateType.intValue) {
                        case ITMG_ROOM_MANAGEMENT_CAPTURE_OP:{
                            [_micSwitch setOn:OperateValue.boolValue
animated:true];
                        }
                            break;
                        case ITMG_ROOM_MANAGEMENT_PLAY_OP:{
                            [_speakerSwitch setOn:OperateValue.boolValue
animated:true];
                            }
                            break;
                        case ITMG_ROOM_MANAGEMENT_AUDIO_SEND_OP:{
                            [_sendSwitch setOn:OperateValue.boolValue
animated:true];
                        }
                            break;
                        case ITMG_ROOM_MANAGEMENT_AUDIO_REC_OP:{
                            [_recvSwitch setOn:OperateValue.boolValue
animated:true];
                        }
                            break;
                        case ITMG_ROOM_MANAGEMENT_MIC_OP:{
                        [_micSwitch setOn:OperateValue.boolValue
animated:true];
```

```
                        [_sendSwitch setOn:OperateValue.boolValue
animated:true];
                        }
                            break;
                        case ITMG_ROOM_MANAGEMENT_SPEAKER_OP:{
                        [_speakerSwitch setOn:OperateValue.boolValue
animated:true];
                        [_recvSwitch setOn:OperateValue.boolValue
animated:true];

                        }
                            break;
                        default:
                            break;
                    }

                if (OperateType.intValue ==
ITMG_ROOM_MANAGEMENT_GET_MIC_STATE || OperateType.intValue ==
ITMG_ROOM_MANAGEMENT_GET_SPEAKER_STATE) {
                        NSString *alterString = [NSString
stringWithFormat:@"收到id:%@ 发送来的%@操作,结
果:%@",SenderID,operatorArr[OperateType.intValue],OperateValue.boolValue?
@"开":@"关"];
                                UIAlertView *alert = [[UIAlertView
alloc] initWithTitle:@"房间管理操作" message:alterString delegate:NULL
cancelButtonTitle:@"OK" otherButtonTitles:nil];
                                [alert show];
                    }
                else{
                        NSString *alterString = [NSString
stringWithFormat:@"收到id:%@ 发送来的%@%@操作,结
果:%@",SenderID,OperateValue.boolValue?
@"开":@"关",operatorArr[OperateType.intValue],Result];
                               UIAlertView *alert = [[UIAlertView alloc]
initWithTitle:@"房间管理操作" message:alterString delegate:NULL
cancelButtonTitle:@"OK" otherButtonTitles:nil];
                                [alert show];
                    }
                }
            }
```

```
            }
            break;
    }
```

语音消息转文本热词热句
语音消息转文本热句

最近更新时间：2024-10-15 14:45:41

为方便开发者快速实现**语音消息转文本热句模型**（以下用**热句**指代）功能，本文为您介绍语音消息转文本热句的使用场景以及接入流程。

使用场景

在 GME 转文本的服务中，对一些特有的人名、产品名、公司名或者某个领域的专有词汇，例如游戏术语开黑、游戏中地名等，可能存在识别准确率不高的情况。对于这些专有词汇，GME 提供了热句模型供用户使用，添加热句可以显著提升专有词汇识别准确率。

> **说明：**
> 热句模型功能目前仅支持中文识别，支持通过 云API 方式和控制台方式进行热句的创建。

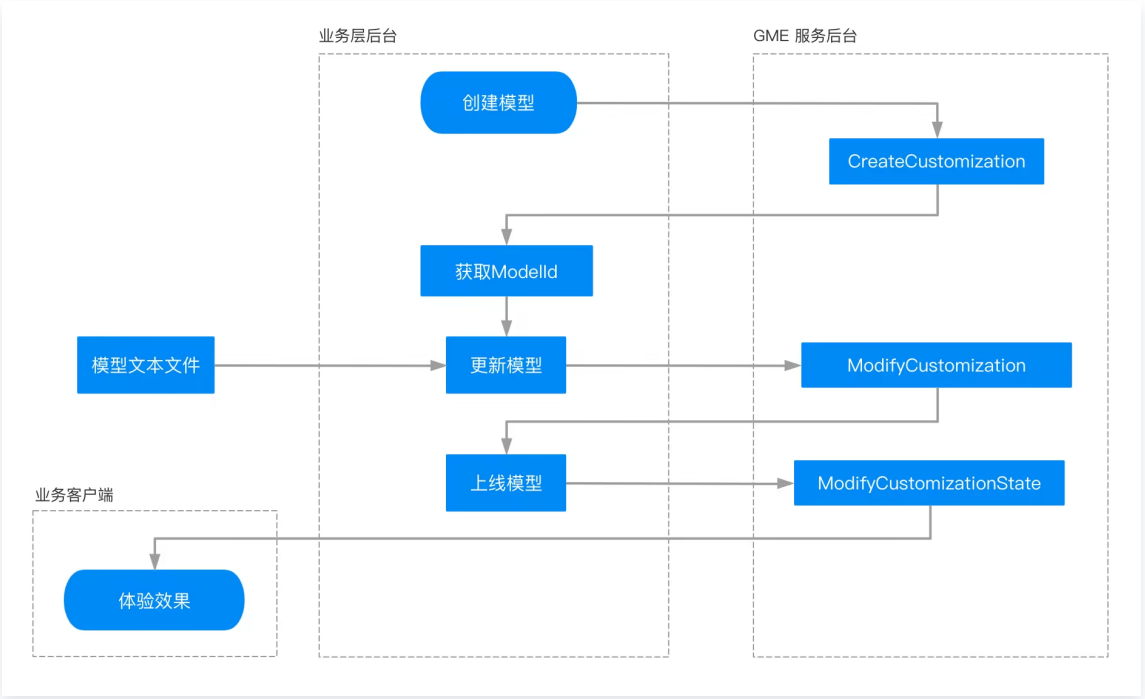
前提条件

- **已开通语音转文本服务**：如果使用流式语音转文本服务，需要在控制台开通语音转文本服务；如果需要语音消息服务，需要在控制台额外开通语音消息服务，具体开通详情可参见 服务开通指引 。
- **已接入 GME SDK**：使用语音消息转文本热句模型功能，至少需要接入流式语音转文本功能，或者接入语音消息功能及转文本功能。GME SDK 的调用和接入请参见 Native SDK 快速接入 、 Unity SDK 快速接入 、 Unreal SDK 快速接入 。
- **开通语音消息转文本热句模型功能**：请 提交工单 联系 GME 团队申请。

> **说明：**
> 语音消息转文本功能将产生语音消息服务费用、语音转文本服务费用。详情请参见 计费文档 。

实现方式1：云 API 调用

通过云 API 使用热句功能的调用流程如下：



1. 创建模型
2. 更新模型
3. 上线模型
4. 体验效果
5. 其他功能

步骤1：创建模型

使用语音消息转文本热句模型功能首先需要创建模型，调用 CreateCustomization 接口创建模型，接口会返回创建完的模型 id。一个 GME Appid 只能对应创建一个模型。

步骤2：更新模型

通过 ModifyCustomization 接口可以更新模型。更新模型时需要提交训练文本文件的下载地址，目前仅支持腾讯云对象存储（COS）。

优化建议

- 训练数据为用户在专有领域和行业积累的文本数据，数据越接近真实使用场景，识别准确率越高。
- 训练文本中的数字部分最好按照发音替换为对应的汉字，例如“167方向”替换为“一百六十七方向”。

格式要求

- 训练文本格式要求 UTF-8 或 GBK 编码的 txt 文件，文件最大不超过2MB。
- 每行添加一个词或一句话，详见下方“训练文本数据示例”。
- 每行的字节数最多为1000，总行数最多为50万行。
- 确认模型后，需要上线模型才能体验效果。

- 训练文本数据示例：
  我还在毒圈，等等我。
  有空投，我们往空投位置走吧。
  谁要倍镜，我有个六倍镜。
  我看到红房子二楼有个人，一百七十五方向。

步骤3：上线模型

模型更新完成后，需要上线才能使用，调用 ModifyCustomizationState 上线模型。

步骤4：体验效果

在客户端已接入的流式语音转文本功能或者语音消息转文本功能中体验热句效果。

步骤5：其他功能

- 已经上线的模型可以调用 ModifyCustomizationState 接口下线。
- 通过 GetCustomizationList 可以查询目前使用的模型 ID。
- 通过 DeleteCustomization 可以删除模型。

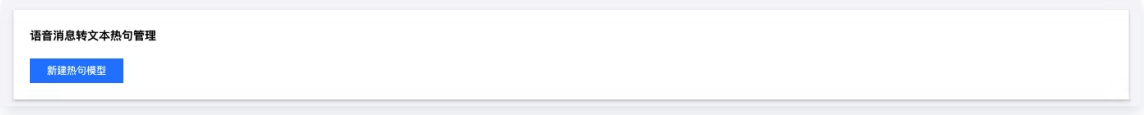
实现方式2：控制台操作

除了使用相关的云API接口外，您也可以直接在控制台界面操作，操作步骤如下：

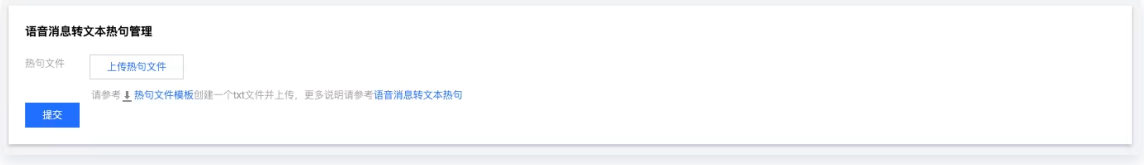
1. 创建模型
2. 上线模型
3. 下线模型
4. 更新模型
5. 删除模型
6. 体验效果

步骤1：创建模型

登录控制台 热词热句管理 页面，在上方选择需要使用语音消息转文本热句模型的应用，单击**新建热句模型**。

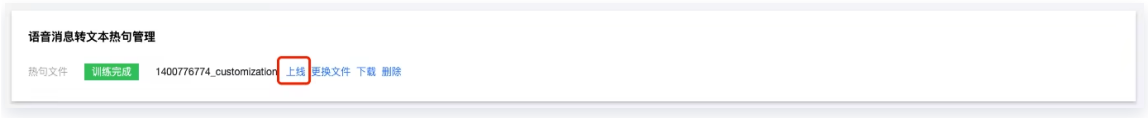


可下载热句文件模板，按格式要求填写完成后单击**上传热句文件**，上传后点击**提交**，模型开始训练。模型的训练、上线、下线需要约一分钟左右时间。



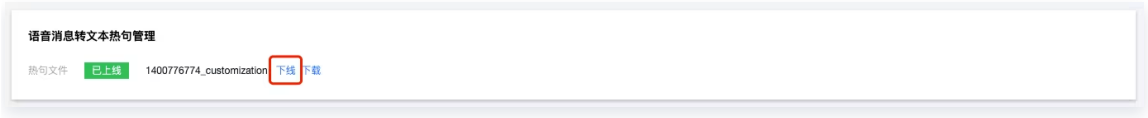
步骤2：上线模型

模型训练完成后，需要上线才能使用，单击**上线**按钮上线该模型。



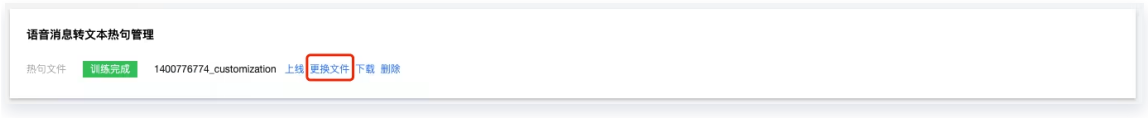
步骤3：下线模型

如需修改模型，需要先将已上线的模型下线。模型下线后，相应的接口调用将使用默认通用模型进行识别。



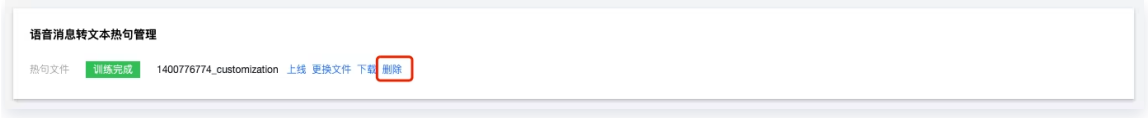
步骤4：更新模型

将原上线的模型下线后，单击**更换文件**重新上传新模型，模型更新后，需要重新训练才能使用。



步骤5：删除模型

单击**删除**即可删除掉现有模型。



步骤6：体验效果

在客户端已接入的流式语音转文本功能或者语音消息转文本功能中体验热句效果。

语音消息转文本热词

最近更新时间：2024-10-15 14:45:41

为方便开发者快速实现**语音消息转文本热词模型**（以下用**热词**指代）功能，本文为您介绍语音消息转文本热词的使用场景以及接入流程。

使用场景

在 GME 转文本的服务中，对一些特有的人名、产品名、公司名或者某个领域的专有词汇，例如游戏术语开黑、游戏中地名等，可能存在识别准确率不高的情况。对于这些专有词汇，GME 提供了热词模型供用户使用，添加热词可以显著提升专有词汇识别准确率。

> **说明：**
> 热词模型功能目前仅支持中文识别，支持通过控制台方式进行热词的创建。

前提条件

- **已开通语音转文本服务**：如果使用流式语音转文本服务，需要在控制台开通语音转文本服务；如果需要语音消息服务，需要在控制台额外开通语音消息服务，具体开通详情可参见 服务开通指引 。
- **已接入 GME SDK**：使用语音消息转文本热词模型功能，至少需要接入流式语音转文本功能，或者接入语音消息功能及转文本功能。GME SDK 的调用和接入请参见 Native SDK 快速接入 、 Unity SDK 快速接入 、 Unreal SDK 快速接入 。
- **开通语音消息转文本热词模型功能**：请 提交工单 联系 GME 团队申请。

> **说明：**
> 语音消息转文本功能将产生语音消息服务费用、语音转文本服务费用。详情请参见 计费文档 。

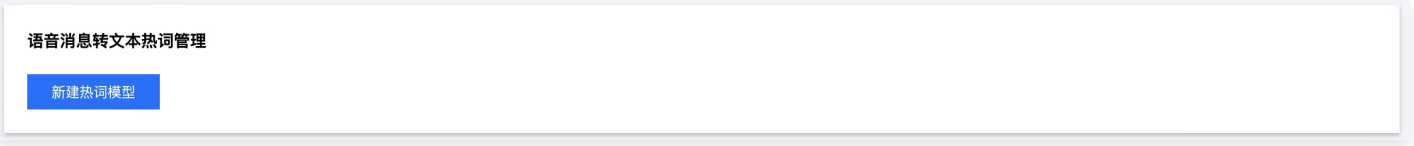
实现方式：控制台操作

您可以直接在控制台界面操作，操作步骤如下：

1. 创建模型
2. 上线模型
3. 下线模型
4. 更新模型
5. 删除模型
6. 体验效果

步骤1： 创建模型

登录控制台 热词热句管理 页面，在上方选择需要使用语音消息转文本热词模型的应用，单击**新建热词模型**。



可下载热词文件模板，按格式要求填写完成后单击**上传热词文件**，上传后单击**提交**即可。

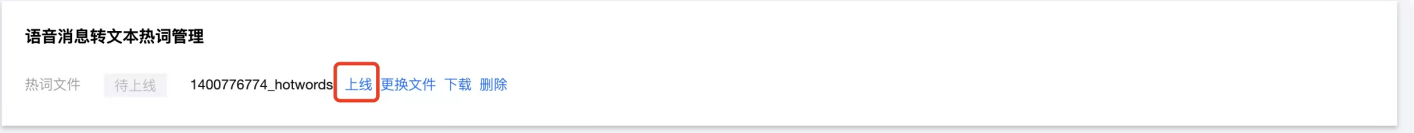
格式要求

- 模型文本格式要求 UTF-8 或 GBK 编码的 txt 文件，文件最大不超过400KB。
- 每行仅添加一个热词和对应权重且不能包含标点和特殊字符（热词和权重间用英文符号|隔开），详见下方“模型文本数据示例”。
- 每个热词表最多可添加128个词，每个词最长10个字，不能超出限制（如有超过限制的需求，请使用 热句模型 ）。
- 热词权重取值范围为[1,10]之间的整数，权重越大代表该词被识别出来的概率越大。
- 模型文本数据示例：

  开黑|10
  上分|10
  上路|9
  中路|8
  空投|9

步骤2： 上线模型

模型训练完成后，需要上线才能使用，单击**上线**按钮上线该模型。

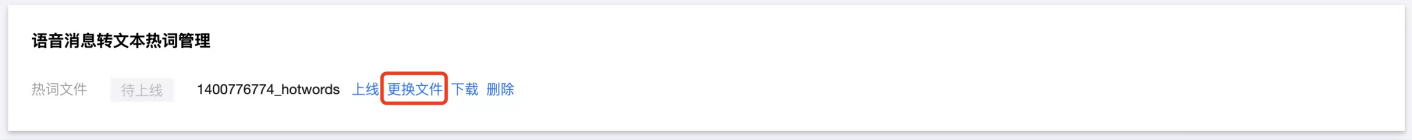


步骤3： 下线模型

如需修改模型，需要先将已上线的模型下线。模型下线后，需要重新上线热词才能使用该热词模型进行识别。

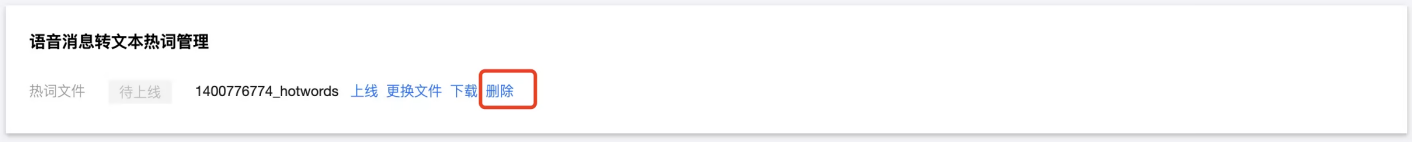
步骤4： 更新模型

将原上线的模型下线后，单击**更换文件**重新上传新模型。



步骤5： 删除模型

单击**删除**即可删除掉现有模型，热词模型删除后，需要重新上传热词文件。



步骤6： 体验效果

在客户端已接入的流式语音转文本功能或者语音消息转文本功能中体验热词效果。

音效与伴奏
语音变声

最近更新时间：2025-11-14 11:19:41

为方便开发者调试和接入腾讯云游戏多媒体引擎产品 API，本文向您介绍游戏多媒体引擎变声特效的接入方法。

使用场景

关于变声玩法的效果请参见 实时语音趣味变声，大叔变声“妙音娘子”Get一下 。

前提条件

- **已开通实时语音服务**：可参见 服务开通指引 。
- **已开通语音转文本服务**：可参见 服务开通指引 。
- **已接入 GME SDK**：包括核心接口和实时语音接口的接入，详情可参见 Native SDK 快速接入 、Unity SDK 快速接入 、Unreal SDK 快速接入 。
- **已接入 GME SDK 库文件libgmesoundtouch**：需要保证工程库文件中带有 libgmesoundtouch，具体参见 库文件对应功能 。

实时语音变声接入

变声接口

在已经进房成功且打开麦克风的情况下，调用 SetVoiceType 接口设置变声特效，接口返回0代表调用成功，房间内的人听到的本端发出的声音带有变声效果。如果要自测变声效果，请使用耳返功能（接口：EnableLoopBack）。

函数原型

Android

```
public abstract int SetVoiceType(int type);
```

iOS

```
-(QAVResult)SetVoiceType:(ITMG_VOICE_TYPE) type
```

Unity

```
public abstract int SetVoiceType(int voiceType);
```

C++

```
virtual int SetVoiceType(ITMG_VOICE_TYPE voiceType) = 0;
```

参数	类型	意义
type	int	表示本端音频变声类型

变声类型

类型参数	参数代表	意义
ITMG_VOICE_TYPE_ORIGINAL_SOUND	0	原声
ITMG_VOICE_TYPE_LOLITA	1	小女孩
ITMG_VOICE_TYPE_UNCLE	2	大叔
ITMG_VOICE_TYPE_INTANGIBLE	3	空灵
ITMG_VOICE_TYPE_DEAD_FATBOY	4	小胖子
ITMG_VOICE_TYPE_HEAVY_MENTA	5	重金属

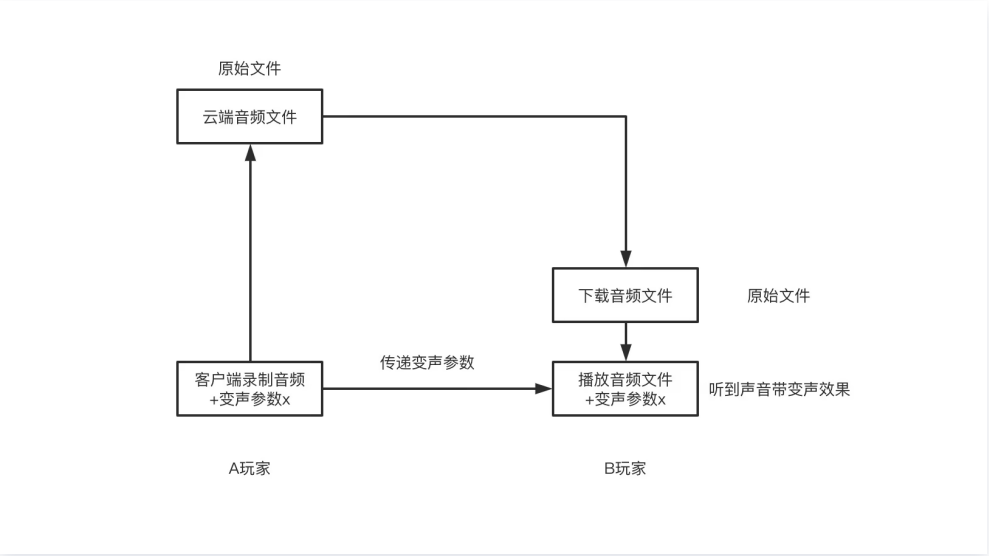
ITMG_VOICE_TYPE_DIALECT	6	歪果仁
ITMG_VOICE_TYPE_INFLUENZA	7	感冒
ITMG_VOICE_TYPE_CAGED_ANIMAL	8	困兽
ITMG_VOICE_TYPE_HEAVY_MACHINE	9	重机器
ITMG_VOICE_TYPE_STRONG_CURRENT	10	强电流
ITMG_VOICE_TYPE_KINDER_GARTEN	11	幼稚园
ITMG_VOICE_TYPE_HUANG	12	小顽童

示例代码

```
int ret = m_pTmgContext->GetAudioEffectCtrl()->SetVoiceType((ITMG_VOICE_TYPE)m_ComBoxVoiceChangeType.GetCurSel());
    if (ret != AV_OK) {
        char result[128] = {0};
        snprintf(result, sizeof result, "SetVoiceType failed error code: %d", ret);
        m_EditMonitor.SetWindowText(MByteToWChar(std::string(result).c_str());
    }
```

语音消息变声接入

语音消息变声流程



语音消息变声不会影响原始的音频信息，在播放的时候才体现变声效果。

语音消息播放

语音消息播放接口，带有变声效果参数。

Android

```
public abstract int PlayRecordedFile(String filePath,int voicetype);
```

iOS

```
-(int)PlayRecordedFile:(NSString*)filePath VoiceType:(ITMG_VOICE_TYPE)
type
```

Unity

```
ITMGPTT PlayRecordedFile(string filePath,int voiceType);
```

C++

```
public abstract int PlayRecordedFile(string filePath,int voiceType);
```

参数	类型	含义

filePath	string	本地语音文件的路径
voicetype	int	变声类型

错误码

错误码值	原因	建议方案
20485	播放未开始	确保文件存在，文件路径的合法性

实时语音伴奏

最近更新时间：2024-10-15 14:45:41

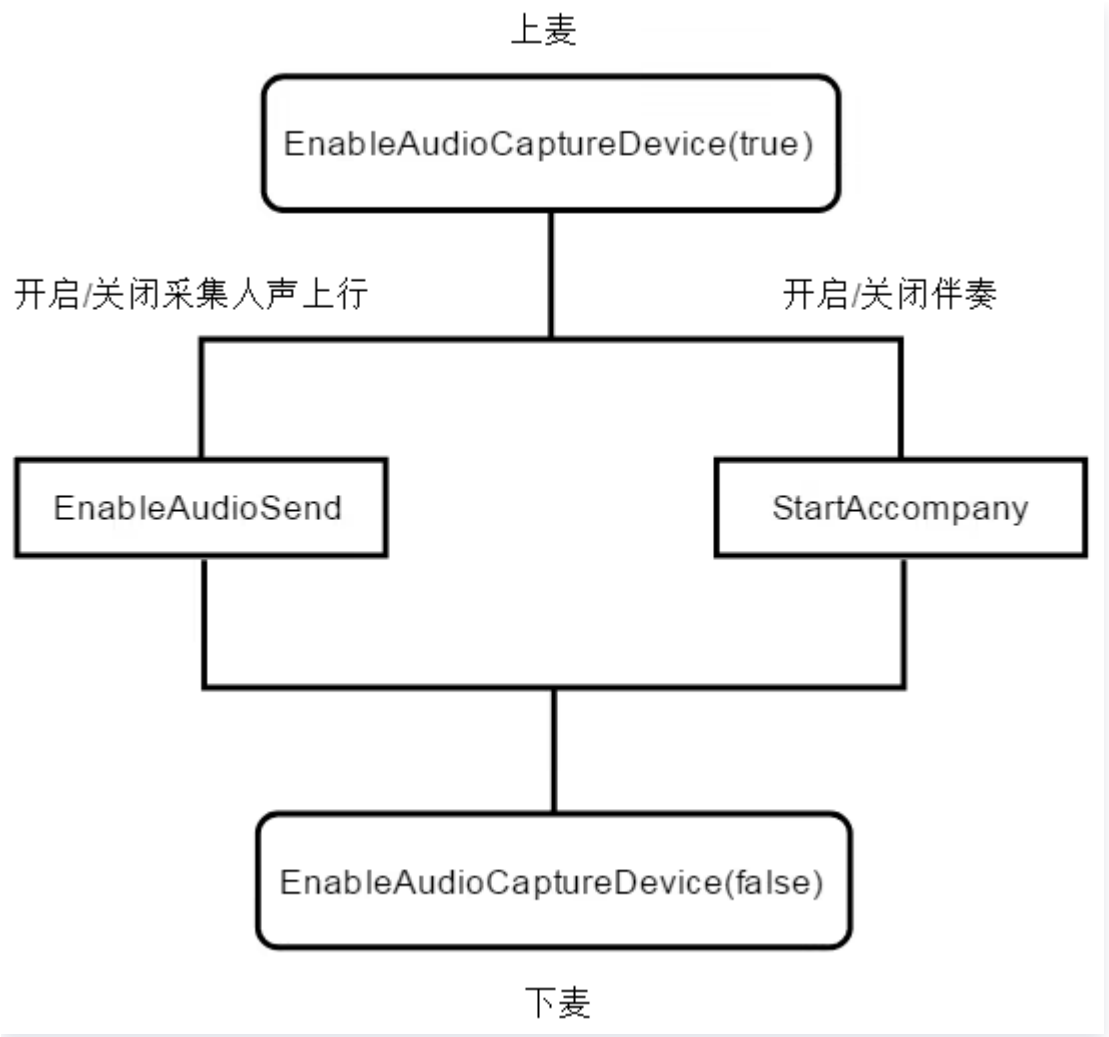
为方便开发者调试和接入腾讯云游戏多媒体引擎产品 API，本文为您介绍游戏多媒体引擎实时语音伴奏的接入技术文档。

实时语音伴奏相关接口

接口	接口含义
StartAccompany	开始播放伴奏。
StopAccompany	停止播放伴奏。
IsAccompanyPlayEnd	伴奏是否播放完毕。
PauseAccompany	暂停播放伴奏。
ResumeAccompany	重新播放伴奏。
SetAccompanyVolume	设置伴奏音量。
GetAccompanyVolume	获取播放伴奏的音量。
SetAccompanyFileCurrentPlayedTimeByMs	设置播放进度。

说明：
如需使用实时语音伴奏，需要在接入 GME SDK 且能在进行实时语音通话的情况下，才可以使用实时语音伴奏。

流程图



如何配合 EnableAudioCaputreDevice 使用

在进入实时语音房间成功之后，调用 EnableAudioCaputreDevice 打开采集设备，再调用 StartAccompany 播放伴奏。如果需要采集人声，可以调用 EnableAudioSend 实现开麦效果。

开始播放伴奏

调用 StartAccompany 接口开始播放伴奏。支持 m4a、wav、mp3 一共三种格式。调用此 API，音量会重置。

函数原型

```
ITMGAudioEffectCtrl virtual int StartAccompany(const char* filePath,
bool loopBack, int loopCount, int msTime)
```

参数	类型	意义
filePath	char*	伴奏音频的路径，支持本地文件和在线音频url。

loopBack	bool	是否混音发送，一般都设置为 true，即其他人也能听到伴奏。
loopCount	int	循环次数，数值为-1表示无限循环。填0不播放。
msTime	int	延迟时间。

示例代码

```
//Windows端代码
ITMGContextGetInstance()->GetAudioEffectCtrl()-
>StartAccompany(filePath,true,-1,0);
//Android端代码
ITMGContext.GetInstance(this).GetAudioEffectCtrl().StartAccompany(filePa
th,true,loopCount,0);
//iOS端代码
[[[ITMGContext GetInstance] GetAudioEffectCtrl] StartAccompany:path
loopBack:isLoopBack loopCount:loopCount msTime:0];
```

说明
iOS 端需要以下配置。

1. 在 iOS 端使用此功能，需要将相关动态库引入工程中，单击下载 mp3 动态库 。
2. 将下载好的文件引入到工程文件中。并在 Link Binary With Libraries 中添加此动态库。
3. 将头文件 TMGEngine_adv.h 加入工程中，与其他 SDK 头文件同目录下。

播放伴奏的回调

开始播放伴奏完成后，回调函数调用 OnEvent，事件消息为 ITMG_MAIN_EVENT_TYPE_ACCOMPANY_FINISH，在 OnEvent 函数中对事件消息进行判断。传递的参数 data 包含两个信息，一个是 result，另一个是 file_path。

示例代码

```
void TMGTestScene::OnEvent(ITMG_MAIN_EVENT_TYPE eventType,const char*
data){
    switch (eventType) {
        case ITMG_MAIN_EVENT_TYPE_ENTER_ROOM:
        {
        //进行处理
        break;
```

```
        }
        ...
        case ITMG_MAIN_EVENT_TYPE_ACCOMPANY_FINISH:
        {
        //进行处理
        break;
        }
    }
}
```

停止播放伴奏

调用 StopAccompany 接口停止播放伴奏。

函数原型

```
ITMGAudioEffectCtrl virtual int StopAccompany(int duckerTime)
```

参数	类型	意义
duckerTime	int	淡出时间。

示例代码

```
ITMGContextGetInstance()->GetAudioEffectCtrl()->StopAccompany(0);
```

伴奏是否播放完毕

如果播放完毕，返回值为 true，如果没播放完，返回值为 false。

函数原型

```
ITMGAudioEffectCtrl virtual bool IsAccompanyPlayEnd()
```

示例代码

```
ITMGContextGetInstance()->GetAudioEffectCtrl()->IsAccompanyPlayEnd();
```

暂停播放伴奏

调用 PauseAccompany 接口暂停播放伴奏。

函数原型

```
ITMGAudioEffectCtrl virtual int PauseAccompany()
```

示例代码

```
ITMGContextGetInstance()->GetAudioEffectCtrl()->PauseAccompany();
```

恢复播放伴奏

ResumeAccompany 接口用于恢复播放伴奏。

函数原型

```
ITMGAudioEffectCtrl virtual int ResumeAccompany()
```

示例代码

```
ITMGContextGetInstance()->GetAudioEffectCtrl()->ResumeAccompany();
```

设置自己是否可以听到伴奏

此接口用于设置自己是否可以听到伴奏。

函数原型

```
ITMGAudioEffectCtrl virtual int EnableAccompanyPlay(bool enable)
```

参数	类型	意义
enable	bool	是否能听到。

示例代码

```
ITMGContextGetInstance()->GetAudioEffectCtrl()-
>EnableAccompanyPlay(false);
```

设置他人是否也可以听到伴奏

设置他人是否也可以听到伴奏。

函数原型

```
ITMGAudioEffectCtrl virtual int EnableAccompanyLoopBack(bool enable)
```

参数	类型	意义
enable	bool	是否能听到。

示例代码

```
ITMGContextGetInstance()->GetAudioEffectCtrl()-
>EnableAccompanyLoopBack(false);
```

设置伴奏音量

调用 SetAccompanyVolume 接口设置伴奏音量，默认值为100，数值大于100音量增益，数值小于100音量减益，值域为0 – 200。

函数原型

```
ITMGAudioEffectCtrl virtual int SetAccompanyVolume(int vol)
```

参数	类型	意义
vol	int	音量数值。

示例代码

```
int vol=100;
ITMGContextGetInstance()->GetAudioEffectCtrl()->SetAccompanyVolume(vol);
```

获取播放伴奏的音量

GetAccompanyVolume 接口用于获取伴奏音量。

函数原型

```
ITMGAudioEffectCtrl virtual int GetAccompanyVolume()
```

示例代码

```
ITMGContextGetInstance()->GetAudioEffectCtrl()->GetAccompanyVolume();
```

获得伴奏播放进度

以下两个接口用于获得伴奏播放进度。需要注意：Current / Total = 当前循环次数，Current % Total = 当前循环播放位置。

函数原型

```
ITMGAudioEffectCtrl virtual int GetAccompanyFileTotalTimeByMs()
ITMGAudioEffectCtrl virtual int GetAccompanyFileCurrentPlayedTimeByMs()
```

示例代码

```
ITMGContextGetInstance()->GetAudioEffectCtrl()-
>GetAccompanyFileTotalTimeByMs();
ITMGContextGetInstance()->GetAudioEffectCtrl()-
>GetAccompanyFileCurrentPlayedTimeByMs();
```

设置播放进度

SetAccompanyFileCurrentPlayedTimeByMs 接口用于设置播放进度。

函数原型

```
ITMGAudioEffectCtrl virtual int
SetAccompanyFileCurrentPlayedTimeByMs(unsigned int time)
```

参数	类型	意义
time	int	播放进度，以毫秒为单位。

示例代码

```
ITMGContextGetInstance()->GetAudioEffectCtrl()->SetAccompanyFileCurrentPlayedTimeByMs(time);
```

设置伴奏音调

SetAccompanyKey 接口用于调整伴奏的音调，在启动伴奏之前调用。

函数原型

```
ITMGAudioEffectCtrl virtual int SetAccompanyKey(int nKey)
```

参数	类型	意义
nKey	int	升降 Key，推荐范围-4到4。当设置为0时为原声调。

错误码列表

错误码名称	错误码值	错误码含义	解决方法
QAV_ERR_ACC_OPENFILE_FAILED	4001	打开文件失败	检查文件路径及文件是否存在，检查是否有访问文件的权限。
QAV_ERR_ACC_FILE_FORAMT_NOT SUPPORT	4002	不支持的文件格式	检查文件格式是否正确。
QAV_ERR_ACC_DECODER_FAILED	4003	解码失败	检查文件格式是否正确。
QAV_ERR_ACC_BAD_PARAM	4004	参数错误	检查代码中所填参数是否正确。
QAV_ERR_ACC_MEMORY_ALLOC_FAILED	4005	内存分配失败	系统资源耗尽，如果一直存在此错误码，请联系开发人员。
QAV_ERR_ACC_CREATE_THREAD_FAILED	4006	创建线程失败	系统资源耗尽，如果一直存在此错误码，请联系开发人员。
QAV_ERR_ACC_STATE_ILLIGAL	4007	状态非法	未处于某种状态，去调用需要处于这个状态才允许调用的接口时，则会产生这个错误。

实时语音音效

最近更新时间：2025-11-14 11:19:41

为方便开发者调试和接入腾讯云游戏多媒体引擎产品 API，这里向您介绍游戏多媒体引擎实时语音音效的接入技术文档。

前提条件

- **已开通实时语音服务**：可参见 服务开通指引 。
- **已接入 GME SDK**：包括核心接口和实时语音接口的接入，详情可参见 Native SDK 快速接入 、Unity SDK 快速接入 、Unreal SDK 快速接入 。
- 使用 GME 实时语音功能成功进入语音房间，并且打开了麦克风（EnableMic）、扬声器（EnableSpeaker）。

实时语音音效相关接口

接口	接口含义
PlayEffect	播放音效
PauseEffect	暂停播放音效
PauseAllEffects	暂停所有音效
ResumeEffect	重新播放音效
ResumeAllEffects	重新播放所有音效
StopEffect	停止播放音效
StopAllEffects	停止播放所有音效
SetVoiceType	变声特效
SetKaraokeType	K 歌音效特效
GetEffectsVolume	获取播放音效的音量
SetEffectsVolume	设置播放音效的音量

播放音效

PlayEffect 接口用于播放音效。参数中音效 ID 需要 App 侧进行管理，ID 代表一次独立的播放事件。后续可以根据此 ID 控制此次播放。文件支持 m4a、wav、mp3 一共三种格式。

函数原型

```
ITMGAudioEffectCtrl virtual int PlayEffect(int soundId,  const char*
filePath, bool loop, double pitch, double pan, double gain)
```

参数	类型	意义
soundId	int	音效 ID
filePath	char*	音效路径
loop	bool	是否重复播放
pitch	double	播放频率，默认为1.0，该值越小播放速度越慢、时间越长
pan	double	声道，取值范围为 -1.0到1.0之间，-1.0表示只开启左声道
gain	double	增益音量，取值范围为 0.0到 1.0之间，默认为1.0

示例代码

```
double pitch = 1.0;
double pan = 0.0;
double gain = 0.0;
//Windows端
ITMGContextGetInstance()->GetAudioEffectCtrl()-
>PlayEffect(soundId,filepath,true,pitch,pan,gain);
//Android端
ITMGContext.GetInstance(this).GetAudioEffectCtrl().PlayEffect(soundId,fi
lePath,loop);
//iOS端
[[[ITMGContext GetInstance] GetAudioEffectCtrl] PlayEffect:soundId
filePath:path loop:isLoop];
```

暂停播放音效

PauseEffect 接口用于暂停播放音效。

函数原型

```
ITMGAudioEffectCtrl virtual int PauseEffect(int soundId)
```

参数	类型	意义
soundId	int	音效 ID

示例代码

```
ITMGContextGetInstance()->GetAudioEffectCtrl()->PauseEffect(soundId);
```

暂停所有音效

调用 PauseAllEffects 接口暂停所有音效

函数原型

```
ITMGAudioEffectCtrl virtual int PauseAllEffects()
```

示例代码

```
ITMGContextGetInstance()->GetAudioEffectCtrl()->PauseAllEffects();
```

重新播放音效

ResumeEffect 接口用于重新播放音效。

函数原型

```
ITMGAudioEffectCtrl virtual int ResumeEffect(int soundId)
```

参数	类型	意义
soundId	int	音效 ID

示例代码

```
ITMGContextGetInstance()->GetAudioEffectCtrl()->ResumeEffect(soundId);
```

重新播放所有音效

调用 ResumeAllEffects 接口重新播放所有音效。

函数原型

```
ITMGAudioEffectCtrl virtual int ResumeAllEffects()
```

示例代码

```
ITMGContextGetInstance()->GetAudioEffectCtrl()->ResumeAllEffects();
```

停止播放音效

StopEffect 接口用于停止播放音效。

函数原型

```
ITMGAudioEffectCtrl virtual int StopEffect(int soundId)
```

参数	类型	意义
soundId	int	音效 ID

示例代码

```
ITMGContextGetInstance()->GetAudioEffectCtrl()->StopEffect(soundId);
```

停止播放所有音效

调用 StopAllEffects 接口停止播放所有音效。

函数原型

```
ITMGAudioEffectCtrl virtual int StopAllEffects()
```

示例代码

```
ITMGContextGetInstance()->GetAudioEffectCtrl()->StopAllEffects();
```

变声特效

调用 SetVoiceType 接口设置变声特效。

函数原型

```
TMGAudioEffectCtrl int setVoiceType(int type)
```

参数	类型	意义
type	int	表示本端音频变声类型

类型参数	参数代表	意义
ITMG_VOICE_TYPE_ORIGINAL_SOUND	0	原声
ITMG_VOICE_TYPE_LOLITA	1	小女孩
ITMG_VOICE_TYPE_UNCLE	2	大叔
ITMG_VOICE_TYPE_INTANGIBLE	3	空灵
ITMG_VOICE_TYPE_DEAD_FATBOY	4	小胖子
ITMG_VOICE_TYPE_HEAVY_MENTA	5	重金属
ITMG_VOICE_TYPE_DIALECT	6	歪果仁
ITMG_VOICE_TYPE_INFLUENZA	7	感冒
ITMG_VOICE_TYPE_CAGED_ANIMAL	8	困兽
ITMG_VOICE_TYPE_HEAVY_MACHINE	9	重机器
ITMG_VOICE_TYPE_STRONG_CURRENT	10	强电流
ITMG_VOICE_TYPE_KINDER_GARTEN	11	幼稚园
ITMG_VOICE_TYPE_HUANG	12	小顽童

示例代码

```
ITMGContext.GetInstance()->GetAudioEffectCtrl()->setVoiceType(0);
```

K 歌音效特效

调用 SetKaraokeType 接口设置 K 歌音效特效。

函数原型

```
TMGAudioEffectCtrl int SetKaraokeType(int type)
```

参数	类型	意义
type	int	表示本端音频变声类型。

类型参数	参数代表	意义
ITMG_KARAOKE_TYPE_ORIGINAL	0	原声
ITMG_KARAOKE_TYPE_POP	1	流行
ITMG_KARAOKE_TYPE_ROCK	2	摇滚
ITMG_KARAOKE_TYPE_RB	3	嘻哈
ITMG_KARAOKE_TYPE_DANCE	4	舞曲
ITMG_KARAOKE_TYPE_HEAVEN	5	空灵
ITMG_KARAOKE_TYPE_TTS	6	语音合成

示例代码

```
ITMGContextGetInstance()->GetAudioEffectCtrl()->SetKaraokeType(0);
```

获取播放音效的音量

调用 GetEffectsVolume 接口获取播放音效的音量，为线性音量，默认值为100，数值大于100为增益效果，数值小于100为减益效果。

函数原型

```
ITMGAudioEffectCtrl virtual int GetEffectsVolume()
```

示例代码

```
ITMGContextGetInstance()->GetAudioEffectCtrl()->GetEffectsVolume();
```

设置播放音效的音量

调用 SetEffectsVolume 接口设置播放音效的音量。

函数原型

```
ITMGAudioEffectCtrl virtual int SetEffectsVolume(int volume)
```

参数	类型	意义
volume	int	音量数值

示例代码

```
int volume=1;
ITMGContextGetInstance()->GetAudioEffectCtrl()-
>SetEffectsVolume(volume);
```

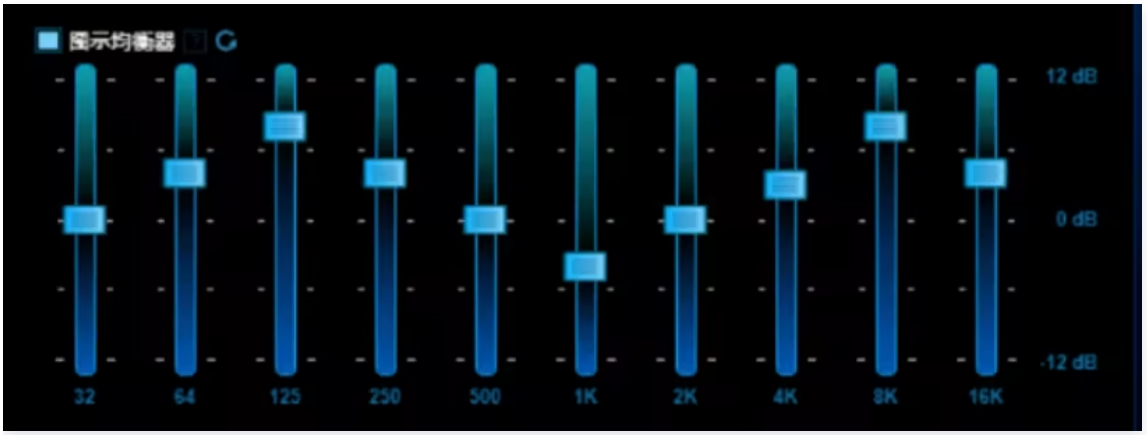
相关文档

关于变声玩法的效果请参见 实时语音趣味变声，大叔变声“妙音娘子”Get一下 。

实时语音均衡器

最近更新时间：2024-10-15 14:45:41

为方便开发者调试和接入腾讯云游戏多媒体引擎产品 API，本文为您介绍游戏多媒体引擎实时语音均衡器功能的接入指引。



使用场景

GME 均衡器功能，可以将 GME SDK 采集的音频流实时进行均衡器调整。该功能可应用于线上K歌场景，玩家开始唱歌后，调用 GME SDK 的 **均衡器** 接口，即可将玩家的实时语音流做声音美化效果方面的调整。

前提条件

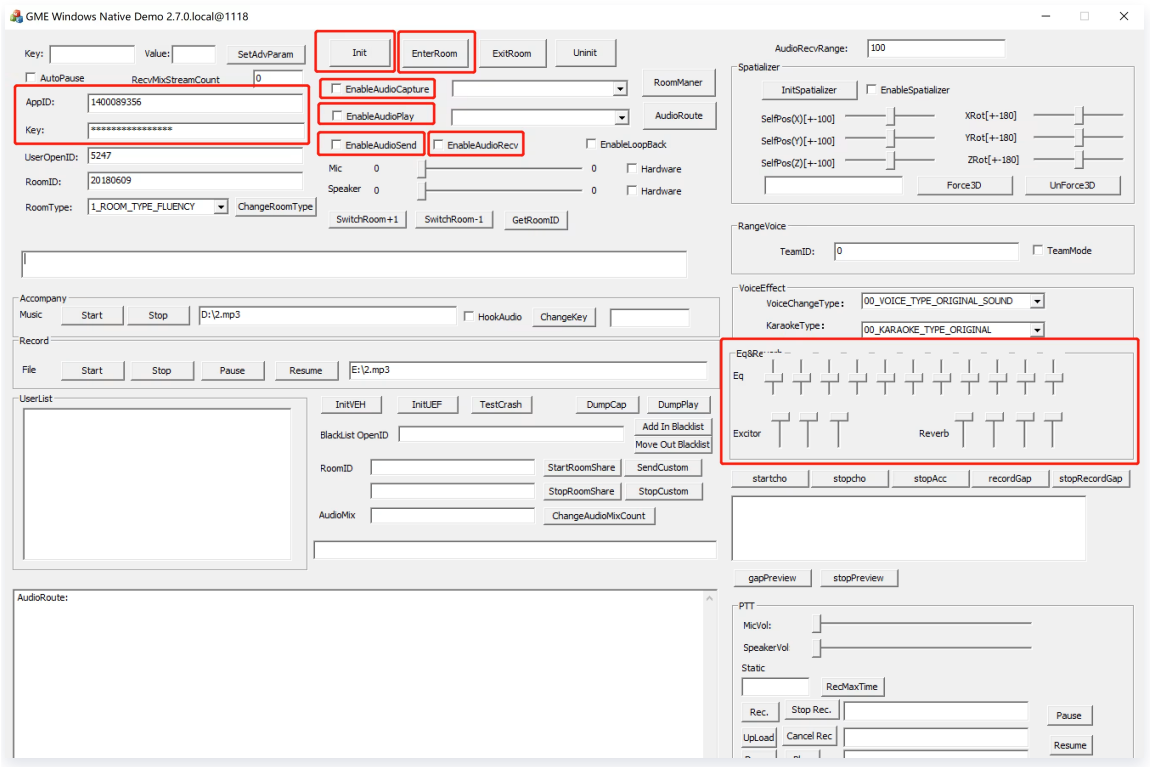
- **已开通实时语音服务**：可参见 服务开通指引 。
- **已接入 GME SDK**：包括核心接口和实时语音接口的接入，详情可参见 Native SDK 快速接入 、Unity SDK 快速接入 、Unreal SDK 快速接入 。

体验 Demo

下载 Demo

下载地址 >>

此体验 Demo 为一个 Windows 可执行程序，界面如下：



配置参数

在 Appid 和 Key 的输入框中填入自己申请的 GME AppID 以及 Key。
如果需要，您也可以输入目标房间号以及 OpenID。

使用方法

1. 进房打开麦克风扬声器流程如下：Init > EnterRoom > EnableCapture > EnablePlay > EnableSend > EnableRecv。
2. 进房成功之后，可以开启 **EnableLoopBack** 听到自己的声音。
3. 后续再调整红框中的 **EQ** 均衡器（EQ 是频段增益，Exditor 和 Reverb 是混响器）。

均衡器功能接入

使用此接口需要在进房成功的状态下，才能对本端采集的声音实现均衡器调节效果。

函数原型

```
int SetKaraokeType(ITMG_VOICE_TYPE_EQUALIZER* pEqualizer,
ITMG_VOICE_TYPE_REVERB* pReverb)
```

参数	类型	意义
pEqualizer	ITMG_VOICE_TYPE_EQUALIZER	频段增益

pReverb	ITMG_VOICE_TYPE_REVERB	涵盖 HARMONIC 和 REVERB

结构体详细信息

ITMG_VOICE_TYPE_EQUALIZER 中结构体成员为 float 类型，数值范围为-12到12。

ITMG_VOICE_TYPE_EQUALIZER	意义
EQUALIZER_32HZ	32HZ频段上施加的增益
EQUALIZER_64HZ	64HZ频段上施加的增益
EQUALIZER_128HZ	128HZ频段上施加的增益
EQUALIZER_250HZ	250HZ频段上施加的增益
EQUALIZER_500HZ	500HZ频段上施加的增益
EQUALIZER_1KHZ	1KHZ频段上施加的增益
EQUALIZER_2KHZ	2KHZ频段上施加的增益
EQUALIZER_4KHZ	4KHZ频段上施加的增益
EQUALIZER_8KHZ	8KHZ频段上施加的增益
EQUALIZER_16KHZ	16KHZ频段上施加的增益
EQUALIZER_MASTER_GAIN	整体音量

ITMG_VOICE_TYPE_REVERB 中结构体成员为 float 类型，数值范围为0到 1。

ITMG_VOICE_TYPE_REVERB
HARMONIC_GAIN
HARMONIC_START_FREQUENCY
HARMONIC_BASS_CONTROL
REVERB_SIZE
REVERB_DEPTH
REVERB_GAIN
REVERB_ECHO_DEPTH

示例代码

```
void CTMGSDK_For_AudioDlg::OnVScroll(UINT nSBCode, UINT nPos,
CScrollBar* pScrollBar)
{
    if ((CWnd*)pScrollBar == (CWnd*)&m_SliderEQ1 ||
        (CWnd*)pScrollBar == (CWnd*)&m_SliderEQ2 ||
        (CWnd*)pScrollBar == (CWnd*)&m_SliderEQ3 ||
        (CWnd*)pScrollBar == (CWnd*)&m_SliderEQ4 ||
        (CWnd*)pScrollBar == (CWnd*)&m_SliderEQ5 ||
        (CWnd*)pScrollBar == (CWnd*)&m_SliderEQ6 ||
        (CWnd*)pScrollBar == (CWnd*)&m_SliderEQ7 ||
        (CWnd*)pScrollBar == (CWnd*)&m_SliderEQ8 ||
        (CWnd*)pScrollBar == (CWnd*)&m_SliderEQ9 ||
        (CWnd*)pScrollBar == (CWnd*)&m_SliderEQ10 ||
        (CWnd*)pScrollBar == (CWnd*)&m_SliderEQ11 ||
        (CWnd*)pScrollBar == (CWnd*)&m_SliderExGain ||
        (CWnd*)pScrollBar == (CWnd*)&m_SliderExStartFrequency ||
        (CWnd*)pScrollBar == (CWnd*)&m_SliderExBaseCtrl ||
        (CWnd*)pScrollBar == (CWnd*)&m_SliderReverbSize ||
        (CWnd*)pScrollBar == (CWnd*)&m_SliderReverbDepth ||
        (CWnd*)pScrollBar == (CWnd*)&m_SliderReverbGain ||
        (CWnd*)pScrollBar == (CWnd*)&m_SliderReverbEchoDepth
        )
    {
        ITMG_VOICE_TYPE_EQUALIZER equalizer = {
            (m_SliderEQ1.GetPos() - 50)  * 24.0f / 100,
            (m_SliderEQ2.GetPos() - 50)  * 24.0f / 100,
            (m_SliderEQ3.GetPos() - 50)  * 24.0f / 100,
            (m_SliderEQ4.GetPos() - 50)  * 24.0f / 100,
            (m_SliderEQ5.GetPos() - 50)  * 24.0f / 100,
            (m_SliderEQ6.GetPos() - 50)  * 24.0f / 100,
            (m_SliderEQ7.GetPos() - 50)  * 24.0f / 100,
            (m_SliderEQ8.GetPos() - 50)  * 24.0f / 100,
            (m_SliderEQ9.GetPos() - 50)  * 24.0f / 100,
            (m_SliderEQ10.GetPos() - 50) * 24.0f / 100,
            (m_SliderEQ11.GetPos() - 50) * 24.0f / 100
        };
```

```
        ITMG_VOICE_TYPE_REVERB reverb = {
            (m_SliderExGain.GetPos())             * 1.0f / 100.0f,
            (m_SliderExStartFrequency.GetPos())   * 1.0f / 100.0f,
            (m_SliderExBaseCtrl.GetPos())         * 1.0f / 100.0f,
            (m_SliderReverbSize.GetPos())         * 1.0f / 100.0f,
            (m_SliderReverbDepth.GetPos())        * 1.0f / 100.0f,
            (m_SliderReverbGain.GetPos())         * 1.0f / 100.0f,
            (m_SliderReverbEchoDepth.GetPos())    * 1.0f / 100.0f
        };

        m_pTmgContext->GetAudioEffectCtrl()->SetKaraokeType(&equalizer,
&reverb);

    }
    CDialogEx::OnVScroll(nSBCode, nPos, pScrollBar);
}
```

均衡器使用指引

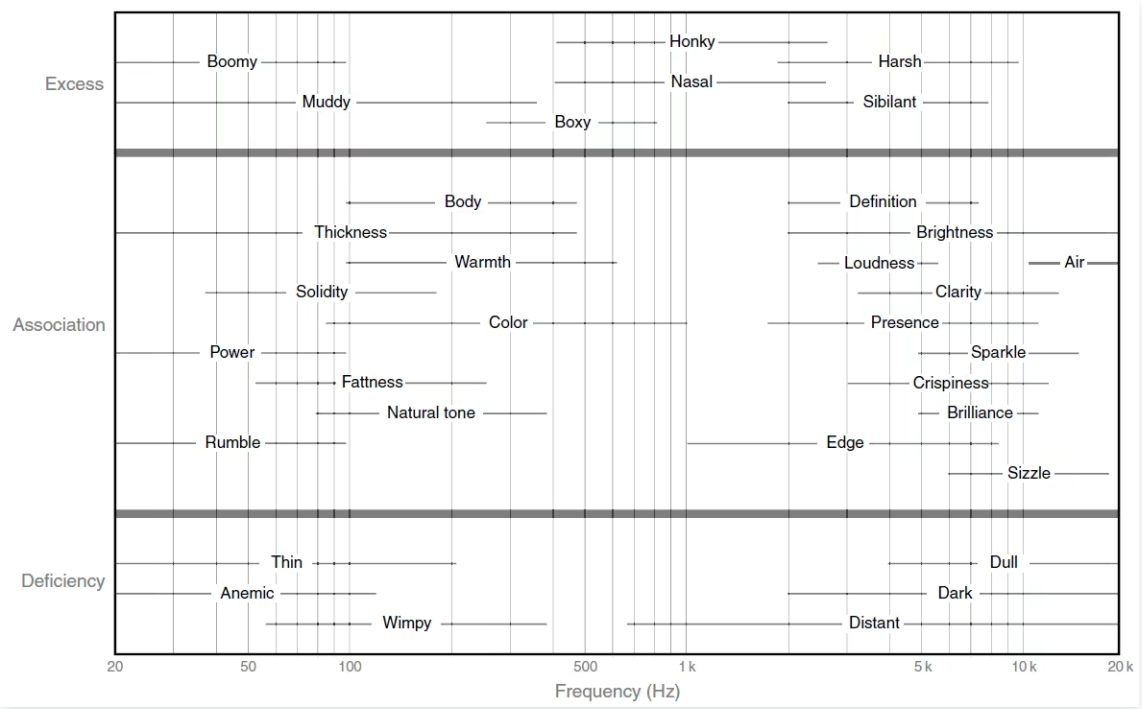
> ⚠ **注意**
> 此处仅提供指引，如果需要更加深入的使用 EQ 效果，需要专业的调音师协作。

人耳可以听到的声音范围大致为20HZ – 20KHZ，人耳对于各频段的感知呈现对数关系，因此混音工程中常将人的可听频段分成10个八度分别进行调整，调音大致分为如下几个区域：

频段	区域	说明
20HZ – 32HZ	次声和超低音区	大部分频段低于人耳听觉下限，多用触觉感知，音乐中超大型管风琴以及电影中的爆炸和雷声音效可以达到这个频率，人声无法达到这个频段，一般 VOIP 通话建议调至z最低，去除直流干扰，将信号能量留给其他频段。
32HZ – 64HZ	重低音区	主要用于调整鼓和贝斯的下潜，音调感觉不明显，部分男低音音域可达这个频段。VOIP 通话人声调音一般建议调低，去除工频干扰，将能量留给其他频段。
64HZ – 125HZ	低音区	大部分管弦乐器的基频的范围，也决定打击乐的力度。
125HZ – 250HZ	低音区	人声的基频的范围，决定人声音调感知，过重会导致声音浑浊。

250HZ – 500HZ	中低音区	包含人声音色的重要的低次谐波所在频段，可用于调整男声音色，适当增强这一段使人声音色温暖，厚重；增强过度声音变浑浊。
500HZ – 1KHZ	中音区	调整女声音色，使其更饱满，调整太高会导致鼻音重。手机等移动播放设备的共振峰处于这个频段，建议避免调整太高容易引起结构振动杂音。
1KHZ – 2KHZ	中音区	人耳较敏感区域，影响响度和临场感，可以适当增强。
2KHZ – 4KHZ	中高音区	人耳最敏感的区域，提升这一判断可以提高声音的响度，提升语音可懂度相关，调整太高会导致齿音过重。
4KHZ – 8KHZ	高音区	表现镲等高频乐器，已经弦乐器摩擦音等声音细节。决定人声的高频细节，如唇齿音，摩擦音等，一般不建议提升。
8KHZ – 16KHZ	超高音和超声区	乐器的高频泛音，调整对语音作用不大。

具体调整也可以参考下图：



Windows 平台播放器伴奏

最近更新时间：2025-06-18 11:29:11

为方便开发者调试和接入腾讯云游戏多媒体引擎产品，本文主要为您介绍适用于使用 Windows 平台第三方播放器伴奏功能的相关技术。

前提条件

- 已完成 GME 应用创建，并获取 SDK AppID 和 Key。请参考 服务开通指引 。
- 已开通 **GME 实时语音服务**。请参考 服务开通指引 。
- GME 使用前请对工程进行配置，否则 SDK 不生效。

配置头文件

1. 请参考 配置文档 ，完成 GME 工程配置。
2. 将文件 tmg_adv_win.h 放置在 GME SDK（For Windows）下，与其他头文件保持同一目录。
3. 单击 下载 ，下载两个 dll 文件，路径与其他库文件一致。导出可执行文件后，应与可执行文件同目录。

调用接口

开始播放

调用此参数，将挂钩整个系统的声音，或者某播放器的声音。

函数原型

```
virtual int StartAccompany(const char* playerPath, int playerPathLength,
const char* mediaFilePath, int mediaFilePathLenght,
GMEAccompany_SourceType sourceType) = 0
```

参数	类型	含义
playerPath	const char*	播放器路径
playerPathLength	int	播放器路径长度
mediaFilePath	const char*	音频资源路径，1. 某音频文件，2. 音频文件夹，可以不传（NULL）
mediaFilePathLenght	int	音频资源路径长度

sourceType	GMEAccompany_SourceType	设置采集对象，详细见下表 sourceType 说明

sourceType 说明：

参数	类型
AV_ACCOMPANY_SOURCE_TYPE_NONE = 0	代表不开启。
AV_ACCOMPANY_SOURCE_TYPE_SYSTEM = 1	代表挂钩整个系统的声音，此时可以不传音频资源路径，但需要传一个已知文件路径作为播放器路径参数及播放器路径长度参数。
AV_ACCOMPANY_SOURCE_TYPE_PROCESS = 2	代表挂钩某进程的声音，例如 QQ 音乐播放器。

示例代码

```
const char* file = "C:\\1.txt";//当挂钩系统声音时，需要传入一个已知文件路径，请确保此文件存在
int ret = ITMGAdcanceGetInstance()->StartAccompany(file, strlen(file), NULL, 0, AV_ACCOMPANY_SOURCE_TYPE_SYSTEM);
```

停止播放

调用此参数，将停止挂钩。

函数原型

```
virtual int StopAccompany() = 0
```

设置声音音量

调用此参数，将设置挂钩的声音音量大小，数值100表示音量不增大也不衰减。数值范围为0到200。

函数原型

```
virtual int SetAccompanyVolume(int value) = 0;
```

获取声音音量

调用此参数，将获取挂钩的声音音量大小。

函数原型

```
virtual int GetAccompanyVolume(int* pVolume) = 0;
```

获取实时声音音量

调用此参数，将获取挂钩的实时声音音量，可以用来展示实时音量能量条。

函数原型

```
virtual int GetAccompanyVolumeDynamic(int* pVolume) = 0;
```

设置采集设备音量

调用此参数，设置采集设备声音音量，默认数值为100，数值范围为0到100，0代表静音。

函数原型

```
virtual int SetMicDeviceVolume(int vol) = 0;
```

获取采集设备音量

调用此参数，将获取采集设备声音音量。

函数原型

```
virtual int GetMicDeviceVolume() = 0;
```

设置播放设备音量

调用此参数，设置播放设备声音音量，默认数值为100，数值范围为0到100，0代表静音。

函数原型

```
virtual int SetSpeakerDeviceVolume(int vol) = 0;
```

获取播放设备音量

调用此参数，将获取播放设备声音音量。

函数原型

```
virtual int GetSpeakerDeviceVolume() = 0;
```

语音内容安全
实时语音审核

最近更新时间：2025-06-18 11:49:31

为方便开发者快速接入 **GME 语音内容安全服务**，这里向您介绍GME语音内容安全服务在**实时语音场景**的接入流程。

适用场景

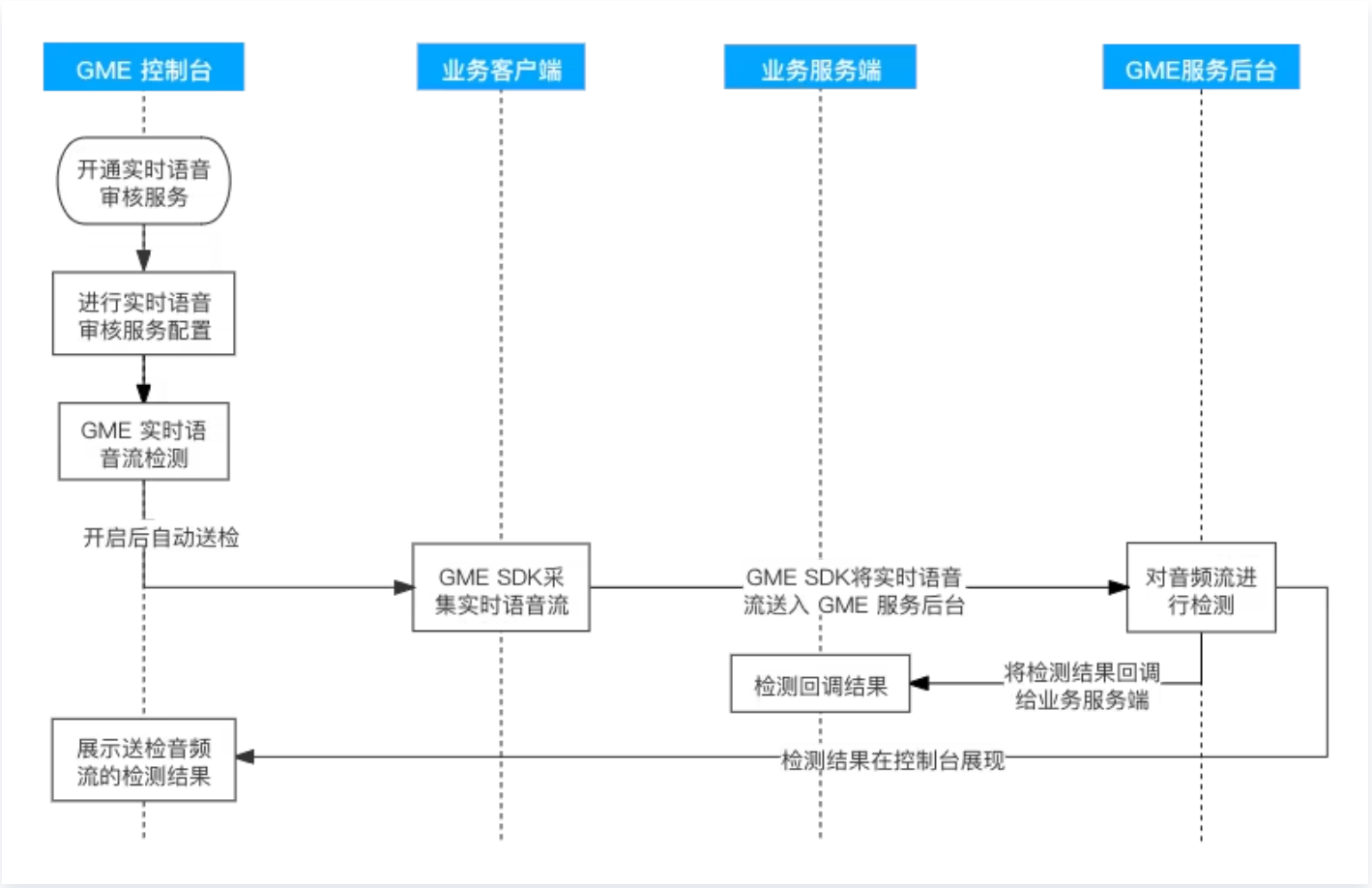
适用于已接入 GME SDK 并开通 实时语音服务 ，需要对实时语音流进行审核的场景。审核覆盖谩骂、涉黄、广告等各类违规内容，保障和谐健康的游戏环境。

> **说明：**
> **语音内容安全服务** 目前仅支持中文/英文音频（呻吟声不区分语种）。

前提条件

- 已完成GME应用创建，并获取 SDK AppID 和 Key。请参考 服务开通指引 。
- 已开通 **GME 实时语音服务**。请参考 服务开通指引 。
- 已完成 **GME SDK 接入**。请参考 SDK 快速接入 。

业务流程



接入流程

实时语音审核接入步骤如下：

1. 开通服务
2. 服务配置
3. 使用实时语音服务
4. 处理审核回调结果
5. 查看检测结果
6. 查看用量统计

步骤1：开通服务

对已创建的应用开通**语音内容安全**服务，详情请参见 服务开通指引 。

> **注意**
> **GME 语音内容安全** 服务会根据实际用量产生费用。计费详情请参考 购买指南 。

步骤2：服务配置

开通**语音内容安全**服务后，在控制台进行**语音内容安全**服务配置。详情请参见 语音内容安全服务配置指引 。

步骤3：使用实时语音服务

在完成服务配置后，当您使用 GME 实时语音服务并产生上行语音流时，GME 将根据您设置的规则，自动将语音流送检。GME SDK 的调用和接入请参见 Native SDK 快速接入 、 Unity SDK 快速接入 、 Unreal SDK 快速接入 。

> **注意**
> 为了帮助您节约成本，当实时语音房间内只有一个用户时，音频流不会送检。

步骤4：处理审核回调结果

当有违规内容命中时，**GME 语音内容安全服务** 将向您配置的业务服务端回调审核结果。业务服务端接收到检测的回调结果后，需要根据业务逻辑对回调结果进行后续处理。回调详情请参见 语音内容安全回调说明 。

> **注意**
> - GME 默认仅回调违规结果。若您需要获取全量审核结果回调，请 提交工单 申请。
> - GME 帮助您的业务识别违规内容，但不为您进行违规内容的处置。

步骤5：控制台查看检测结果

您可以在 控制台 > 检测结果 中可查看送检语音的检测结果。

步骤6：查看用量统计

您可以在 控制台 查看应用的用量统计，在所需查询的应用中单击**用量统计**。详情请参见 控制台指引 > 用量查看 。

语音消息审核

最近更新时间：2025-06-18 14:16:01

为方便开发者快速接入 **GME 语音内容安全服务**，本文向您介绍 GME 语音内容安全服务在**语音消息审核场景**的接入流程。

适用场景

适用于已接入 GME SDK 并开通**语音消息服务**，需要对语音消息进行审核的场景。在使用 **GME 语音消息** 服务的基础上，自动审核识别出语音消息音频中的敏感违规内容，即可识别出呻吟/娇喘/违规歌曲等敏感音频，也可基于音频中的文本进行涉黄/广告/谩骂等内容进行关键词识别。GME 将识别出的语音消息敏感违规内容通过客户端或服务端回调的形式通知业务，由业务自行进行合规处理。GME 语音消息审核服务仅支持以 AppID 为维度 **全量开启/关闭**，不支持指定送审的用户 ID 或房间 ID。

> **说明：**
> **语音内容安全服务** 目前仅支持中文/英文音频（呻吟声不区分语种）。

前提条件

- 已完成 GME 应用创建，并获取 SDK AppID 和 Key。详情请参见 服务开通指引 。
- 已开通 **GME 语音消息服务**。详情请参见 服务开通指引 。
- 已完成 **GME SDK 接入**。详情请参见 SDK 快速接入 。
- 已接入 **GME 2.8.4** 以上版本 SDK。

接入流程

语音消息审核功能，需要业务接收审核结果并自行处理。可通过两种路径进行接收和处理：**客户端处理**、**游戏后台处理**。

- 客户端处理，相当于在游戏的客户端下载语音文件的时候，或者转文本的时候，GME会将审核结果返回给游戏客户端，游戏客户端根据审核结果决定**是否播放该语音**或者**展示该语音识别出来的文字**。
- 游戏后台处理，即游戏客户端发送语音消息后，游戏后台收到该语音内容的审核结果回调。游戏后台需要根据审核结果判断是否将当前语音内容转发到接收端。

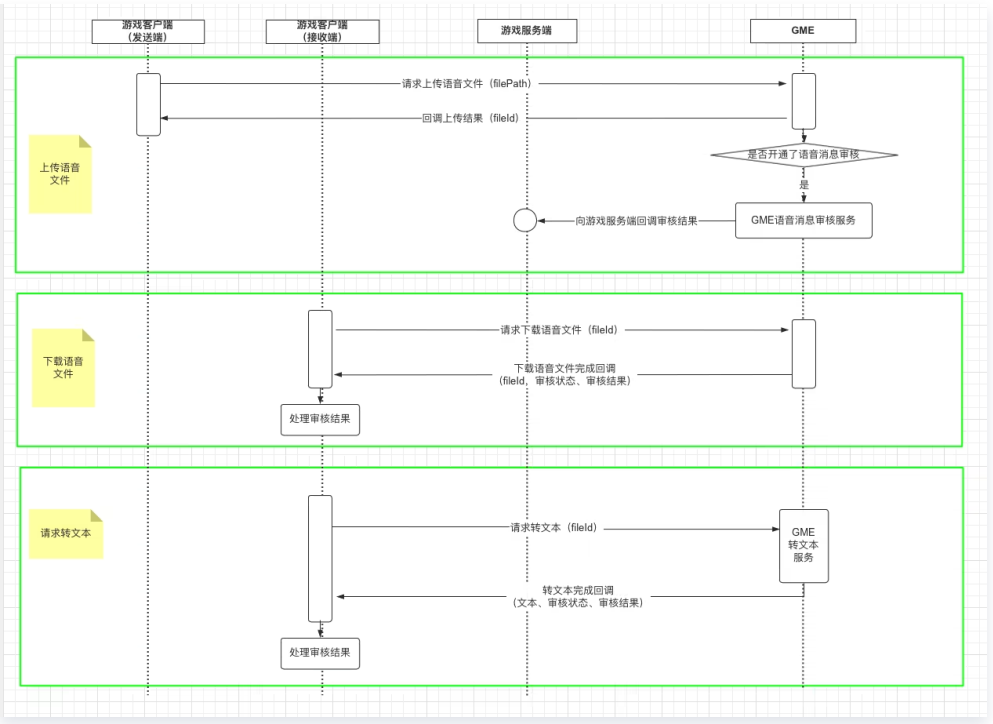
场景一：录音文件转文本

场景描述
发送端录制语音消息并上传语音文件，接收端下载语音文件音频或为语音文件请求转文本。常用于接收端需要默认接收音频消息，并请求进行转文本展示的场景。
涉及接口

- UploadRecordedFile（上传语音文件）
- DownloadRecordedFile（下载语音文件）
- SpeechToText（语音转文本）

业务流程

1. 在上传语音文件阶段，如果游戏服务端接收到审核结果，可以根据审核结果判断是否转发语音 fileid 到接收端。
2. 在下载语音文件以及请求转文本阶段，在收到审核结果后，客户端决定是否下载或者在业务层展示语音文件及文本。



注意

语音消息的审核耗时取决于音频文件的长短，一般一段长度为5s的音频，审核耗时约为2秒。如果请求下载音频文件或请求转文本时，GME 尚未完成音频文件审核，回调的审核状态将表现为“审核中”。此时业务可根据需要，自行决定是否允许音频播放或文本展示；建议业务接收端在获取到审核完成的音频前，每隔1秒轮询请求下载语音文件或转文本接口，短时间内的轮询请求转文本接口不会重复计算费用。

场景二：语音消息流式转文本（语音及文本发送）

场景描述

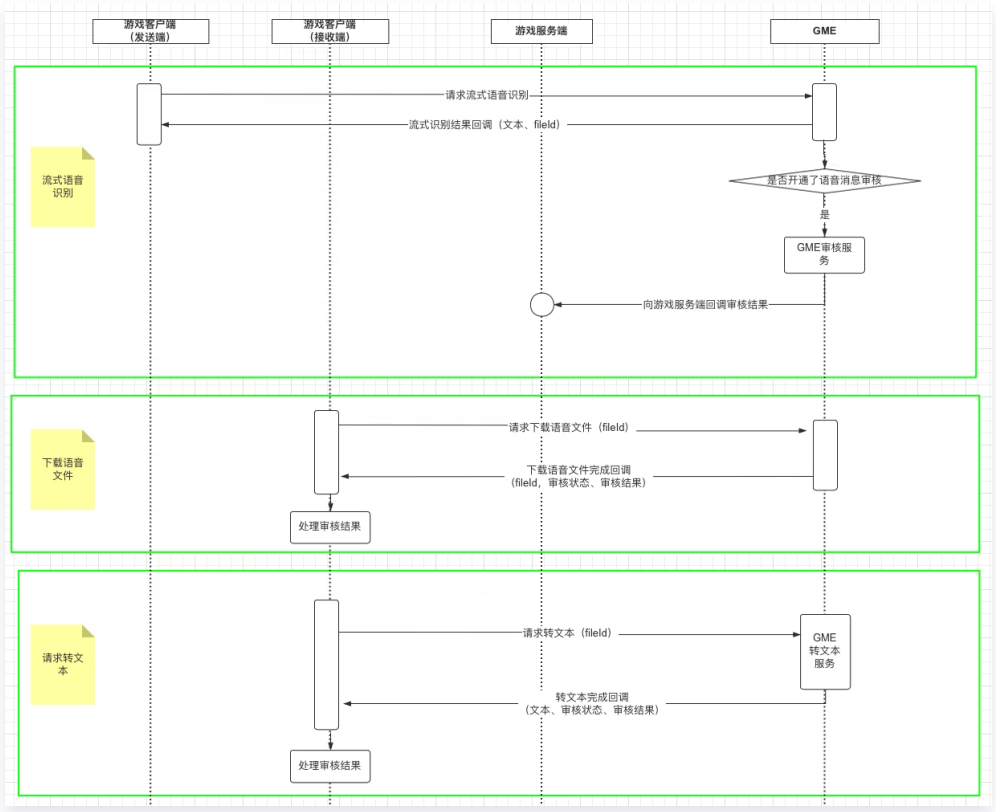
发送端使用语音消息流式转文本服务，接收端下载语音文件音频或为语音文件请求转文本。常用于接收端需要默认接收音频消息并默认展示转文本结果的场景。

涉及接口

- StartRecordingWithStreamingRecognition（语音消息流式转文本）
- DownloadRecordedFile（下载语音文件）

业务流程

1. 在流式语音识别阶段，如果游戏服务端接收到审核结果，可以根据审核结果判断是否转发语音 fileid 到接收端。
2. 在下载语音文件以及请求转文本阶段，在收到审核结果后，客户端决定是否下载或者在业务层展示语音文件及文本。



场景三：语音消息流式转文本（只发送文本）

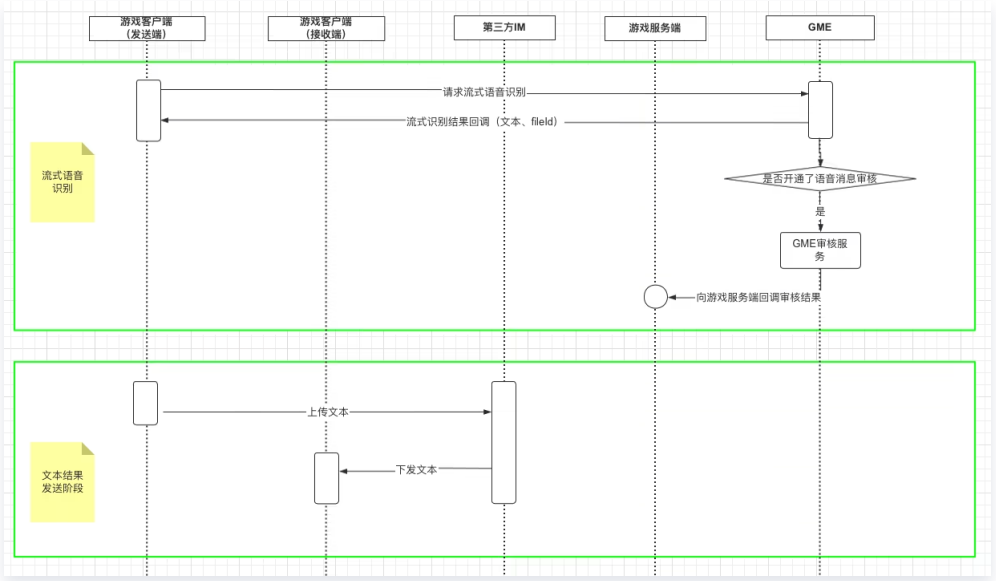
场景描述

发送端流式语音识别，通过外部信令通道传输识别文本结果。接收端不向 GME 发起请求。常用于接收端不需要播放音频的场景。

涉及接口

StartRecordingWithStreamingRecognition（语音消息流式转文本）

业务流程



> **注意**
> 此场景下，GME 无法将审核结果通知业务客户端，需要业务在游戏服务端自行处置审核结果。若业务本身已接入第三方文本审核能力，建议关闭 GME 语音消息审核服务。

接入步骤

步骤1：服务开通与服务配置

基于语音消息场景使用 **GME 语音内容安全** 服务，暂不支持通过控制台自助服务开通和配置。请 提交工单 联系 GME 团队申请开通和配置。

> **注意**
> **GME 语音内容安全** 服务会根据实际用量产生费用。计费详情请参考 购买指南 。

步骤2：使用语音消息服务以及语音消息转文本服务

在完成服务配置后，当您发送语音消息或请求将语音消息转文本时，GME 会自动将消息内容送检。GME SDK 的调用和接入请参见 Native SDK 快速接入 、 Unity SDK 快速接入 、 Unreal SDK 快速接入 。

步骤3：在游戏客户端增加审核结果处置逻辑

可在游戏客户端增加下述逻辑：

- 基于 label 判断是否禁止当前音频文件播放。
- 基于命中的违规关键词对转文本结果进行内容替换或屏蔽。
- 对于“审核中”的回调结果增加轮询请求逻辑，直至审核状态变更为“审核完成”。轮询间隔不小于1秒。

步骤4：在游戏服务端增加审核结果处置逻辑（建议）

可在游戏服务端增加下述逻辑：基于 label 和命中的违规关键词，对玩家进行分级警告或处罚，净化游戏文明环境。

回调说明

客户端的审核结果，通过下载语音完成回调、转文本完成回调中的参数进行通知；服务端的审核结果，通过服务端审核结果回调中的参数进行通知。详情请参见 语音消息审核回调说明 。

第三方语音流或音频文件审核

最近更新时间：2025-06-18 14:16:01

为方便开发者快速接入GME语音内容安全服务，本文为您介绍GME语音内容安全服务在第三方语音流审核或音频文件审核场景的接入流程。

适用场景

适用于需要对第三方实时语音流或者第三方音频文件进行审核的场景。审核覆盖谩骂、涉黄、广告等各类违规内容，保障和谐健康的游戏环境。

> **说明：**
> **语音内容安全服务** 目前仅支持中文/英文音频（呻吟声不区分语种）。

业务流程

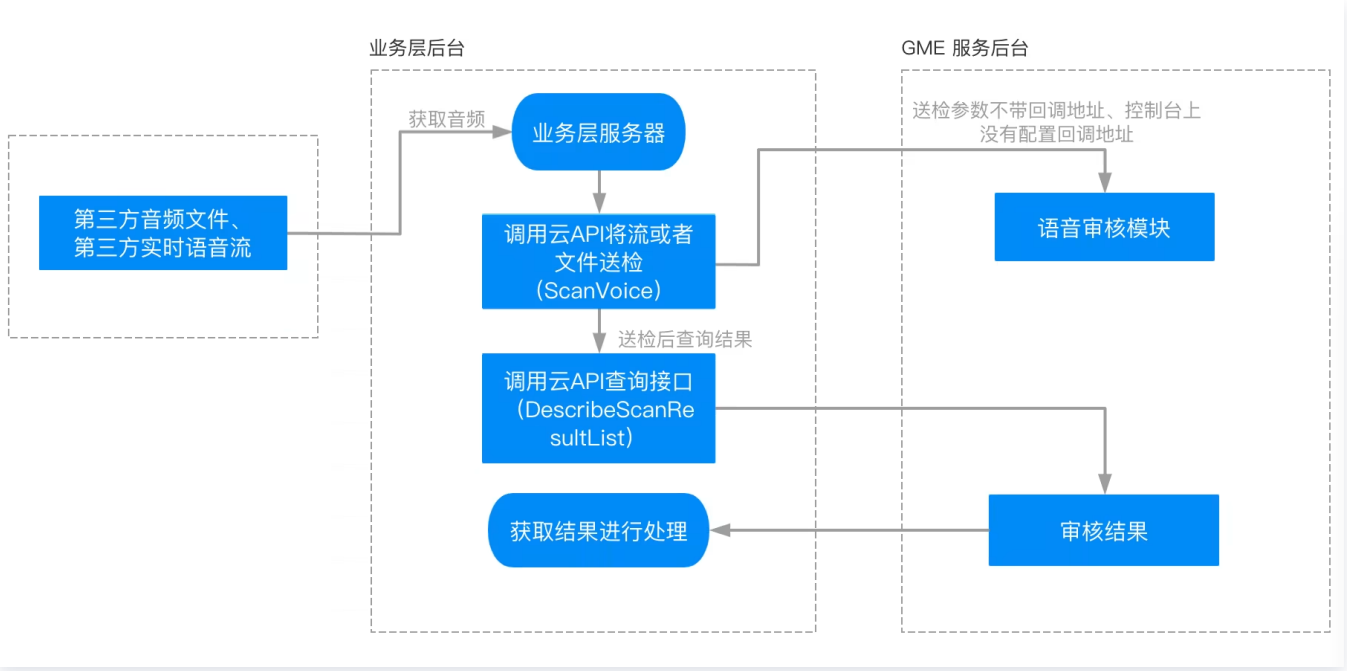
异步查询送审结果

场景描述

将第三方实时语音流或者第三方音频文件，通过业务侧后台调用 GME 云 API 进行送审，如果在送审接口中没有回调地址参数，以及控制台上没有配置回调地址，则需要调用查询接口，轮询审核结果。

涉及接口 ScanVoice（提交语音检测任务）、DescribeScanResultList（查询语音检测结果）

业务流程



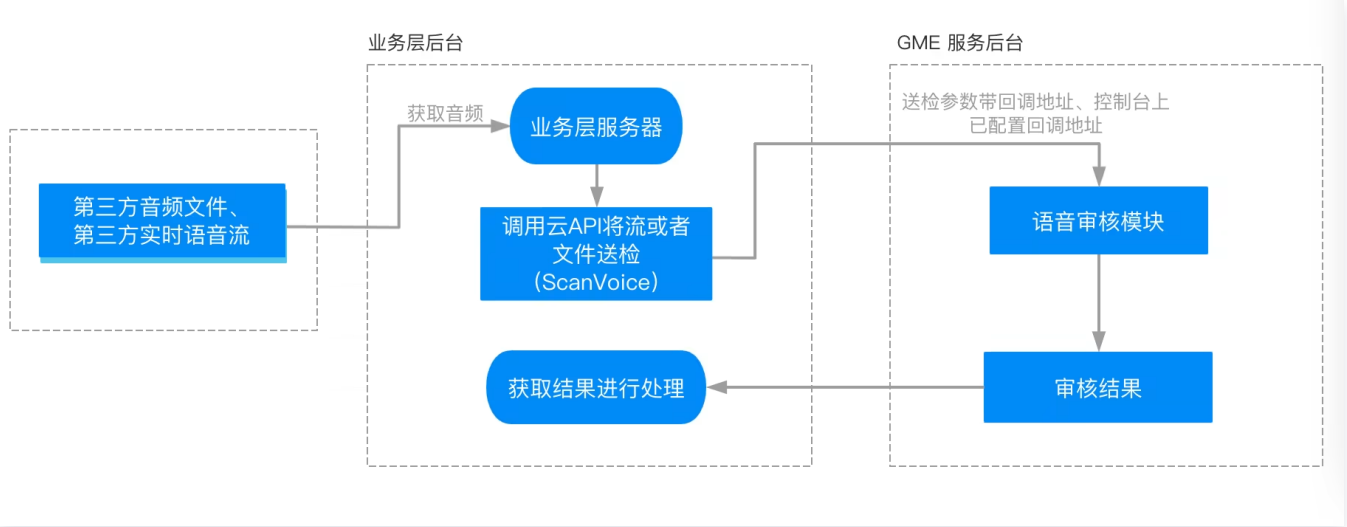
实时获取送审结果

场景描述

将第三方实时语音流或者第三方音频文件，通过业务侧后台调用 GME 云 API 进行送审，如果在送审接口中有地址参数，或者控制台上已经配置回调地址，则审核结果通过回调地址进行返回。（如果调用送审接口中带有回调地址参数，则优先使用此回调地址，否则使用控制台已配置的回调地址）

涉及接口 ScanVoice（提交语音检测任务）

业务流程



接入步骤

1. 开通服务
2. 服务配置
3. 调用接口提交检测任务
4. 通过调用接口获取检测结果（可选）
5. 通过接收回调获取检测结果（可选）
6. 处理检测结果
7. 查看检测结果（可选）
8. 查看用量统计（可选）

步骤1：开通服务

- 对已创建的应用开通**语音内容安全**服务，详情请参见 服务开通指引 。
- 如未创建应用，需要登录 控制台 > 新建应用 ，详情请参见 服务开通 > 新建应用 。

> ⚠ 注意

> **GME 语音内容安全** 服务会根据实际用量产生费用。计费详情请参考 购买指南 。

步骤2：服务配置

开通**语音内容安全**服务后，在控制台进行**语音内容安全**服务配置。详情请参见 语音内容安全服务配置指引 。

步骤3：调用接口提交检测任务

使用第三方语音流或音频文件审核服务，业务服务端可直接调用第三方语音流或音频文件审核的云 API提交检测任务 。接口详情请参见 提交语音检测任务（ScanVoice） 。

步骤4：通过调用接口查询检测结果（可选）

提交检测任务后，业务服务端可主动请求接口查询检测结果。查询接口详情请参见 查询语音检测结果（DescribeScanResultList） 。

步骤5：通过接收回调获取检测结果（可选）

提交检测任务后，若您在控制台或在提交检测任务时已配置回调地址，GME将对该地址进行检测结果回调。默认情况下，仅针对违规内容进行回调。

步骤6：根据业务场景需求，对违规结果进行后续处理

GME 语音内容安全服务 不会为业务进行结果处置，需要业务方通过步骤4或步骤5的方式获取检测结果，自行对违规内容进行业务逻辑处理.

步骤7：控制台查看检测结果（可选）

您可以在 控制台 > 检测结果 中可查看送检语音的检测结果。

步骤8：查看用量统计（可选）

您可以在 控制台 查看应用的用量统计，在所需查询的应用中单击**用量统计**。详情请参见 控制台指引 > 用量查看 。

回调说明

最近更新时间：2025-08-01 15:26:42

实时语音审核、第三方语音流及音频文件审核回调说明

实时语音审核 和 第三方语音流或音频文件审核 的回调均为服务端回调，且回调参数一致，详情如下文所述。

网络协议

如果在请求参数中指定了回调地址参数 Callback，即一个 HTTP(S) 协议接口的 URL，则需要支持 POST 方法，传输数据编码采用 UTF-8。

URL Query 参数

名称	类型	是否必需	描述
Signature	string	是	签名，具体见下方 签名生成 说明。

签名生成

Signature = Base64 (HMAC-SH1 (strContent, SecretKey))

- **strContent**：签名原文串，为 POST+body 的整个 JSON 内容（长度以 Content-Length 为准）。
- **body**：回调给业务的 JSON 内容，下方 回调示例 中的全部内容即为 body。
- **SecretKey**：密钥，为应用的 SecretKey，可通过 控制台 > 应用详情 查看。
- **HMAC-SH1**：签名算法。
- **Base64**：加密函数，进行 Base64 编码后，即可生成 Signature 签名。
- 将 Signature 加到 url 的参数里，即可生成最终的回调请求。

回调示例

```
{
    "Code": 0,
    "DataId": "1400000000_test_data_id",
    "ScanFinishTime": 1566720906,
    "HitFlag": true,
    "Live": false,
    "Msg": "",
    "ScanPiece": [{
        "DumpUrl": "",
        "HitFlag": true,
        "MainType": "abuse",
```

```
        "RoomId": "123",
        "OpenId": "1111",
        "Info":"",
        "Offset": 0,
        "Duration": 3400,
        "PieceStartTime":1574684231,
        "SubLabel": "PersonalAttack",
        "ScanDetail": [{
            "EndTime": 1110,
            "KeyWord": "违规字",
            "Label": "abuse",
            "Rate": "90.00",
            "StartTime": 1110
        }, {
            "EndTime": 1380,
            "KeyWord": "违规字",
            "Label": "abuse",
            "Rate": "90.00",
            "StartTime": 930
        }, {
            "EndTime": 1560,
            "KeyWord": "违规字",
            "Label": "abuse",
            "Rate": "90.00",
            "StartTime": 930
        }, {
            "EndTime": 2820,
            "KeyWord": "违规字",
            "Label": "abuse",
            "Rate": "90.00",
            "StartTime": 2490
        }]
    }],
    "ScanStartTime": 1566720905,
    "Scenes": [
        "default"
    ],
    "Status": "Success",
    "TaskId": "6330xxxx-9xx7-11ed-98e3-52xxxxe4ac3b",
    "Url": "https://xxx/xxx.m4a"
```

```
}
```

> **说明：**
> - 回调详细字段说明见结构：DescribeScanResult 。
> - 默认接口请求频率限制：1000次/秒。

语音消息审核回调说明

语音消息审核回调中，**下载语音完成回调**、**转文本完成回调**为客户端回调，在客户端接口中，除会返回原先的 code、filepath、fileid 外，还会增加 auditResult 的回调数据。**审核结果回调**为服务端回调。

客户端回调示例

```
FileDownloadDelegateImpl filedAudioDownloadDelegate = new
FileDownloadDelegateImpl(delegate (int code, string filepath, string
fileid, string auditResult) {
            showLoadingView(false, "");
            if (code == 0)
            {
                showWarningText("下载成功");
                InputField field =
transform.Find("downloadFilePath").GetComponent<InputField>();
                field.text = filepath;
                InputField audioInput =
transform.Find("auditResult").GetComponent<InputField>();
                audioInput.text = auditResult;

            }
            else
            {
                showWarningText("下载失败" + Convert.ToString(code));
                InputField field =
transform.Find("downloadFilePath").GetComponent<InputField>();
                field.text = "";
                InputField audioInput =
transform.Find("auditResult").GetComponent<InputField>();
                audioInput.text = auditResult;
            }
```

```
        });
```

语音消息审核回调参数

名称	类型	描述
user_Id	Integer	业务方用户 ID
audit_res	Integer	审核状态，对应如下： 1：审核完成 2：审核中 3：语言不支持 4：没有勾选该语言送
DataId	string	数据唯一标识
HitFlag	Boolean	是否违规：true 违规，false 不违规
Label	string	审核结果标签，对应如下： normal：正常文本 politics： 政治 porn：色情 abuse：谩骂 ad：广告 customized：自定义词库 illegal：违法 terrorism: 暴恐 moan: 呻吟/娇喘
AsrText	string	语音转文本结果
ScanDetail	Array of ScanDetail	包含语音消息的审核结果和违规内容
RequestId	Integer	唯一请求ID，用于跟踪查询问题

ScanDetail 数据结构

名称	类型	描述
Label	String	审核结果标签
Rate	String	机器判断当前违规类型的置信度，取值范围：0~100。分数越高，表示越有可能属于当前的违规类型
KeyWords	Array of String	违规关键词

StartTime	Integer	违规事件开始时间，单位为毫秒（ms）
EndTime	Integer	违规事件结束时间，单位为毫秒（ms）

审核结果回调示例


```
{
    "user_id": 3333XXXX112221,
    "audit_res": 1,
    "DataId": "D3667B11-XXXX-XXXX-XXXX-14699D372EBB",
    "HitFlag": true,
    "Label": "abuse",
    "AsrText": "识别文本",
    "ScanDetail": [
        {
            "Label": "abuse",
            "Rate": "90",
            "KeyWord": [
                "违规字",
                "违规字"
            ]
        }
    ],
    "RequestId": "fe656b61-XXXX-XXXX-XXXX-b0e7ad972656"
}
```

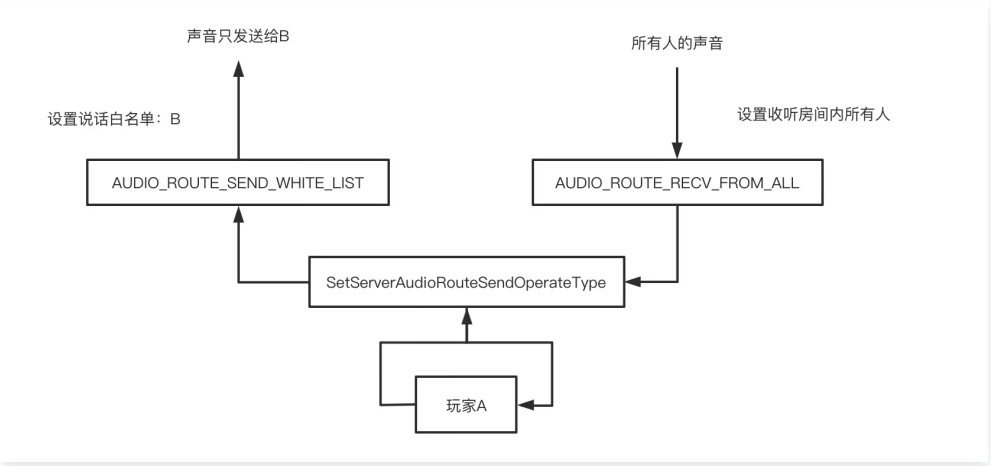
网络音频流转发路由

最近更新时间：2024-10-15 14:45:41

为方便 GME 开发者调试和接入腾讯云游戏多媒体引擎产品 API，本文为您介绍适用于 GME 自定义音频转发路由功能的使用参考文档。

使用场景

场景描述：2个好友组小队后，匹配了3个路人组了一个大队伍，需要实现听全队的声音，但只和小队的好友说话。
自定义音频路由功能可以实现。这里5个人都进入同一个语音房间，然后通过音频路由的接口设置，玩家可以设置只听见2人队伍的声音或者全房间的声音，也可以设置为说话只被2人队伍的人听到或者全房间的人听到。
音频规则距离：SetServerAudioRouteSendOperateType(AUDIO_ROUTE_SEND_WHITE_LIST,"2人队伍的list",ITMG_SERVER_AUDIO_ROUTE_RECV_TYPE,"2人队伍的list");
这样声音就只发送给list里面的人，且只接收3人队伍的声音。



前提条件

- **已开通实时语音服务**：可参见 服务开通指引 。
- **已接入 GME SDK**：包括核心接口和实时语音接口的接入，详情可参见 Native SDK 快速接入 、Unity SDK 快速接入 、Unreal SDK 快速接入 。
- 使用 GME 实时语音功能成功进入语音房间，并且打开了麦克风（EnableMic）、扬声器（EnableSpeaker）。

接入音频转发路由功能

设置音频转发规则

调用此接口设置音频转发规则，此接口在进房成功回调中调用，调用后此次进房生效，退房后失效。

> ⚠ **注意：**

> 禁言功能 AddBlackList 为本端生效，优先级高于自定义音频路由。例如 A 通过 SetServerAudioRouteSendOperateType 设置了只听 B 说话，但是又调用了 AddBlackList 将B 禁言，此时 A 将听不到 B 的声音。

接口原型

Unity

```
public abstract class ITMGRoom{
    public abstract int
SetServerAudioRouteSendOperateType(ITMG_SERVER_AUDIO_ROUTE_SEND_TYPE
Sendtype, string[] OpenIDforSend, ITMG_SERVER_AUDIO_ROUTE_RECV_TYPE
Recvtype, string[] OpenIDforRecv);
}
```

C++

```
virtual int SetServerAudioRoute(ITMG_SERVER_AUDIO_ROUTE_SEND_TYPE
SendType, const char OpenIDforSend[][21], int OpenIDforSendSize,
ITMG_SERVER_AUDIO_ROUTE_RECV_TYPE RecvType,const char OpenIDforRecv[]
[21], int OpenIDforRecvSize) = 0;
```

Android

```
public abstract int
SetServerAudioRoute(ITMGContext.ITMG_SERVER_AUDIO_ROUTE_SEND_TYPE
sendType, ArrayList<String> SendList,
ITMGContext.ITMG_SERVER_AUDIO_ROUTE_RECV_TYPE recvType,
ArrayList<String> RecvList);
```

iOS

```
-(int)SetServerAudioRouteSendOperateType:
(ITMG_SERVER_AUDIO_ROUTE_SEND_TYPE) Sendtype  SendList:(NSArray
*)OpenIDForSend  RecvOperateType:(ITMG_SERVER_AUDIO_ROUTE_RECV_TYPE)
Recvtype RecvList:(NSArray *)OpenIDForRecv;
```

类型说明

ITMG_SERVER_AUDIO_ROUTE_SEND_TYPE

设置发送音频规则，不同的规则填入后，会有不同的发送规则。

接收类型	效果
AUDIO_ROUTE_NOT_SEND_TO_ANYONE	本端音频上行发送到后台，但后台不转发给任何人，相当于将自己静音，此时参数 OpenIDForSend 无效，只需填 null
AUDIO_ROUTE_SEND_TO_ALL	本端音频上行将转发给所有人，此时参数 OpenIDForSend 无效，只需填 null
AUDIO_ROUTE_SEND_BLACK_LIST	本端音频上行将不转发给黑名单的人，黑名单由参数 OpenIDForSend 提供
AUDIO_ROUTE_SEND_WHITE_LIST	本端音频上行将只转发给白名单的人，白名单由参数 OpenIDForSend 提供

说明

- 如果类型传入 AUDIO_ROUTE_NOT_SEND_TO_ANYONE 以及 AUDIO_ROUTE_SEND_TO_ALL ，此时的参数 OpenIDForSend 不生效，只需要填 null。
- 如果类型传入 AUDIO_ROUTE_SEND_BLACK_LIST ，此时参数 OpenIDForSend 为黑名单列表，最多支持 10 个。
- 如果类型传入 AUDIO_ROUTE_SEND_WHITE_LIST ，此时参数 OpenIDForSend 为白名单列表，最多支持 10 个。

ITMG_SERVER_AUDIO_ROUTE_RECV_TYPE

设置接收音频规则，不同的规则填入后，会有不同的接收规则。

接收类型	效果
AUDIO_ROUTE_NOT_RECV_FROM_ANYONE	本端不接受任何音频，相当于关闭房间内扬声器效果，此时参数 OpenIDforRecv 无效，只需填 null
AUDIO_ROUTE_RECV_FROM_ALL	本端接收所有人的音频，此时参数 OpenIDforRecv 无效，只需填 null
AUDIO_ROUTE_RECV_BLACK_LIST	本端不接收黑名单的人的音频声音，黑名单由参数 OpenIDforRecv 提供
AUDIO_ROUTE_RECV_WHITE_LIST	本端只接收白名单的人的音频声音，白名单由参数 OpenIDforRecv 提供

说明

- 如果类型传入 AUDIO_ROUTE_NOT_RECV_FROM_ANYONE 以及 AUDIO_ROUTE_RECV_FROM_ALL OpenIDForSend 不生效。
- 如果类型传入 AUDIO_ROUTE_RECV_BLACK_LIST ，此时参数 OpenIDForSend 为黑名单列表，最多支持 10 个。
- 如果类型传入 AUDIO_ROUTE_RECV_WHITE_LIST ，此时参数 OpenIDForSend 为白名单列表，最多支持 10 个。

返回值

接口返回值为 QAV_OK 则表示成功。

- 若回调返回 1004 表示参数错误，建议重新检查参数是否正确。
- 若回调返回 1001 表示重复操作。
- 若回调返回 1201 表示房间不存在，建议检查房间号是否正确。
- 若回调返回 10001 以及 1005 ，建议重新调用接口一次。

更多返回结果解释详情请参见 错误码 。

示例代码

执行语句

```
@synthesize _sendListArray;
@synthesize _recvListArray;

int ret =  [[[ITMGContext GetInstance] GetRoom]
SetServerAudioRouteSendOperateType:SendType  SendList:_sendListArray
RecvOperateType:RecvType RecvList:_recvListArray];
if (ret != QAV_OK) {
    UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"更新
audioroute列表失败" message:[NSString stringWithFormat:@"错误码:%d",ret]
delegate:NULL cancelButtonTitle:@"OK" otherButtonTitles:nil];
    [alert show];
}
```

回调

```
-(void)OnEvent:(ITMG_MAIN_EVENT_TYPE)eventType data:(NSDictionary
*)data{
```

```
    NSString* log =[NSString stringWithFormat:@"OnEvent:%d,data:%@",
(int)eventType, data];
    switch (eventType) {
                case ITMG_MAIN_EVENT_TYPE_SERVER_AUDIO_ROUTE_EVENT:{
            {
                UIAlertView *alert = [[UIAlertView alloc]
initWithTitle:@"更新audioroute" message:[NSString stringWithFormat:@"结
果:%@,sub_type: %@ errorinof: %@",
data[@"result"],data[@"sub_type"],data[@"error_info"]] delegate:NULL
cancelButtonTitle:@"OK" otherButtonTitles:nil];
                                [alert show];
            }
        }
        default:
            break;
    }
}
```

获取音频设置转发规则

调用此接口获取音频转发规则。调用后接口返回规则，传入的数组参数会返回相应规则的 openId。

接口原型

Unity

```
public abstract ITMG_SERVER_AUDIO_ROUTE_SEND_TYPE
GetCurrentSendAudioRoute(List<string> OpenIDforSend);
public abstract ITMG_SERVER_AUDIO_ROUTE_RECV_TYPE
GetCurrentRecvAudioRoute(List<string> OpenIDforRecve);
```

iOS

```
-(ITMG_SERVER_AUDIO_ROUTE_SEND_TYPE)GetCurrentSendAudioRoute:
(NSMutableArray *) OpenIDForSend;
-(ITMG_SERVER_AUDIO_ROUTE_RECV_TYPE)GetCurrentRecvAudioRoute:
(NSMutableArray *) OpenIDForRecv;
```

返回规则

ITMG_SERVER_AUDIO_ROUTE_SEND_TYPE

接收类型	效果
AUDIO_ROUTE_NOT_SEND_TO_ANYONE	本端音频上行发送到后台，但后台不转发给任何人，相当于将自己静音
AUDIO_ROUTE_SEND_TO_ALL	本端音频上行将转发给所有人
AUDIO_ROUTE_SEND_BLACK_LIST	本端音频上行将不转发给黑名单的人
AUDIO_ROUTE_SEND_WHITE_LIST	本端音频上行将只转发给白名单的人
AUDIO_ROUTE_RECV_INQUIRE_ERROR	获取出错，检查是否进入房间，是否已经初始化 SDK

ITMG_SERVER_AUDIO_ROUTE_RECV_TYPE

接收类型	效果
AUDIO_ROUTE_NOT_RECV_FROM_ANYONE	本端不接受任何音频，相当于关闭房间内扬声器效果
AUDIO_ROUTE_RECV_FROM_ALL	本端接收所有人的音频
AUDIO_ROUTE_RECV_BLACK_LIST	本端不接收黑名单的人的音频声音
AUDIO_ROUTE_RECV_WHITE_LIST	本端只接收白名单的人的音频声音
AUDIO_ROUTE_RECV_INQUIRE_ERROR	获取出错，检查是否进入房间，是否已经初始化 SDK

注意

在 `SetServerAudioRouteSendOperateType` 接口中请勿使用 `AUDIO_ROUTE_RECV_INQUIRE_ERROR` 。

自定义消息通道

最近更新时间：2024-10-15 14:45:41

为方便 GME 开发者调试和接入腾讯云游戏多媒体引擎产品 API，本文为您介绍适用于 GME 用户自定义音频包附带消息功能的使用指引。

使用场景

GME 用户自定义音频包附带消息功能可以让开发者在 GME 音频包中携带自定义消息，作为信令传递广播给同房间的人。

前提条件

- **已开通实时语音服务**：可参见 服务开通指引 。
- **已接入 GME SDK**：包括核心接口和实时语音接口的接入，详情可参见 Native SDK 快速接入 、Unity SDK 快速接入 、Unreal SDK 快速接入 。

使用限制

调用此接口需要在房间类型为**标准**及**高清**（ITMG_ROOM_TYPE_STANDARD 及 ITMG_ROOM_TYPE_HIGHQUALITY）的情况下，发送端需要打开麦克风，接收端需要打开扬声器。

自定义消息功能接入

发送自定义消息

接口原型

iOS

```
-(int) SendCustomData:(NSData *)data repeatCout:(int)reaptCout;
```

Android

```
public abstract int SendCustomData(byte[] data,int repeatCout);
```

Unity

```
public abstract int SendCustomData(byte[] customdata,int repeatCout);
```

参数说明

参数	类型	含义
data	NSData * 、byte[]	需要传递的信息
reaptCout	int	重复次数，填入-1为无限次重复发送

返回值

接口返回值为 QAV_OK 则表示成功。
回调返回1004表示参数错误，建议重新检查参数是否正确。返回1201表示房间不存在，建议检查房间号是否正确。
更多错误码请参见 错误码 文档。

示例代码

执行语句

iOS

```
-(IBAction)SendCustData:(UIButton*)sender {
    int ret = 0;
    NSString *typeString;
    switch (sender.tag) {
        case 1:
            ret = [[[ITMGContext GetInstance] GetRoom] SendCustomData:
[NSData dataWithBytes:_shareRoomID.text.UTF8String
length:_roomIdText.text.length] repeatCout:_shareOpenID.text.intValue];
            typeString = @"sendCustData";
            break;
          case 2:
            ret = [[[ITMGContext GetInstance] GetRoom]
StopSendCustomData];
            typeString = @"recvCustData";
            break;
        default:
            break;
    }
    if(ret != 0) {
        UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"set
fail" message:[NSString stringWithFormat:@"%@:errorcode
```

```
:%d",typeString,ret] delegate:NULL cancelButtonTitle:@"OK"
otherButtonTitles:nil];
        [alert show];
    }
}
```

Android

```
String strData = mEditData.getText().toString();
String repeatCount = mEditRepeatCount.getText().toString();
int nRet =
ITMGContext.GetInstance(getActivity()).GetRoom().SendCustomData(strData.
getBytes(), Integer.parseInt(repeatCount));
```

Unity

```
InputField SendCustom_Count_InputField =
transform.Find("inroomPanel/imPanel/SendCustom_Count_InputField").GetCom
ponent<InputField>();
InputField SendCustom_Data_InputField =
transform.Find("inroomPanel/imPanel/SendCustom_Data_InputField").GetComp
onent<InputField>();

transform.Find("inroomPanel/imPanel/SendCustom_Btn").GetComponent<Button
>().onClick.AddListener(delegate ()
       {
           string data = SendCustom_Data_InputField.text;
           string str_count = SendCustom_Count_InputField.text;
           int count = 0;
           if (int.TryParse(str_count, out count)) {
               Debug.Log(data+ count.ToString());
               byte[] byteData = Encoding.Default.GetBytes(data);
              int ret =
ITMGContext.GetInstance().GetRoom().SendCustomData(byteData, count);
              if(ret != 0 ) {
                 ShowWarnning(string.Format("send customdata failed err:
{0}",ret));
```

```
                }
            }
        });
}
```

回调

iOS

```
-(IBAction)SendCustData:(UIButton*)sender {
    int ret = ret = [[[ITMGContext GetInstance] GetRoom] SendCustomData:
[NSData dataWithBytes:dataInput_textfiled_.text.UTF8String
length:dataInput_textfiled_.text.length] repeatCout:-1];

    if(ret != 0) {
        UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"set
fail" message:[NSString stringWithFormat:@"SendCustomData:errorcode
:%d",ret] delegate:NULL cancelButtonTitle:@"OK" otherButtonTitles:nil];
        [alert show];
    }
}
```

Android

```
if
(ITMGContext.ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVENT_TYPE_CUSTOMDATA_UPDATE
== type) {
    int subtype   =   data.getIntExtra("sub_event",-1);
    if (subtype == 0) {
        String content =   data.getStringExtra("content");
        String sender = data.getStringExtra("senderid");
        Toast.makeText(getActivity(), String.format("recv content =%s,
from:%s", content,
            sender), Toast.LENGTH_SHORT).show();
}
```

Unity

```
void OnEvent(int eventType,int subEventType,string data)
{
    Debug.Log (data);
    switch (eventType) {
     case
(int)ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVENT_TYPE_CUSTOMDATA_UPDATE:
      {
         if(subEventType ==
(int)ITMG_CUSTOMDATA_SUB_EVENT.ITMG_CUSTOMDATA_AV_SUB_EVENT_UPDATE) {
             _customData = JsonUtility.FromJson<CustomDataInfo>(data);
           ShowWarnning(string.Format("recve customdata {0}  from
{1}",_customData.content,_customData.senderid));
         }
      }
     break;
}
```

停止发送自定义消息

调用此接口停止发送自定义消息。

接口原型

iOS

```
-(int) StopSendCustomData;
```

Android

```
public abstract int StopSendCustomData();
```

Unity

```
public abstract int StopSendCustomData();
```

返回值

如果接口返回1003代表已经操作了 StopSendCustomData，SDK 正在进行这个操作，无需再次调用。

如何应对公司防火墙限制

最近更新时间：2024-10-15 14:45:41

如果公司内部有外网访问限制，需要添加防火墙白名单才能访问，相关的规则如下：

客户端 Native SDK（版本号≥2.2）

防火墙端口：

端口类型	白名单项目
TCP 端口	443
UDP 端口	8000

域名白名单：

```
tcloud.tim.qq.com
gmeconf.qcloud.com
yun.tim.qq.com
```

> **注意**
> - 因为腾讯云服务端 IP 地址是动态更新的，并不是固定的一批 IP 地址，所以我们无法提供固定的一组 IP 列表给您。
> - Windows XP 下使用 GME SDK 还需添加以下防火墙白名单。

防火墙端口：

端口类型	白名单项目
TCP 端口	15000

域名白名单：

```
cloud.tim.qq.com
openmsf.3g.qq.com
```

使用 H5 SDK

防火墙端口：

端口类型	白名单项目
TCP 端口	443,8687
UDP 端口	8000；8800；843；443

域名白名单：

```
qcloud.rtc.qq.com
rtc.qcloud.qq.com
```

使用语音消息及转文字服务

防火墙端口：

端口类型	白名单项目
TCP 端口	80，443

域名白名单：

```
gmespeech.qcloud.com
yun.tim.qq.com
gmeconf.qcloud.com
```

> ⚠ **注意：**
> 如果依然无法访问 GME 服务，可以通过 提交工单 联系 GME 开发者。

语言参数参考列表

最近更新时间：2024-02-21 14:33:31

本文主要为您介绍语音转文本服务、文本翻译服务以及文本转语音服务的语言参数参考列表，方便您调试和接入腾讯云游戏多媒体引擎服务。

录音文件极速转文本

录音文件极速转文本只支持中文普通话，在 SDK 中所对应的接口是 SpeechToText。

语言	参数	中文翻译
中文普通话	cmn-Hans-CN	中文，普通话（简体中文，中文）

实时语音转文本

实时语音转文本服务包含 实时语音转文本 与语音消息流式转文本，在 SDK 中所对应的接口是 StartRealTimeASR 以及 StartRecordingWithStreamingRecognition，支持中文普通话、英文、粤语、日语和上海话方言。

语言	参数	中文翻译
中文普通话	cmn-Hans-CN	中文，普通话（简体中文，中文）
English	en-US	英语
粤语，广东话（方言）	yue-Hant-HK	中文，广东话（简体中文，中文）
日本 （日本）	ja-JP	日文（日本）
上海话（方言）	wuu-SH	中文，上海话（简体中文，中文）

海外语音转文本服务、文本翻译、文本转语音

海外语音转文本服务、文本翻译、文本转语音服务目前支持全语种，文本翻译对应的接口是 TranslateText，文本转语音对应的接口是 TextToSpeech。

语言	参数	中文翻译
普通话（中国大陆）	cmn-Hans-CN	中文，普通话（简体中文，中文）
（中国台 ）	cmn-Hant-TW	中文，普通话（繁体，中国台湾）

（中国香港）	yue-Hant-HK	中文，广东话（繁体，中国香港）
普通 （中国香港）	cmn-Hans-HK	中文，普通话（简体，中国香港）
Afrikaans (Suid-Afrika)	af-ZA	南非荷兰语（南非）
አማርኛ (ኢትዮጵያ)	am-ET	阿姆哈拉语（埃塞俄比亚）
Հայ (Հայաստան)	hy-AM	亚美尼亚语（亚美尼亚）
Azərbaycan (Azərbaycan)	az-AZ	阿塞拜疆（阿塞拜疆）
Bahasa Indonesia (Indonesia)	id-ID	印度尼西亚（印度尼西亚）
Bahasa Melayu (Malaysia)	ms-MY	马来语（马来西亚）
বাংলা (বাংলাদেশ)	bn-BD	孟加拉语（孟加拉国）
বাংলা (ভারত)	bn-IN	孟加拉语（印度）
Català (Espanya)	ca-ES	加泰罗尼亚（西班牙）
Čeština (Česká republika)	cs-CZ	捷克（捷克共和国）
Dansk (Danmark)	da-DK	丹麦语（丹麦）
Deutsch (Deutschland)	de-DE	德语（德国）
English (Australia)	en-AU	英语（澳大利亚）
English (Canada)	en-CA	英语（加拿大）
English (Ghana)	en-GH	英语（加纳）
English (Great Britain)	en-GB	英语（英国）
English (India)	en-IN	英文（印度）
English (Ireland)	en-IE	英语（爱尔兰）
English (Kenya)	en-KE	英语（肯尼亚）
English (New Zealand)	en-NZ	英语（新西兰）
English (Nigeria)	en-NG	英语（尼日利亚）
English (Philippines)	en-PH	英语（菲律宾）

English (South Africa)	en-ZA	英语（南非）
English (Tanzania)	en-TZ	英语（坦桑尼亚）
English (United States)	en-US	英语（美国）
Español (Argentina)	es-AR	西班牙语（阿根廷）
Español (Bolivia)	es-BO	西班牙语（玻利维亚）
Español (Chile)	es-CL	西班牙语（智利）
Español (Colombia)	es-CO	西班牙语（哥伦比亚）
Español (Costa Rica)	es-CR	西班牙语（哥斯达黎加）
Español (Ecuador)	es-EC	西班牙语（厄瓜多尔）
Español (El Salvador)	es-SV	西班牙语（萨尔瓦多）
Español (España)	es-ES	西班牙语（西班牙）
Español (Estados Unidos)	es-US	西班牙语（美国）
Español (Guatemala)	es-GT	西班牙语（危地马拉）
Español (Honduras)	es-HN	西班牙语（洪都拉斯）
Español (México)	es-MX	西班牙语（墨西哥）
Español (Nicaragua)	es-NI	西班牙语（尼加拉瓜）
Español (Panamá)	es-PA	西班牙语（巴拿马）
Español (Paraguay)	es-PY	西班牙语（巴拉圭）
Español (Perú)	es-PE	西班牙语（秘鲁）
Español (Puerto Rico)	es-PR	西班牙语（波多黎各）
Español (República Dominicana)	es-DO	西班牙语（多米尼加共和国）
Español (Uruguay)	es-UY	西班牙语（乌拉圭）
Español (Venezuela)	es-VE	西班牙语（委内瑞拉）
Euskara (Espainia)	eu-ES	巴斯克（西班牙）

Filipino (Pilipinas)	fil-PH	菲律宾（菲律宾）
Français (Canada)	fr-CA	法语（加拿大）
Français (France)	fr-FR	法国（法国）
Galego (España)	gl-ES	加利西亚（西班牙）
ქართული (საქართველო)	ka-GE	格鲁吉亚（格鲁吉亚）
ગુજરાતી (ભારત)	gu-IN	古吉拉特语（印度）
Hrvatski (Hrvatska)	hr-HR	克罗地亚（克罗地亚）
IsiZulu (Ningizimu Afrika)	zu-ZA	祖鲁（南非）
Íslenska (Ísland)	is-IS	冰岛（冰岛）
Italiano (Italia)	it-IT	意大利（意大利）
Jawa (Indonesia)	jv-ID	爪哇（印度尼西亚）
( )	kn-IN	卡纳达（印度）
( )	km-KH	高棉（柬埔寨）
( )	lo-LA	老挝（老挝）
Latviešu (latviešu)	lv-LV	拉脱维亚语（拉脱维亚）
Lietuvių (Lietuva)	lt-LT	立陶宛语（立陶宛）
Magyar (Magyarország)	hu-HU	匈牙利语（匈牙利）
മലയാളം (ഇന്ത്യ)	ml-IN	马拉雅拉姆语（印度）
मराठी (भारत)	mr-IN	马拉地语（印度）
Nederlands (Nederland)	nl-NL	荷兰（荷兰）
नेपाली (नेपाल)	ne-NP	尼泊尔语（尼泊尔）
Norsk bokmål (Norge)	nb-NO	挪威语Bokmål（挪威）
Polski (Polska)	pl-PL	波兰语（波兰）
Português (Brasil)	pt-BR	葡萄牙语（巴西）
Português (Portugal)	pt-PT	葡萄牙（葡萄牙）

Română (România)	ro-RO	罗马尼亚（罗马尼亚）
සිංහල (ශ්‍රී ලංකාව)	si-LK	僧伽罗语（斯里兰卡）
Slovenčina (Slovensko)	sk-SK	斯洛伐克（斯洛伐克）
Slovenščina (Slovenija)	sl-SI	斯洛文尼亚（斯洛文尼亚）
Urang (Indonesia)	su-ID	Sundanese（印度尼西亚）
Swahili (Tanzania)	sw-TZ	斯瓦希里语（坦桑尼亚）
Swahili (Kenya)	sw-KE	斯瓦希里语（肯尼亚）
Suomi (Suomi)	fi-FI	芬兰语（芬兰）
Svenska (Sverige)	sv-SE	瑞典语（瑞典）
தமிழ் (இந்தியா)	ta-IN	泰米尔语（印度）
தமிழ் (சிங்கப்பூர்)	ta-SG	泰米尔语（新加坡）
தமிழ் (இலங்கை)	ta-LK	泰米尔语（斯里兰卡）
தமிழ் (மலேசியா)	ta-MY	泰米尔语（马来西亚）
( )	te-IN	泰卢固语（印度）
Tiếng Việt (Việt Nam)	vi-VN	越南语（越南）
Türkçe (Türkiye)	tr-TR	土耳其语（土耳其）
(پاکستان (اردو	ur-PK	乌尔都语（巴基斯坦）
(بھارت (اردو	ur-IN	乌尔都语（印度）
Ελληνικά (Ελλάδα)	el-GR	希腊语（希腊）
Български (България)	bg-BG	保加利亚语（保加利亚）
Русский (Россия)	ru-RU	俄语
Српски (Србија)	sr-RS	塞尔维亚语（塞尔维亚）
Українська (Україна)	uk-UA	乌克兰语（乌克兰）
(ישראל (עברית	he-IL	希伯来语（以色列）

(العربية)إسرائيل	ar-IL	阿拉伯语（以色列）
(العربية)الأردن	ar-JO	阿拉伯语（约旦）
(العربية)الإمارات	ar-AE	阿拉伯语（阿拉伯联合酋长国）
(العربية)البحرين	ar-BH	阿拉伯语（巴林）
(العربية)الجزائر	ar-DZ	阿拉伯语（阿尔及利亚）
(العربية)السعودية	ar-SA	阿拉伯语（沙特阿拉伯）
(العربية)العراق	ar-IQ	阿拉伯语（伊拉克）
(العربية)الكويت	ar-KW	阿拉伯语（科威特）
(العربية)المغرب	ar-MA	阿拉伯语（摩洛哥）
(العربية)تونس	ar-TN	阿拉伯语（突尼斯）
(العربية)عُمان	ar-OM	阿拉伯语（阿曼）
(العربية)فلسطين	ar-PS	阿拉伯语（巴勒斯坦国）
(العربية)قطر	ar-QA	阿拉伯语（卡塔尔）
(العربية)لبنان	ar-LB	阿拉伯语（黎巴嫩）
(العربية)مصر	ar-EG	阿拉伯语（埃及）
(فارسی)ایران	fa-IR	波斯语（伊朗）
हिन्दी (भारत)	hi-IN	印地语（印度）
ไทย (ประเทศไทย)	th-TH	泰国（泰国）
( )	ko-KR	韩国（韩国）

特殊接口

最近更新时间：2025-01-03 11:59:12

为方便开发者调试和接入腾讯云游戏多媒体引擎产品 API，这里向您介绍 GME SDK 高级接口。

通用特殊接口

openid 支持字符串

> **注意：**
> 此接口在 GME 2.9.3 及以上版本 SDK 生效。

目前 GME SDK 只支持数字以字符串形式传入 SDK，如果需要 openid 支持以字符串形式传入，需要在调用 Init 接口之前调用一次以下接口：

```
SetAdvanceParams("StringOpenID", "1");
```

修复打印日志大小

> **说明：**
> 此接口在 GME 2.8.4 以上版本 SDK 上生效，为了不影响开发者调试，请勿在测试阶段修改日志大小。

在 GME Init 初始化接口前调用此接口，用于修改默认日志文件大小。目前日志文件单个为50m，最多存在3个日志文件。

函数原型

```
SetAdvanceParams(const char* key, const char* object)
```

参数	类型	含义
key	const char*	MAX_LOG_FILE_SIZE_MB 和 MAX_LOG_FILE_COUNT，分别代表单个日志的大小和日志的数量。
object	const char*	当key 为 MAX_LOG_FILE_SIZE_MB 时，object 为 Log 文件大小默认值，取值范围（单位M）：5至50；当key 为 MAX_LOG_FILE_COUNT 时，object 为 Log 文件个数默认值，取值范围：1至3。

> **参数范围：**

> 当输入的 object 超过取值范围上限时，则设置为上限值；当输入的 object 小于取值范围下限时，则设置为下限值。

示例代码

```
SetAdvanceParams("MAX_LOG_FILE_SIZE_MB", "5");
SetAdvanceParams("MAX_LOG_FILE_COUNT", "1");
```

音频相关特殊接口

iOS 音频设置接口

此部分接口使用 SetAdvanceParams 接口进行调用，在进房前调用。 调用后影响音频播放效果，需要根据业务需求谨慎调用。

接口原型

```
[[ITMGContext GetInstance] SetAdvanceParams:keyString value:_value]
```

参数	含义
keyString	不同的 Key 代表不同的功能
value	• 0：代表关闭 • 1：代表开启

Key

不同的 Key 代表不同的功能，参数 Key 可填写的字段如下：

- **OptionMixWithOthers**

  混音选项。开启后可以把后台播放的音乐与前台通话语音同时播放。
- **OptionDuckOthers**

  压低背景音。在 OptionMixWithOthers 开启的情况下，如果开启此功能，则开启扬声器播放语音时，将会压低其他后台背景声音。
- **ReleaseAudioFoucus**

  释放音频焦点。
  - 如果开启，退房之后将会释放音频焦点，系统恢复后台其他音频相关应用。例如 QQ 音乐。
  - 如果关闭，退房之后将不会恢复其他音频相关应用。

检查 iPhone 静音键是否开启

> **说明：**
> 此接口在 GME 2.8.4 以上版本 SDK 上生效。

调用接口检查 iPhone 手机静音功能是否开启。

接口原型

```
CheckDeviceMuteState();
```

返回值

返回值为 0 代表关闭物理静音键，返回值为 1 代表打开物理静音键。

设置 Android 蓝牙设备使用媒体音量

> **说明：**
> 此接口在 GME 2.8.4 以上版本 SDK 上生效。

调用此接口后蓝牙耳机使用媒体音量，需要在进房前调用。如果您的用户经常反馈如下问题：

- 蓝牙耳机在开关麦的时候漏声音。
- Android 设备在已连接蓝牙耳机的情况下，切换连接状态后出现声音从扬声器播放。

可以在业务层的设置界面设计一个开关，当用户反馈类似问题，建议客户打开开关，业务层在进房前设置蓝牙耳机使用媒体音量。

调用情况	音量类型影响	采集音频方式
调用前	通话音量	从耳机对用户说话声音进行采集。
调用后	媒体音量	从手机麦克风对用户说话声音进行采集，如果用户离手机麦克风比较远，可能造成采集到的音量很小，影响通话。

```
SetAdvanceParams(“BluetoothUseMedia”, “1”);
```

设置最大混音路数

使用 SetRecvMixStreamCount 接口可以设置最高混音路数，在进房前调用。 各平台均有此接口，下面以 PC 端为例。

```
virtual int SetRecvMixStreamCount(int nCount) = 0;
```

参数说明

参数	含义
nCount	混音路数，最大为20

开关 3A 效果

通过此接口，可对 3A 效果（AEC、AGC、ANS）进行开关，默认为 3A 效果都开启。

> **警告：**
> 非特殊情况，无需调用此接口开关 3A 效果，以免影响音质。

示例代码

C++

```
ITMGContextGetInstance()->SetAdvanceParams("AECFlag", "1");
ITMGContextGetInstance()->SetAdvanceParams("AECFlag", "0");
ITMGContextGetInstance()->SetAdvanceParams("ANSFlag", "1");
ITMGContextGetInstance()->SetAdvanceParams("ANSFlag", "0");
ITMGContextGetInstance()->SetAdvanceParams("AGCFlag", "1");
ITMGContextGetInstance()->SetAdvanceParams("AGCFlag", "0");
```

参数	类型	含义
keyString	String	AECFlag：代表 AEC 功能开关。 ANSFlag：代表 ANS 功能开关。 AGCFlag：代表 AGC 功能开关。
value	String	"1"：代表开。 "0"：代表关。

设置房间音频类型

进房前使用 SetForceUseMediaVol ，可以让流畅音质房间或者标准音质房间使用媒体音量。

> **警告：**
> 非特殊情况，无需调用此接口，以免影响音质。

接口原型

```
[[ITMGContext GetInstance] SetAdvanceParams:SetForceUseMediaVol value:1]
```

value

不同的 value 代表不同的功能，具体如下：

- 0：房间类型1与2的语音房间都还原为开麦克风后音量为通话音量。
- 1：房间类型1的语音房间开麦克风后为媒体音量（原为通话音量）。
- 2：房间类型2的语音房间开麦克风后为媒体音量（原为通话音量）。

获取房间内成员说话音量

调用 TrackingVolume 接口之后，监听 `TIMGContext.ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVENT_TYPE_USER_VOLUMES` 事件，里面的键值对为 uin-volume，通过此接口可以根据房间内某 uin 说话的音量强度画出相应的能量柱状图。
如果不再获取，请调用 StopTrackingVolume 接口。

```
//TMGAudioCtrl
public int TrackingVolume(float fTrackingTimeS)
public int StopTrackingVolume();
```

参数	类型	含义
fTrackingTimeS	Float	监听的秒数，建议设置为 0.5f

音文同传相关接口

文本翻译接口

调用此接口，可以将一段文本进行翻译，例如将一段中文翻译成英文。调用接口后通过回调返回翻译结果。此处以 C++ 接口为例。

函数原型

```
virtual int TranslateText(const char* text, const char* sourceLanguage,
const char* translateLanguage) = 0;
```

参数	类型	含义
text	const char*	待翻译的文本，不可为空，最大长度5000字符

sourceLanguage	const char*	指定待翻译文本的语言，可为空，后台自动检测语音
translateLanguage	const char*	指定文本翻译后的语言，不可为空，以英文逗号作为间隔，例如"cmn-Hans-CN,en-GB"、"cmn-Hans-CN"

语言参数请参见 语言参数列表 。

接口回调

参数	类型	含义
code	Int	错误码，0代表成功，其他返回请参考 错误码 进行解决
targetText	jason(Unity 中返回 String 格式)	翻译后的目标文本，例如 {"target_text":[{"target_language_code":"cmn-Hans-CN","target_text":"我是中国人"},{"target_language_code":"de-DE","target_text":"Ich bin Chinese"}]}

Unity 工程示例代码

1. 添加监听。

```
ITMGContext.GetInstance().GetPttCtrl().OnTranslateTextComplete+=
OnTranslateTextComplete;
```

2. 调用接口。

```
private void OnTranslateTextBtn()
{
  mTargetText.text = "";
 int ret =
ITMGContext.GetInstance().GetPttCtrl().TranslateText(mSourceText.
text, mSourceLanguageText.text, mTargetLanguageText.text);
 if (0 != ret)
 {
      mTargetText.text = "Invalid Atgument";
 }
}
```

3. 处理回调。

```
void OnTranslateTextComplete(int code, string targetText)
{
 if (0 == code)
 {
     mTargetText.text = targetText;
 }
 else
 {
     mTargetText.text = String.Format(
         "Translate Text Error, Error Code:{0}", code);
 }
}
```

4. 取消监听。

```
ITMGContext.GetInstance().GetPttCtrl().OnTranslateTextComplete-=
OnTranslateTextComplete;
```

文本转语音接口

调用此接口，可以将一段文本转成对应的一段语音，调用接口后，通过回调返回语音 fileid，通过下载语音接口进行下载此语音。此处以 C++ 接口为例。

> ⚠ **注意：**
> 此接口仅对白名单用户开放，且只在SDK 291版本上才能使用。若您有需求，请 提交工单 联系工作人员申请服务。

函数原型

```
virtual int TextToSpeech(const char* text, const char* voiceName,const
char* languageCode, float speakingRate) = 0;
```

参数	类型	含义
text	const char*	原始文本，不可为空，最大长度5000字符
voiceName	const char*	声音类型，提供英语及普通话的示例，如需其他语言请 提交工单 咨询

languageCode	const char*	指定目标语言。不可为空
speakingRate	Float	音频语速，取值范围 [0.6-1.5]，1代表正常速度

voiceName 说明如下：

声音类型	性别	语言
cmn-CN-Standard-A	女声	普通话
cmn-CN-Standard-B	男声	普通话
en-US-Neural2-A	女声	英语
en-US-Neural2-B	男声	英语

语言参数请参见 语言参数列表 。

接口回调

参数	类型	含义
code	Int	错误码，0代表成功，其他返回请参考 错误码 进行解决
isCos	Bool	文件是否上传到 COS
fileID	String	文件 ID，提供下载接口入参，可以通过 DownloadRecordedFile 接口下载音频

Unity 工程示例代码

1. 添加监听。

```
ITMGContext.GetInstance().GetPttCtrl().OnTextToSpeechComplete += new
QAVTextToSpeechCallback(TextToSpeechComplate);
```

2. 调用接口。

```
void OnTextToSpeech()
{
 float fSpeakingRate;
```

```
 if (!float.TryParse(mSpeakingRate.text, out fSpeakingRate))
 {
     mReturnData.text = "SpeakingRate invalid";
      return;
 }

 int iRet = ITMGContext.GetInstance().GetPttCtrl().TextToSpeech(
     mSouceText.text, mVoiceName.text,
     mLanguageCode.text, fSpeakingRate);

 Debug.Log(string.Format("TextToSpeech Code:{0}", iRet));

 if (0 != iRet)
 {
     mReturnData.text = string.Format("TextToSpeech Error, errorCode:
{0}", iRet);
     return;
 }
}
```

3. 处理回调。

```
void TextToSpeechComplate(int code, bool isCos, string fileID)
{
 Debug.Log(string.Format("TextToSpeechComplate Code:{0}", code));

 if (0 != code)
 {
     mReturnData.text = string.Format("TextToSpeech Error, errorCode:
{0}", code);
     return;
 }

 mReturnData.text = string.Format("code:{0}\r\nisCos:{1}\r\nfileID:
{2}", code, isCos, fileID);
 mDownloadUrl.text = fileID;
}
```

4. 取消监听。

```
ITMGContext.GetInstance().GetPttCtrl().OnTextToSpeechComplete -= new
QAVTextToSpeechCallback(TextToSpeechComplate);
```

临时热词接口

通过该接口设置临时热词表，在调用转文本接口前调用此接口，对接下来的转文本调用生效。如果无需临时热词功能，请在参数 object 中填入“”，用于清空热词表。
临时热词表只对中文生效，如果此次调用转文本功能使用了临时热词表，则在控制台上传的热词表此次调用不生效。

接口原型

```
int SetAdvanceParams(const char* key, const char* object);
```

接口参数

参数	类型	含义
key	const char*	请填入 "SetPTTHotWorldList"
object	const char*	• 传入要设置的热词列表。传"" 清空上次设置的热词。 • 单个热词规则：“热词|权重”，不超过30个字符(最多10个汉字)，权重1-10。1代表权重最低，10代表权重最高。 • 热词列表：多个热词之间用英文逗号分割，最多128个热词，参数示例："腾讯云|10"、"语音识别|5,ASR|10"。

示例代码

```
ITMGContext.GetInstance().SetAdvanceParams("SetPTTHotWorldList", "会敢去
采|10,塌下来又怎样|2,何来罪恶|2,自在喝奶咖|7,几成痴|10,");
```