游戏多媒体引擎

客户端 API

腾讯云

版权所有：腾讯云计算（北京）有限责任公司

【版权声明】

©2013-2025 腾讯云版权所有

本文档（含所有文字、数据、图片等内容）完整的著作权归腾讯云计算（北京）有限责任公司单独所有，未经腾讯云事先明确书面许可，任何主体不得以任何形式复制、修改、使用、抄袭、传播本文档全部或部分内容。前述行为构成对腾讯云著作权的侵犯，腾讯云将依法采取措施追究法律责任。

【商标声明】

及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。未经腾讯云及有关权利人书面许可，任何主体不得以任何方式对前述商标进行使用、复制、修改、传播、抄录等行为，否则将构成对腾讯云及有关权利人商标权的侵犯，腾讯云将依法采取措施追究法律责任。

【服务声明】

本文档意在向您介绍腾讯云全部或部分产品、服务的当时的相关概况，部分产品、服务的内容可能不时有所调整。
您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

文档目录

客户端 API
Unity SDK
集成 SDK
实时语音
语音消息及转文本
工程导出
Unreal Engine SDK
集成 SDK
实时语音
语音消息及转文本
Cocos2D SDK
工程配置
快速入门
实时语音
语音消息及转文本
Windows SDK
工程配置
实时语音
语音消息及转文本
iOS SDK
集成 SDK
实时语音
语音消息及转文本
工程导出
iOS 工程升级指引
Android SDK
集成 SDK
实时语音
语音消息及转文本
工程导出
macOS SDK
工程配置
实时语音
语音消息及转文本
H5 SDK
工程配置
接口文档
Electron SDK
集成 SDK
实时语音
语音消息及转文本
Flutter SDK
集成 SDK
实时语音
语音消息及转文本
SDK 版本升级指引

版权所有：腾讯云计算（北京）有限责任公司

编译工具链文档
错误码

客户端 API
Unity SDK
集成 SDK

最近更新时间：2023-03-20 15:36:28

为方便 Unity 开发者调试和接入腾讯云游戏多媒体引擎产品 API，这里向您介绍适用于 Unity 开发的工程配置。

下载 SDK

1. 请先下载相关 Demo 及 SDK。详细下载链接请查看 SDK 下载指引 。
2. 在界面中找到 Unity 版本的 SDK 资源。
3. 单击**下载**。下载完的 SDK 资源解压后有以下几个部分。文件说明如下表：

文件名	说明	作用
Plugins	SDK 库文件	存放导出各个平台的库文件
GMESDK	SDK 代码文件	提供 API 接口

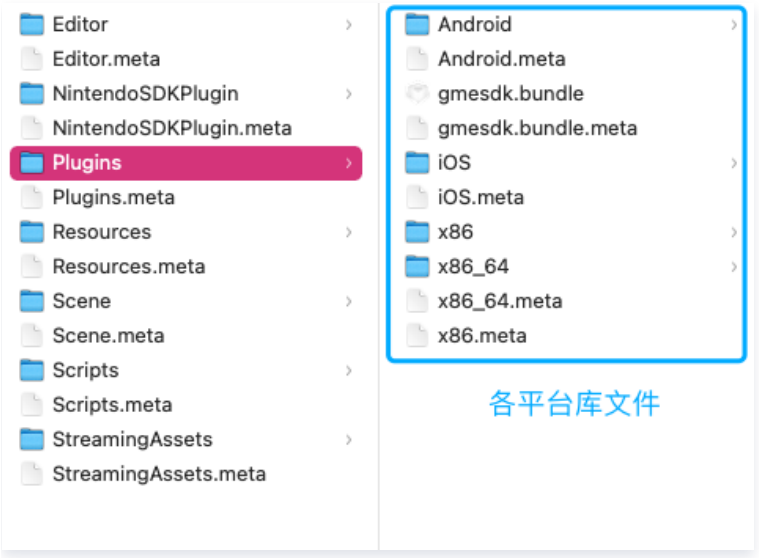
4. 如果使用到高清音质请参见 Unity 配置高清音质文档 进行解决。

> **平台支持**
> Unity SDK 已同时集成 Windows、Mac、Android、iOS、PlayStation、Xbox、Switch、WebGL 平台架构。

工程配置步骤

步骤1：导入 Plugins 文件

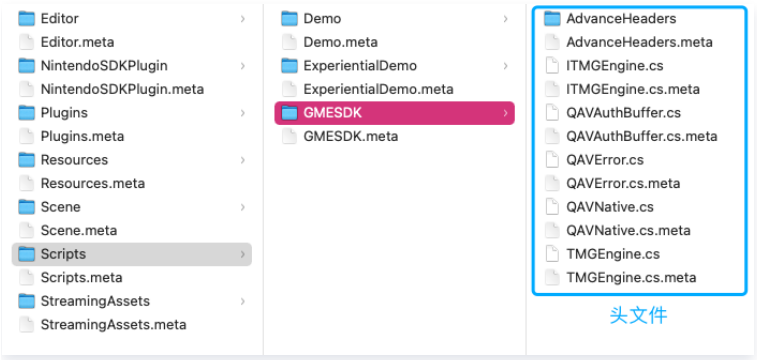
将开发工具包中 Plugins 文件夹中的文件复制在 **Unity 工程** > **Assets** > **Plugins** 文件夹中，如图所示。



> **说明**
> 如果不需要导出 win32 架构的可执行文件，请删除 Plugins 文件夹下的 x86 文件夹。

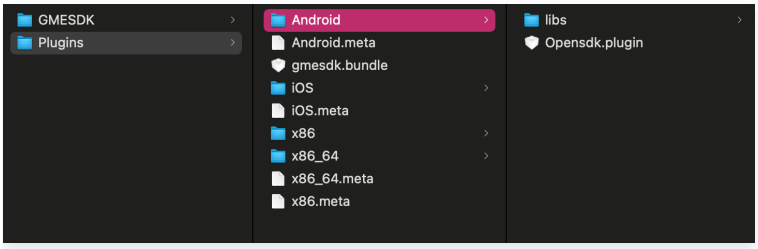
步骤2：导入代码文件

将开发工具包中 Scripts 文件夹中的文件复制在 Unity 工程中存放代码的文件夹中，如图所示：



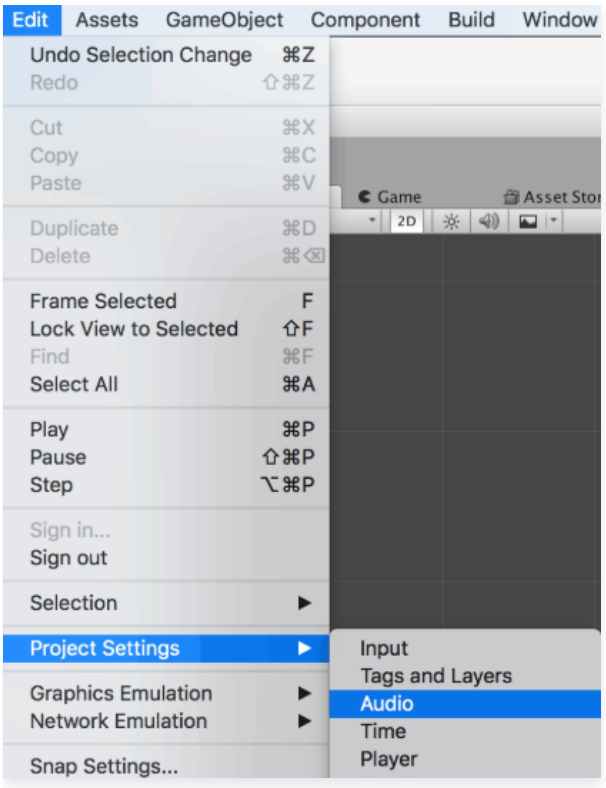
Unity 2021配置

如果使用的 Unity Editor 是 Unity 2021 及以上版本，需要将 Plugins > Android > Opensdk.plugin 下的 lib 文件夹，剪切后放在工程中 Plugins 文件中的 Android 目录下，与 Opensdk.plugin 同一级。



音频设置

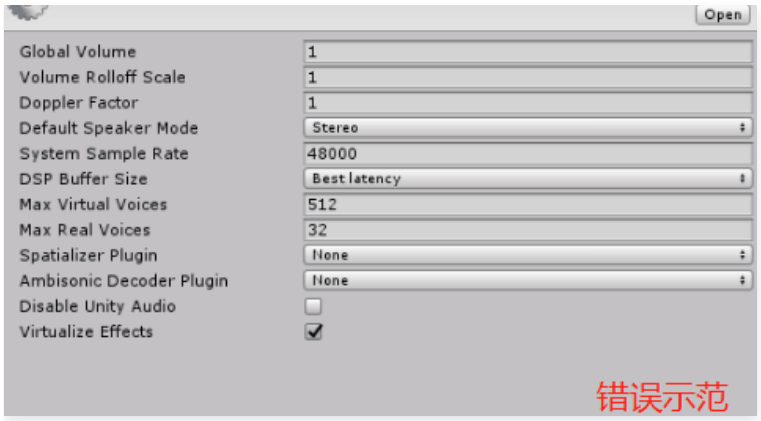
在 Unity 编辑器中，**Edit>Project Setting>Audio** 使用系统默认即可。如果进行修改，Unity 播放音效会因为在 iOS 上设置硬件缓存区受影响，表现为音效被打断。如图所示。



⊘ Unity Audio Setting

> 禁止设置 Project Setting 中的 Audio 模块。

若按照下图进行配置，则 Unity 播放音效会因为在 iOS 上设置硬件缓存区受影响，表现为音效被打断。如图所示。



MacOS 平台使用操作

若在 MacOS 10.15.x 版本使用 Unity 集成 GME SDK，执行运行操作时报错显示文件已损坏，原因为 `com.apple.quarantine` 属性导致。

最直接的解决方案是删除 `com.apple.quarantine` 属性，具体操作步骤如下。

1. 通过终端执行 cd 命令快速定位到工程中的文件夹下：`Unity_OpenSDK_Audio/Assets/Plugins/` 。
2. 执行以下命令。

```
$ xattr -d com.apple.quarantine gmesdk.bundle
```

> **说明**
> 此操作有风险，建议使用低版本的 MacOS 进行接入。

实时语音

最近更新时间：2025-02-17 15:28:43

为方便 Unity 开发者调试和接入腾讯云游戏多媒体引擎客户端 API，本文为您介绍适用于 Unity 实时语音功能的开发接入技术文档。

使用 GME 重要事项

GME 提供实时语音服务、语音消息服务及转文本服务，使用 GME 服务都依赖 Init 和 Poll 等核心接口。

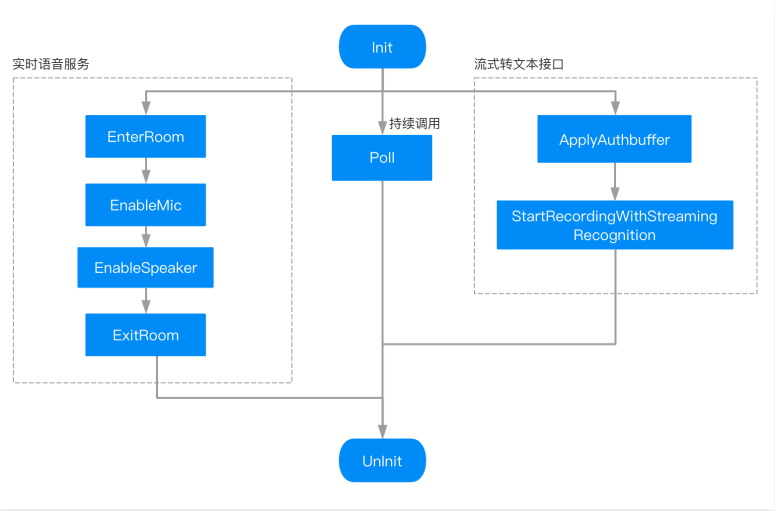
重点提示

- 已完成 GME 应用创建，并获取 SDK AppID 和 Key。请参见 服务开通指引 。
- 已开通 **GME 实时语音服务、语音消息服务以及转文本服务**。请参见 服务开通指引 。
- GME 使用前请对工程进行配置，否则 SDK 不生效。
- GME 的接口调用成功后返回值为 QAVError.OK，数值为 0。
- GME 的接口调用要在同一个线程下。
- GME 需要周期性的调用 Poll 接口触发事件回调。
- 错误码详情可参见 错误码 。
- GME 在 Unity-WebGL 平台只支持简易的实时语音通话功能，具体参见 H5 工程配置 。

接入 SDK

重要步骤

接入 SDK 重要流程如下：



1. 初始化 GME
2. 周期性调用 Poll 触发回调
3. 进入实时语音房间
4. 打开麦克风
5. 打开扬声器
6. 退出语音房间
7. 反初始化 GME

C# 类

类	含义

ITMGContext	核心接口
ITMGRoom	房间相关接口
ITMGRoomManager	房间管理接口
ITMGAudioCtrl	音频相关接口
ITMGAudioEffectCtrl	音效及伴奏相关接口

核心接口

接口	接口含义
Init	初始化 GME
Poll	触发事件回调
Pause	系统暂停
Resume	系统恢复
Uninit	反初始化 GME

引用头文件

```
using GME;
```

获取实例

请使用 ITMGContext 的方法获取 Context 实例，不要直接使用 QAVContext.GetInstance() 去获取实例。

初始化 SDK

未初始化前，SDK 处于未初始化阶段，**需要通过接口 Init 初始化 SDK**，才可以使用实时语音服务、语音消息服务及转文本服务。调用 Init 接口的线程必须于其他接口在同一线程,建议都在主线程调用接口。

接口原型

```
//class ITMGContext
public abstract int Init(string sdkAppID, string openID);
```

参数	类型	含义
sdkAppId	string	来自 腾讯云控制台 的 GME 服务提供的 AppID，获取请参见 语音服务开通指引 。
openID	string	openID 只支持 Int64 类型（转为 string 传入），规则由 App 开发者自行制定，App 内不重复即可。如需使用字符串作为 Openid 传入，可 提交工单 联系开发者。

返回值

返回值	处理
QAVError.OK= 0	初始化 SDK 成功
AV_ERR_SDK_NOT_FULL_UPDATE=7015	检查 SDK 文件是否完整，建议删除后重新导入 SDK

> ⚠ **注意：**
> - 7015错误码是通过 MD5 进行判断，在接入过程中若出现此错误，请根据提示检查 SDK 文件是否完整、SDK 文件版本是否一致。
> - 出现返回值 AV_ERR_SDK_NOT_FULL_UPDATE 时，此返回值**只有提示作用**，并不会造成初始化失败。
> - 由于第三方加固、Unity 打包机制等因素会影响库文件 md5，造成误判，所以**正式发布请在逻辑中忽略此错误**，并尽量不在 UI 中提示。

示例代码

```
int ret = ITMGContext.GetInstance().Init(sdkAppId, openID);
//通过返回值判断是否初始化成功
if (ret != QAVError.OK)
    {
        Debug.Log("SDK初始化失败:"+ret);
        return;
    }
```

触发事件回调

通过在 update 里面周期的调用 Poll 可以触发事件回调。Poll 是 GME 的消息泵，GME 需要周期性的调用 Poll 接口触发事件回调。如果没有调用 Poll ，将会导致整个 SDK 服务运行异常。详情请参见 Sample Project 中的 EnginePollHelper 文件。

> ⚠ **注意：**
> 务必周期性调用 Poll 接口且在主线程调用，以免接口回调异常。

接口原型

```
ITMGContext public abstract int Poll();
```

示例代码

```
public void Update()
    {
        ITMGContext.GetInstance().Poll();
    }
```

系统暂停

当系统发生 Pause 事件时，需要同时通知引擎进行 Pause。例如在应用退后台时候（OnApplicationPause, isPause=True），如果不需要后台播放房间内声音，请调用 Pause 接口暂停整个 GME 服务。

接口原型

```
ITMGContext public abstract int Pause()
```

系统恢复

当系统发生 Resume 事件时，需要同时通知引擎进行 Resume。Resume 接口只恢复实时语音。

接口原型

```
ITMGContext public abstract int Resume()
```

反初始化 SDK

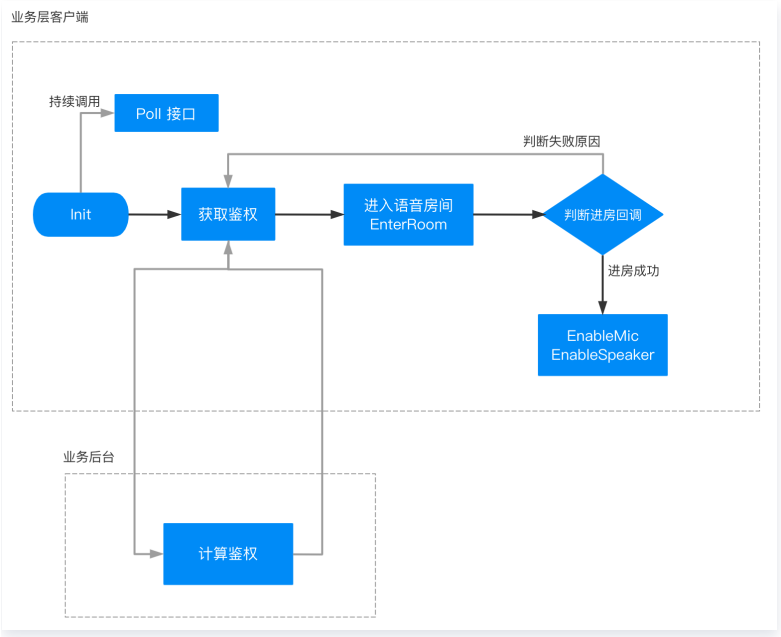
反初始化 SDK，进入未初始化状态。**如果游戏业务侧账号与 openid 是绑定的，那切换游戏账号需要反初始化 GME，再用新的 openid 初始化。**

接口原型

```
ITMGContext public abstract int Uninit()
```

实时语音房间相关接口

初始化之后，SDK 调用进房后进去了房间，才可以进行实时语音通话。
使用问题可参见 实时语音相关问题 。



接口	接口含义
GenAuthBuffer	本地鉴权计算
EnterRoom	加入房间
ExitRoom	退出房间
IsRoomEntered	判断是否已经进入房间
SwitchRoom	快速切换房间

本地鉴权计算

生成 AuthBuffer，用于相关功能的加密和鉴权，如正式发布请使用后台部署密钥，后台部署请参见 鉴权密钥 。

接口原型

```
QAVAuthBuffer GenAuthBuffer(int appId, string roomId, string openId, string key)
```

参数	类型	含义
appId	int	来自腾讯云控制台的 AppID 号码。
roomId	string	房间号，最大支持127字符。
openId	string	用户标识。与 Init 时候的 openID 相同。
key	string	来自腾讯云 控制台 的权限密钥。

示例代码

```
public static byte[] GetAuthBuffer(string AppID, string RoomID,string OpenId, string AuthKey){
        return QAVAuthBuffer.GenAuthBuffer(int.Parse(AppID), RoomID, OpenId, AuthKey);
}
```

WebGL 端适配

- WebGL 平台下，调用本地鉴权函数后，鉴权值是保存在js代码中，并没有将鉴权的authBuffer返回给c#层，用户在调用 GetAuthBuffer 接口本地鉴权后，进入房间时调用的接口参数中，鉴权参数填空或任意值。
- 如果使用后台计算鉴权的方案，则不需要调用 GetAuthBuffer 接口。

加入房间

用生成的鉴权信息进房，加入房间默认不打开麦克风及扬声器。

> ⚠ **注意：**
> - 加入房间事件回调结果 result 为 0 代表进房成功，进房接口 EnterRoom 返回值为 0 不代表进房成功。
> - 房间的音频类型由第一个进房的人确定，此后房间里有成员修改房间类型，将对此房间所有成员生效。例如第一个进入房间的人使用的房间音频类型是流畅音质，第二个进房的是即使进房时候调用接口的音频类型参数是高清音质，进入房间之后也会变成流畅音质。需要有成员调用 ChangeRoomType 才会修改房间的音频类型。

接口原型

```
ITMGContext EnterRoom(string roomId, int roomType, byte[] authBuffer)
```

参数	类型	含义
roomId	String	房间号，最大支持127字符
roomType	ITMGRoomType	房间类型，游戏建议使用 ITMG_ROOM_TYPE_FLUENCY。房间音频类型请参见 音质选择 。
authBuffer	Byte[]	鉴权码

示例代码

```
ITMGContext.GetInstance().EnterRoom(strRoomId, ITMGRoomType.ITMG_ROOM_TYPE_FLUENCY,
byteAuthbuffer);
```

加入房间事件回调

加入房间完成后会通过回调返回进房结果，监听进房结果事件后进行处理。如果回调为成功，即此时进房成功，开始进行计费。

> **计费问题参考：**
> - 购买指南。
> - 计费相关问题。
> - 使用实时语音后，如果客户端掉线了，是否还会继续计费？

接口原型

```
public delegate void QAVEnterRoomComplete(int result, string error_info);
public abstract event QAVEnterRoomComplete OnEnterRoomCompleteEvent;
```

示例代码

```
//对事件进行监听：
ITMGContext.GetInstance().OnEnterRoomCompleteEvent += new
QAVEnterRoomComplete(OnEnterRoomComplete);

//监听处理：
void OnEnterRoomComplete(int err, string errInfo)
    {
    if (err != 0) {
              ShowLoginPanel("错误码：" + err + " 错误信息：" + errInfo);
            return;
    }else{
        //进房成功
    }
}
```

Data 详情

消息	Data	例子
ITMG_MAIN_EVENT_TYPE_ENTER_ROOM	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_ROOM_DISCONNECT	result; error_info	{"error_info":"waiting timeout, please check your network","result":0}

如果断网，将会有断网的回调提示 `ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVENT_TYPE_ROOM_DISCONNECT`，此时 SDK 会自动进行重连，回调是 `ITMG_MAIN_EVENT_TYPE_RECONNECT_START`，当重连成功时，会有 `ITMG_MAIN_EVENT_TYPE_RECONNECT_SUCCESS` 回调。

错误码

错误码值	原因及建议方案
7006	鉴权失败原因： • AppID 不存在或者错误 • authbuff 鉴权错误 • 鉴权过期 • OpenId 不符合规范
7007	已经在其它房间
1001	已经在进房过程中，然后又重复了此操作。建议在进房回调返回之前不要再调用进房接口

1003	已经进房了在房间中，又调用一次进房接口
1101	确保已经初始化 SDK，确保 OpenId 是否符合规则，或者确保在同一线程调用接口，以及确保 Poll 接口正常调用

退出房间

通过调用此接口可以退出所在房间。这是一个异步接口，返回值为 AV_OK 的时候代表异步投递成功。如果应用中有退房后立即进房的场景，在接口调用流程上，开发者无需要等待 ExitRoom 的回调 RoomExitComplete 通知，只需直接调用接口。

接口原型

```
ITMGContext ExitRoom()
```

示例代码

```
ITMGContext.GetInstance().ExitRoom();
```

退出房间事件回调

退出房间完成回调，通过委托传递消息。

接口原型

```
public delegate void QAVExitRoomComplete();
public abstract event QAVExitRoomComplete OnExitRoomCompleteEvent;
```

示例代码

```
对事件进行监听：
ITMGContext.GetInstance().OnExitRoomCompleteEvent += new
QAVExitRoomComplete(OnExitRoomComplete);
监听处理：
void OnExitRoomComplete(){
    //退出房间后的处理
}
```

判断是否已经进入房间

通过调用此接口可以判断是否已经进入房间，返回值为 bool 类型。请勿在进房过程中调用。

接口原型

```
ITMGContext abstract bool IsRoomEntered()
```

示例代码

```
ITMGContext.GetInstance().IsRoomEntered();
```

快速切换房间

调用此接口快速切换实时语音房间。此接口在进房后调用。切换房间后，不重置设备，即如果在此房间已经是打开麦克风状态，在切换房间后也会是打开麦克风状态。

快速切换房间的回调是 ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVENT_TYPE_SWITCH_ROOM，字段是 error_info 以及 result。

接口原型

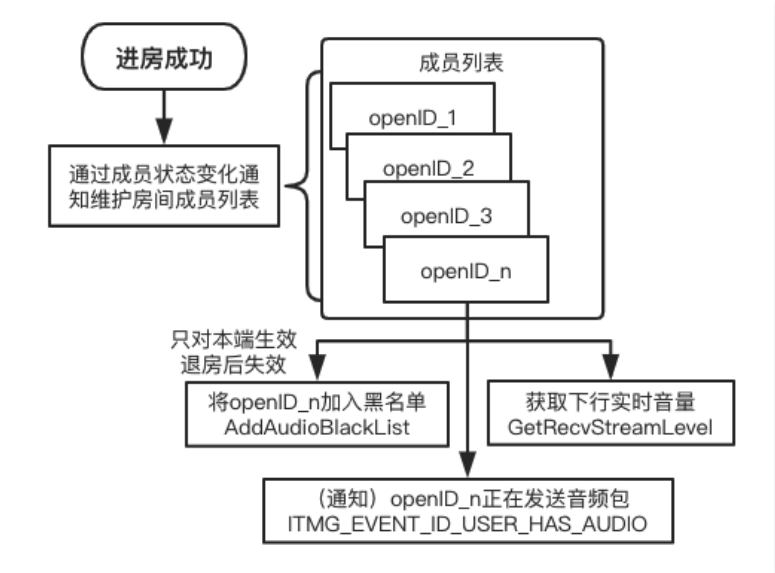
```
public abstract int SwitchRoom(string targetRoomID, byte[] authBuffer);
```

类型说明

参数	类型	含义
targetRoomID	String	将要进入的房间号
authBuffer	Byte[]	用将要进入的房间号生成的新鉴权

房间内状态维护

此部分接口用于业务层显示说话成员、进退房成员，以及将房间内某成员禁言等功能。



接口/通知	含义
ITMG_MAIN_EVNET_TYPE_USER_UPDATE	成员状态变化通知
AddAudioBlackList	房间中禁言某成员
RemoveAudioBlackList	移除禁言

成员进房、开关麦克风、说话状态通知事件

- 该事件适用于获取房间中说话的人并在 UI 中展示，以及有人进入、退出语音房间的一个通知。
- 该事件在状态变化才通知，状态不变化的情况下不通知。如需实时获取成员状态，请在业务层收到通知时缓存，事件消息为 ITMG_MAIN_EVNET_TYPE_USER_UPDATE，包含 event_id、count 及 openIdList，在 OnEvent 通知中对事件消息进行判断。
- 音频事件 EVENT_ID_ENDPOINT_NO_AUDIO 的通知有一个阈值，超过这个阈值才会发送通知。即本端两秒没采集到声音后， 房间其他成员才收到本端停止说话的通知。
- 音频事件只会返回成员说话状态，没有返回具体的音量。如需房间内成员具体音量可使用接口 GetRecvStreamLevel 进行获取。

event_id	含义	应用侧维护内容
EVENT_ID_ENDPOINT_ENTER	有成员进入房间，返回此时进房的 openid	应用侧维护成员列表
EVENT_ID_ENDPOINT_EXIT	有成员退出房间，返回此时退房的 openid	应用侧维护成员列表
EVENT_ID_ENDPOINT_HAS_AUDIO	有成员发送音频包，返回此时房间内说话的 openid，通过此事件可以判断用户是否说话，并展示声纹效果	应用侧维护通话成员列表
EVENT_ID_ENDPOINT_NO_AUDIO	有成员停止发送音频包，返回此时房间内停止说话的 openid	应用侧维护通话成员列表
EVENT_ID_ENDPOINT_MIC_OPENED	有成员打开麦克风设备（2.9.10及以后版本支持）	应用侧维护通话成员列表
EVENT_ID_ENDPOINT_MIC_CLOSED	有成员关闭麦克风设备（2.9.10及以后版本支持）	应用侧维护通话成员列表

示例代码

```
public delegate void QAVEndpointsUpdateInfo(int eventID, int count,
[MarshalAs(UnmanagedType.LPArray, SizeParamIndex = 1)]string[] openIdList);
public abstract event QAVEndpointsUpdateInfo OnEndpointsUpdateInfoEvent;

//对事件进行监听：
ITMGContext.GetInstance().OnEndpointsUpdateInfoEvent += new
QAVEndpointsUpdateInfo(OnEndpointsUpdateInfo);
//监听处理：
void OnEndpointsUpdateInfo(int eventID, int count, string[] openIdList)
{
                //进行处理
            switch (eventID)
             {
             case EVENT_ID_ENDPOINT_ENTER:
                  //有成员进入房间
                  break;
             case EVENT_ID_ENDPOINT_EXIT:
                  //有成员退出房间
                break;
            case EVENT_ID_ENDPOINT_HAS_AUDIO:
                //有成员发送音频包
                break;
            case EVENT_ID_ENDPOINT_NO_AUDIO:
                //有成员停止发送音频包
                break;
            case EVENT_ID_ENDPOINT_MIC_OPENED:
                //有成员开启麦克风设备
                break;
            case EVENT_ID_ENDPOINT_MIC_CLOSED:
                //有成员关闭麦克风设备
                break;
            default:
                break;
             }
        break;
```

```
}
```

房间中禁言某成员

将某个 ID 加入音频数据黑名单，即不接受某人的语音， 只对本端生效，不会影响其他端。返回值为 0 表示调用成功。例如： A、B、C 都在同一个房间开麦说话：

- 如果 A 设置了 C 的黑名单， 则 A 只能听见 B 的声音。
- B 因为没有设置黑名单， 仍旧可以听见 A 和 C 的声音。
- C 同样因为没有设置黑名单， 可以听见 A 和 B 的声音。

此接口适用于在语音房间中将某用户禁言的场景。

接口原型

```
ITMGContext ITMGAudioCtrl AddAudioBlackList(String openId)
```

参数	类型	含义
openId	string	需添加黑名单的用户 openid

示例代码

```
ITMGContext.GetInstance().GetAudioCtrl ().AddAudioBlackList (openId);
```

移除禁言

将某个 ID 移除音频数据黑名单。返回值为0表示调用成功。

接口原型

```
ITMGContext ITMGAudioCtrl RemoveAudioBlackList(string openId)
```

参数	类型	含义
openId	string	需移除黑名单的用户 openid

示例代码

```
ITMGContext.GetInstance().GetAudioCtrl ().RemoveAudioBlackList (openId);
```

实时语音采集相关接口

- 初始化 SDK 之后进房，在房间中，才可以调用实时音频语音相关接口。
- 当用户界面单击打开/关闭麦克风/扬声器按钮时，建议采用 EnableMic 以及 EnableSpeaker 接口进行调用。
- 当用户界面按住麦克风按钮时发言，放开按钮不发言，建议采用进房时候调用 EnableAudioCaptureDevice 一次，后续按住发言调用 EnableAudioSend 来实现。

接口	接口含义
EnableMic	开关麦克风
GetMicState	获取麦克风状态

EnableAudioCaptureDevice	开关采集设备
IsAudioCaptureDeviceEnabled	获取采集设备状态
EnableAudioSend	打开关闭音频上行
IsAudioSendEnabled	获取音频上行状态
GetMicLevel	获取实时麦克风音量
GetSendStreamLevel	获取音频上行实时音量
SetMicVolume	设置麦克风音量
GetMicVolume	获取麦克风音量

开启或关闭麦克风

此接口用来开启关闭麦克风。加入房间默认不打开麦克风及扬声器。EnableMic = EnableAudioCaptureDevice + EnableAudioSend

接口原型

```
ITMGAudioCtrl EnableMic(bool isEnabled)
```

参数	类型	含义
isEnabled	boolean	如果需要打开麦克风，则传入的参数为 true，如果关闭麦克风，则参数为 false

示例代码

```
//打开麦克风
ITMGContext.GetInstance().GetAudioCtrl().EnableMic(true);
```

麦克风状态获取

此接口用于获取麦克风状态，返回值0为关闭麦克风状态，返回值1为打开麦克风状态。

接口原型

```
ITMGAudioCtrl GetMicState()
```

示例代码

```
micToggle.isOn = ITMGContext.GetInstance().GetAudioCtrl().GetMicState();
```

开启或关闭采集设备

此接口用来开启/关闭采集设备。加入房间默认不打开设备。

- 只能在进房后调用此接口，退房会自动关闭设备。
- 在移动端，打开采集设备通常会伴随权限申请，音量类型调整等操作。

接口原型

```
ITMGAudioCtrl int EnableAudioCaptureDevice(bool isEnabled)
```

参数	类型	含义
isEnabled	bool	如果需要打开采集设备，则传入的参数为 true，如果关闭采集设备，则参数为 false

示例代码

```
//打开采集设备
ITMGContext.GetInstance().GetAudioCtrl().EnableAudioCaptureDevice(true);
```

采集设备状态获取

此接口用于采集设备状态获取。

接口原型

```
ITMGAudioCtrl bool IsAudioCaptureDeviceEnabled()
```

示例代码

```
bool IsAudioCaptureDevice =
ITMGContext.GetInstance().GetAudioCtrl().IsAudioCaptureDeviceEnabled();
```

打开或关闭音频流发送

此接口用于打开/关闭音频流发送。如果采集设备已经打开，那么会发送采集到的音频数据。如果采集设备没有打开，那么仍旧无声。采集设备的打开关闭请参见接口 EnableAudioCaptureDevice。

接口原型

```
ITMGAudioCtrl int EnableAudioSend(bool isEnabled)
```

参数	类型	含义
isEnabled	bool	如果需要打开音频上行，则传入的参数为 true，如果关闭音频上行，则参数为 false

示例代码

```
ITMGContext.GetInstance().GetAudioCtrl().EnableAudioSend(true);
```

音频流发送状态获取

此接口用于音频流发送状态获取。

接口原型

```
ITMGAudioCtrl bool IsAudioSendEnabled()
```

示例代码

```
bool IsAudioSend = ITMGContext.GetInstance().GetAudioCtrl().IsAudioSendEnabled();
```

版权所有：腾讯云计算（北京）有限责任公司

获取麦克风实时音量

此接口用于获取麦克风实时音量，返回值为 int 类型。建议20ms获取一次。值域为0 – 100，通过此接口可以获取到麦克风采集到的实时音量情况。

接口原型

```
ITMGAudioCtrl int GetMicLevel
```

示例代码

```
ITMGContext.GetInstance().GetAudioCtrl().GetMicLevel();
```

获取音频流发送实时音量

此接口用于获取自己音频流发送实时音量，返回值为 int 类型，取值范围为0 – 100。

接口原型

```
ITMGAudioCtrl int GetSendStreamLevel()
```

示例代码

```
int Level = ITMGContext.GetInstance().GetAudioCtrl().GetSendStreamLevel();
```

设置麦克风软件音量

此接口用于设置麦克风的音量。参数 volume 用于设置麦克风的音量，相当于对采集的声音做衰减或增益。

接口原型

```
ITMGAudioCtrl SetMicVolume(int volume)
```

参数	类型	含义
volume	int	取值范围为 0–200，数值为0的时候表示静音，当数值为100的时候表示音量不增不减，默认数值为100。

示例代码

```
int micVol = (int)(value * 100);
ITMGContext.GetInstance().GetAudioCtrl().SetMicVolume (micVol);
```

获取麦克风软件音量

此接口用于获取麦克风的音量。返回值为一个int类型数值，返回值为101代表没调用过接口 SetMicVolume。

接口原型

```
ITMGAudioCtrl GetMicVolume()
```

示例代码

```
ITMGContext.GetInstance().GetAudioCtrl().GetMicVolume();
```

实时语音播放相关接口

接口	接口含义
EnableSpeaker	开关扬声器
GetSpeakerState	获取扬声器状态
EnableAudioPlayDevice	开关播放设备
IsAudioPlayDeviceEnabled	获取播放设备状态
EnableAudioRecv	打开关闭音频下行
IsAudioRecvEnabled	获取音频下行状态
GetSpeakerLevel	获取实时扬声器音量
GetRecvStreamLevel	获取房间内其他成员下行实时音量
SetSpeakerVolume	设置扬声器音量
GetSpeakerVolume	获取扬声器音量
TrackingVolume	开始监听房间内用户的音量变化
StopTrackingVolume	停止监听房间内用户的音量变化

开启或关闭扬声器

此接口用于开启关闭扬声器。**EnableSpeaker = EnableAudioPlayDevice + EnableAudioRecv**

接口原型

```
ITMGAudioCtrl EnableSpeaker(bool isEnabled)
```

参数	类型	含义
isEnabled	bool	如果需要关闭扬声器，则传入的参数为 false，如果打开扬声器，则参数为 true

示例代码

```
//打开扬声器
ITMGContext.GetInstance().GetAudioCtrl().EnableSpeaker(true);
```

扬声器状态获取

此接口用于扬声器状态获取。返回值0为关闭扬声器状态，返回值1为打开扬声器状态。

接口原型

```
ITMGAudioCtrl GetSpeakerState()
```

示例代码

```
speakerToggle.isOn = ITMGContext.GetInstance().GetAudioCtrl().GetSpeakerState();
```

开启或关闭播放设备

此接口用于开启关闭播放设备。

接口原型

```
ITMGAudioCtrl EnableAudioPlayDevice(bool isEnabled)
```

参数	类型	含义
isEnabled	bool	如果需要关闭播放设备，则传入的参数为 false，如果打开播放设备，则参数为 true

示例代码

```
ITMGContext.GetInstance().GetAudioCtrl().EnableAudioPlayDevice(true);
```

播放设备状态获取

此接口用于播放设备状态获取。

接口原型

```
ITMGAudioCtrl bool IsAudioPlayDeviceEnabled()
```

示例代码

```
bool IsAudioPlayDevice = ITMGContext.GetInstance().GetAudioCtrl().IsAudioPlayDeviceEnabled();
```

打开或关闭音频流接收

此接口用于打开/关闭音频流接收。如果播放设备已经打开，那么会播放房间里其他人的音频数据。如果播放设备没有打开，那么仍旧无声。播放设备的打开关闭参见接口请参见 EnableAudioPlayDevice。

接口原型

```
ITMGAudioCtrl int EnableAudioRecv(bool isEnabled)
```

参数	类型	含义
isEnabled	bool	如果需要打开音频下行，则传入的参数为 true，如果关闭音频下行，则参数为 false

示例代码

```
ITMGContext.GetInstance().GetAudioCtrl().EnableAudioRecv(true);
```

音频流接收状态获取

此接口用于音频流接收状态获取。

接口原型

```
ITMGAudioCtrl bool IsAudioRecvEnabled()
```

示例代码

```
bool IsAudioRecv = ITMGContext.GetInstance().GetAudioCtrl().IsAudioRecvEnabled();
```

获取扬声器实时音量

此接口用于获取扬声器实时音量。返回值为 int 类型数值，表示扬声器实时音量。建议20ms获取一次。

接口原型

```
ITMGAudioCtrl GetSpeakerLevel()
```

示例代码

```
ITMGContext.GetInstance().GetAudioCtrl().GetSpeakerLevel();
```

获取房间内其他成员音频流音量

此接口用于获取接收到的房间内其他成员音频流实时音量，返回值为 int 类型，取值范围为0 – 200。

接口原型

```
ITMGAudioCtrl int GetRecvStreamLevel(string openId)
```

参数	类型	含义
openId	string	房间其他成员的 openId

示例代码

```
int Level = ITMGContext.GetInstance().GetAudioCtrl().GetRecvStreamLevel(openId);
```

动态设置房间内某成员音量

此接口用于设置房间内某成员的说话音量大小，此设置只在本端生效。

接口原型

```
public abstract int SetSpeakerVolumeByOpenID(string openid, int volume);
```

参数	类型	含义
openId	string	需要调节音量大小的OpenID
volume	int	百分比，建议[0-200]，其中100为默认值

设置扬声器的音量

此接口用于设置扬声器的音量。

接口原型

```
ITMGAudioCtrl SetSpeakerVolume(int volume)
```

参数	类型	含义
volume	int	设置音量，范围0 – 200，当数值为0时，表示静音，当数值为100时，表示音量不增不减，默认数值为100。

示例代码

```
int speVol = (int)(value * 100);
ITMGContext.GetInstance().GetAudioCtrl().SetSpeakerVolume(speVol);
```

获取扬声器的音量

此接口用于获取扬声器的音量。返回值为 int 类型数值，代表扬声器的音量，返回值为101代表没调用过接口 SetSpeakerVolume。Level 是实时音量，Volume 是扬声器的音量，最终声音音量 = Level × Volume %。例如实时音量是数值是100，此时 Volume 的数值是60，那么最终发出来的声音数值也是60。

接口原型

```
ITMGAudioCtrl GetSpeakerVolume()
```

示例代码

```
ITMGContext.GetInstance().GetAudioCtrl().GetSpeakerVolume();
```

开始监听房间内用户的音量变化

开始监听房间类用户（包括自己）的音量变化，当音量有变化时会触发ITMG_MAIN_EVENT_TYPE_USER_VOLUMES事件回调

接口原型

```
int TrackingVolume(float trackingTimeS);
```

参数	类型	含义
trackingTimeS	float	音量变化事件回调的间隔，单位为秒

消息	含义	data
ITMG_MAIN_EVENT_TYPE_USER_VOLUMES	房间内用户音量发生变化	[{"open_id":"abc", "volume":2}, {"open_id":"def", "volume":2}]

示例代码

```
ITMGContext.GetInstance().GetAudioCtrl().TrackingVolume(1.0);
```

停止监听房间内用户的音量变化

停止监听房间内用户（包括自己）的音量变化

接口原型

```
int StopTrackingVolume();
```

示例代码

```
ITMGContext.GetInstance().GetAudioCtrl().StopTrackingVolume();
```

设备选择相关接口

设备选择相关接口只能在 PC 端上使用。

接口	接口含义
GetMicListCount	获取麦克风设备数量
GetMicList	枚举麦克风设备
GetSpeakerListCount	获取扬声器设备数量
GetSpeakerList	枚举扬声器设备
SelectMic	选定麦克风设备
SelectSpeaker	选定扬声器设备

获取麦克风设备数量

此接口用来获取麦克风设备数量。

函数原型

```
public abstract int GetMicListCount()
```

示例代码

```
ITMGContext.GetInstance().GetAudioCtrl().GetMicListCount();
```

枚举麦克风设备

此接口用来枚举麦克风设备。配合 GetMicListCount 接口使用。

函数原型

```
public abstract int GetMicList(out List<TMGAudioDeviceInfo> devicesInfo, int count)
```

参数	类型	含义
devicesInfo	List<TMGAudioDeviceInfo>	设备列表
count	int	获取的麦克风设备数量

TMGAudioDeviceInfo 类型：

参数	类型	含义
m_strDeviceName	string	设备名称
m_strDeviceID	string	设备 ID

示例代码

```
ITMGContext.GetInstance().GetAudioCtrl().GetMicList(devicesInfo,count);
```

选中麦克风设备

此接口用来选中麦克风设备。如果不调用或者传入 "DEVICEID_DEFAULT"，则选中系统默认设备。
GetMicList 接口中返回的第0个设备 ID 为默认设备，未选中设备时通话设备为默认设备，选中后业务层维护通话设备。如果该通话设备被拔出，则此时通话设备为默认设备，拔出的通话设备插入后，此时通话设备恢复为插入的通话设备。

函数原型

```
public abstract int SelectMic(string micID);
```

参数	类型	含义
micID	string	麦克风设备 ID，设备 ID 来自于 GetMicList 返回列表

示例代码

```
string deviceID = DEVICE_ID_DEFAULT;
                if (index != 0)
                {
                    deviceID = listMicInfo[index - 1].m_strDeviceID;
                }
                ITMGContext.GetInstance().GetAudioCtrl().SelectMic(deviceID);
                selectedMicID = deviceID;
```

此接口用来获取扬声器设备数量。

函数原型

```
public abstract int GetSpeakerListCount();
```

示例代码

```
ITMGContext.GetInstance().GetAudioCtrl().GetSpeakerListCount();
```

枚举扬声器设备

此接口用来枚举扬声器设备。配合 GetSpeakerListCount 接口使用。

函数原型

```
public abstract int GetSpeakerList(out List<TMGAudioDeviceInfo> devicesInfo, int count)
```

参数	类型	含义
devicesInfo	List<TMGAudioDeviceInfo>	设备列表
count	int	获取的扬声器设备数量

TMGAudioDeviceInfo类型：

参数	类型	含义
m_strDeviceName	string	设备名称
m_strDeviceID	string	设备 ID

示例代码

```
int speakerCount = ITMGContext.GetInstance().GetAudioCtrl().GetSpeakerListCount();
Debug.LogFormat("speakerCount = {0}", speakerCount);
if (speakerCount > 0)
    {
        int ret = ITMGContext.GetInstance().GetAudioCtrl().GetSpeakerList(out listSpeakerInfo,
speakerCount);
        Debug.LogFormat("GetSpeakerList ret = {0}", ret);
        if (ret != 0)
        {
            listSpeakerInfo = null;
        }
    }
}
```

选中扬声器设备

此接口用来选中播放设备。如果不调用或者传入 "DEVICEID_DEFAULT"，则选中系统默认播放设备。

函数原型

```
public abstract int SelectSpeaker(string speaker);
```

参数	类型	含义
speaker	string	扬声器设备 ID，设备 ID 来自于 GetSpeakerList 返回列表。

示例代码

```
speakerDropdown = transform.Find("DevicePanel/SpeakerSelect").GetComponent<Dropdown>();
        if (speakerDropdown != null)
        {
            speakerDropdown.onValueChanged.AddListener(delegate (int index)
            {
                string deviceID = DEVICE_ID_DEFAULT;
                if (index != 0)
```

```
                {
                    deviceID = listSpeakerInfo[index - 1].m_strDeviceID;
                }
                ITMGContext.GetInstance().GetAudioCtrl().SelectSpeaker(deviceID);
                selectedSpeakerID = deviceID;
            });
        }
```

特殊接口

启动耳返

此接口用于启动耳返，需要 EnableLoopBack+EnableSpeaker 才可以听到自己声音。

接口原型

```
ITMGContext GetAudioCtrl EnableLoopBack(bool enable)
```

参数	类型	含义
enable	bool	设置是否启动

示例代码

```
ITMGContext.GetInstance().GetAudioCtrl().EnableLoopBack(true);
```

设备占用和释放事件回调

在房间内，占用设备和释放设备时会回调，通过委托传递事件的相关消息。

```
public delegate void QAVOnDeviceStateChangedEvent(int deviceType, string deviceId, bool
openOrClose);
public abstract event QAVOnDeviceStateChangedEvent OnDeviceStateChangedEvent;
```

参数	类型	含义
deviceType	int	• 1：代表采集设备 • 2：代表播放设备
deviceId	string	设备 GUID，用于标记设备，仅在 Windows 端和 Mac 端有效
openOrClose	bool	采集设备/播放设备占用或者释放

示例代码

对事件进行监听：

```
ITMGContext.GetInstance().GetAudioCtrl().OnDeviceStateChangedEvent += new
QAVAudioDeviceStateCallback(OnAudioDeviceStateChange);
```

监听处理：

```
void QAVAudioDeviceStateCallback(int deviceType, string deviceId, bool openOrClose){
    //设备占用和释放事件相关回调处理
}
```

获取用户房间音频类型

此接口用于获取用户房间音频类型，返回值为房间音频类型，返回值为0时代表获取用户房间音频类型发生错误，房间音频类型参考 EnterRoom 接口。

接口原型

```
ITMGContext ITMGRoom public int GetRoomType()
```

示例代码

```
ITMGContext.GetInstance().GetRoom().GetRoomType();
```

房间类型修改

此接口用于修改用户房间音频类型，结果参见回调事件，事件类型为 ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE。房间的音频类型由第一个进房的人确定，此后房间里有成员修改房间类型，将对此房间所有成员生效。

接口原型

```
ITMGContext ITMGRoom public int ChangeRoomType(ITMGRoomType roomtype)
```

参数	类型	含义
roomtype	ITMGRoomType	房间切换成的目标类型，房间音频类型参考 EnterRoom 接口

示例代码

```
ITMGContext.GetInstance().GetRoom().ChangeRoomType(ITMG_ROOM_TYPE_FLUENCY);
```

回调事件

主动设置房间类型，房间类型设置后，通过委托传递修改完成的相关消息。

返回的参数	含义
roomtype	返回切换后的 roomtype 类型

```
public abstract event QAVCallback OnChangeRoomtypeCallback;
public abstract event QAVOnRoomTypeChangedEvent OnRoomTypeChangedEvent;
```

示例代码

```
//对事件进行监听：
ITMGContext.GetInstance ().OnRoomTypeChangedEvent += new QAVOnRoomTypeChangedEvent
(OnRoomTypeChangedEvent);
//监听处理：
void OnRoomTypeChangedEvent(int roomtype)
{
        ShowWarnning (string.Format ("RoomTypeChanged current:{0}",roomtype));
}
```

房间类型变化通知

用户主动修改房间类型，或者房间内其它用户修改房间类型，只要房间类型发生变化，就会通过通知事件告知业务层房间类型发生变化，返回的是房间类型，参考 EnterRoom 接口。

```
public delegate void QAVOnRoomTypeChangedEvent(int roomtype);
public abstract event QAVOnRoomTypeChangedEvent OnRoomTypeChangedEvent;
```

示例代码

```
//对事件进行监听：
ITMGContext.GetInstance().OnRoomTypeChangedEvent += new
QAVOnRoomTypeChangedEvent(OnRoomTypeChangedEvent);
//监听处理：
void OnRoomTypeChangedEvent(int roomtype){
    //房间类型改变后的处理
}
```

房间通话质量监控事件

质量监控事件，此通知事件适用于监听网络质量，如果用户网络差的话，业务层将通过 UI 提醒用户切换网络。在进房后触发，事件2秒回调一次，事件消息为 ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_QUALITY，返回的参数为 weight、loss 及 delay，代表的信息如下：

参数	类型	含义
weight	int	范围是1 – 50，数值为50是音质评分极好，数值为1是音质评分很差，几乎不能使用，数值为0代表初始值，无含义。一般数值在30以下就可以提醒用户网络较差，建议切换网络。
loss	double	上行丢包率。
delay	int	音频触达延迟时间（ms）。

获取版本号

获取 SDK 版本号，用于分析。

接口原型

```
ITMGContext  abstract string GetSDKVersion()
```

示例代码

```
ITMGContext.GetInstance().GetSDKVersion();
```

设置打印日志等级

用于设置打印日志等级。建议保持默认等级。需要在 Init 之前调用。

接口原型

```
ITMGContext  SetLogLevel(ITMG_LOG_LEVEL levelWrite, ITMG_LOG_LEVEL levelPrint)
```

参数含义

参数	类型	含义
levelWrite	ITMG_LOG_LEVEL	设置写入日志的等级，TMG_LOG_LEVEL_NONE 表示不写入，默认为 TMG_LOG_LEVEL_INFO
levelPrint	ITMG_LOG_LEVEL	设置打印日志的等级，TMG_LOG_LEVEL_NONE 表示不打印，默认为 TMG_LOG_LEVEL_ERROR

ITMG_LOG_LEVEL 说明如下：

ITMG_LOG_LEVEL	含义
TMG_LOG_LEVEL_NONE	不打印日志
TMG_LOG_LEVEL_ERROR	打印错误日志（默认）
TMG_LOG_LEVEL_INFO	打印提示日志
TMG_LOG_LEVEL_DEBUG	打印开发调试日志
TMG_LOG_LEVEL_VERBOSE	打印高频日志

示例代码

```
ITMGContext.GetInstance().SetLogLevel(TMG_LOG_LEVEL_INFO,TMG_LOG_LEVEL_INFO);
```

设置打印日志路径

用于设置打印日志路径。默认路径如下。需要在 Init 之前调用。

平台	路径
Windows	%appdata%\GMEGLOBAL\GME\ProcessName
iOS	Application/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/Documents
Android	/sdcard/Android/data/xxx.xxx.xxx/files
Mac	/Users/username/Library/Containers/xxx.xxx.xxx/Data/Documents

接口原型

```
ITMGContext  SetLogPath(string logDir)
```

参数	类型	含义
logDir	string	路径

示例代码

```
ITMGContext.GetInstance().SetLogPath(path);
```

获取诊断信息

获取音视频通话的实时通话质量的相关信息。该接口主要用来查看实时通话质量、排查问题等，业务侧可以忽略。

接口原型

```
ITMGRoom GetQualityTips()
```

示例代码

```
string tips = ITMGContext.GetInstance().GetRoom().GetQualityTips();
```

语音消息及转文本

最近更新时间：2024-12-02 18:05:32

为方便 Unity 开发者调试和接入腾讯云游戏多媒体引擎客户端 API，本文为您介绍适用于 Unity 语音消息服务及转文本服务的接入技术文档。

使用 GME 重要事项

GME 提供实时语音服务、语音消息服务及转文本服务，使用 GME 服务都依赖 Init 和 Poll 等核心接口。

重点提示

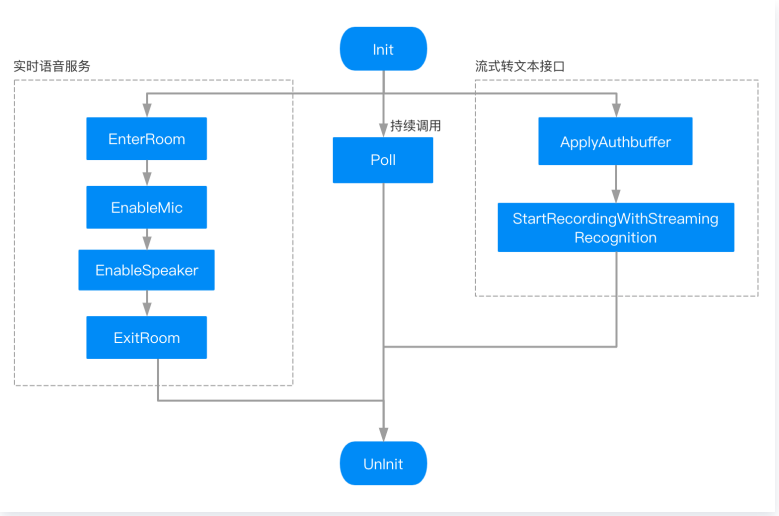
- 已完成 GME 应用创建，并获取 SDK AppID 和 Key。请参见 服务开通指引 。
- 已开通 **GME 实时语音服务、语音消息服务以及转文本服务**。请参见 服务开通指引 。
- GME 使用前请对工程进行配置，否则 SDK 不生效。
- GME 的接口调用成功后返回值为 QAVError.OK，数值为 0。
- GME 的接口调用要在同一个线程下。
- GME 需要周期性的调用 Poll 接口触发事件回调。
- 错误码详情可参见 错误码 。

> **注意**
> 语音转文本相关接口有默认频率限制，限额范围内计费方式请参见 计费文档；若需提升接口频率限额或了解超额计费方式，请联系商务或 提交工单咨询 。
> - 语音消息非流式转文本接口 **SpeechToText()**：默认单账号限制并发数为10路
> - 语音消息流式转文本接口 **StartRecordingWithStreamingRecognition()**：默认单账号限制并发数为50路
> - 实时语音流式转文本接口 **StartRealTimeASR()**：默认单账号限制并发数为50路

接入 SDK

重要步骤

接入 SDK 重要流程如下：



1. 初始化 GME
2. 周期性调用 Poll 触发回调
3. 鉴权初始化
4. 启动流式语音识别

5. 停止录制
6. 反初始化 GME

C# 类

类	含义
ITMGContext	核心接口
ITMGPTT	语音消息及转文本相关接口

核心接口

接口	接口含义
Init	初始化 GME
Poll	触发事件回调
Pause	系统暂停
Resume	系统恢复
Uninit	反初始化 GME

引用头文件

```
using GME;
```

获取实例

请使用 ITMGContext 的方法获取 Context 实例，不要直接使用 QAVContext.GetInstance() 去获取实例。

初始化 SDK

未初始化前，SDK 处于未初始化阶段，**需要通过接口 Init 初始化 SDK**，才可以使用实时语音服务、语音消息服务及转文本服务。调用 Init 接口的线程必须于其他接口在同一线程,建议都在主线程调用接口。

接口原型

```
//class ITMGContext
public abstract int Init(string sdkAppID, string openID);
```

参数	类型	含义
sdkAppId	string	来自 腾讯云控制台 的 GME 服务提供的 AppID，获取请参见 语音服务开通指引 。
openID	string	openID 只支持 Int64 类型（转为 string 传入），规则由 App 开发者自行制定，App 内不重复即可。如需使用字符串作为 Openid 传入，可 提交工单 联系开发者。

返回值

返回值	处理
QAVError.OK= 0	初始化 SDK 成功

AV_ERR_SDK_NOT_FULL_UPDATE=7015	检查 SDK 文件是否完整，建议删除后重新导入 SDK

⚠ 关于7015错误提示

- 7015错误码是通过 md5 进行判断，在接入过程中若出现此错误，请根据提示检查 SDK 文件是否完整、SDK 文件版本是否一致。
- 出现返回值 AV_ERR_SDK_NOT_FULL_UPDATE 时，此返回值**只有提示作用**，并不会造成初始化失败。
- 由于第三方加固、Unity 打包机制等因素会影响库文件 md5，造成误判，所以**正式发布请在逻辑中忽略此错误**，并尽量不在 UI 中提示。

示例代码

```
int ret = ITMGContext.GetInstance().Init(sdkAppId, openID);
//通过返回值判断是否初始化成功
if (ret != QAVError.OK)
    {
        Debug.Log("SDK初始化失败:"+ret);
        return;
    }
```

触发事件回调

通过在 update 里面周期的调用 Poll 可以触发事件回调。Poll 是 GME 的消息泵，GME 需要周期性的调用 Poll 接口触发事件回调。如果没有调用 Poll ，将会导致整个 SDK 服务运行异常。详情请参见 Sample Project 中的 EnginePollHelper 文件。

⚠ 注意

务必周期性调用 Poll 接口且在主线程调用，以免接口回调异常。

接口原型

```
ITMGContext public abstract int Poll();
```

示例代码

```
public void Update()
    {
        ITMGContext.GetInstance().Poll();
    }
```

系统暂停

当系统发生 Pause 事件时，需要同时通知引擎进行 Pause。例如在应用退后台时候（OnApplicationPause, isPause=True），如果不需要后台播放房间内声音，请调用 Pause 接口暂停整个 GME 服务。

接口原型

```
ITMGContext public abstract int Pause()
```

系统恢复

当系统发生 Resume 事件时，需要同时通知引擎进行 Resume。Resume 接口只恢复实时语音。

接口原型

```
ITMGContext  public abstract int Resume()
```

反初始化 SDK

反初始化 SDK，进入未初始化状态。**如果游戏业务侧账号与 openid 是绑定的，那切换游戏账号需要反初始化 GME，再用新的 openid 初始化。**

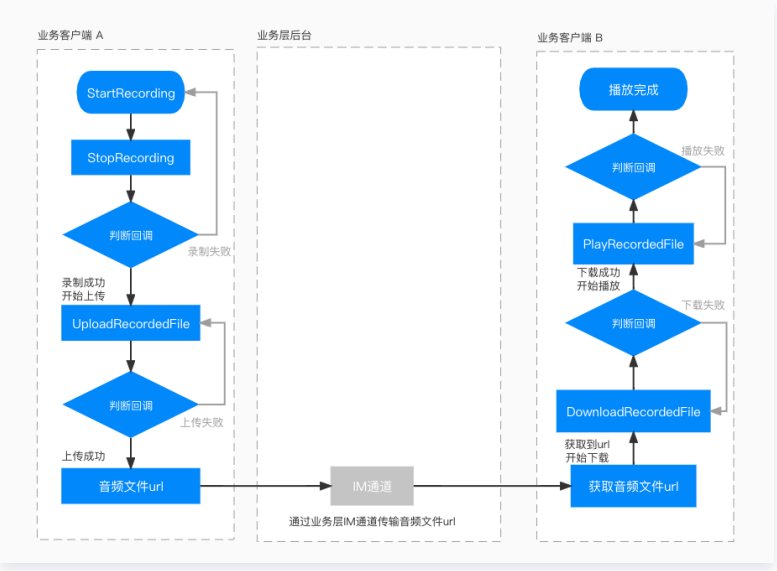
接口原型

```
ITMGContext public abstract int Uninit()
```

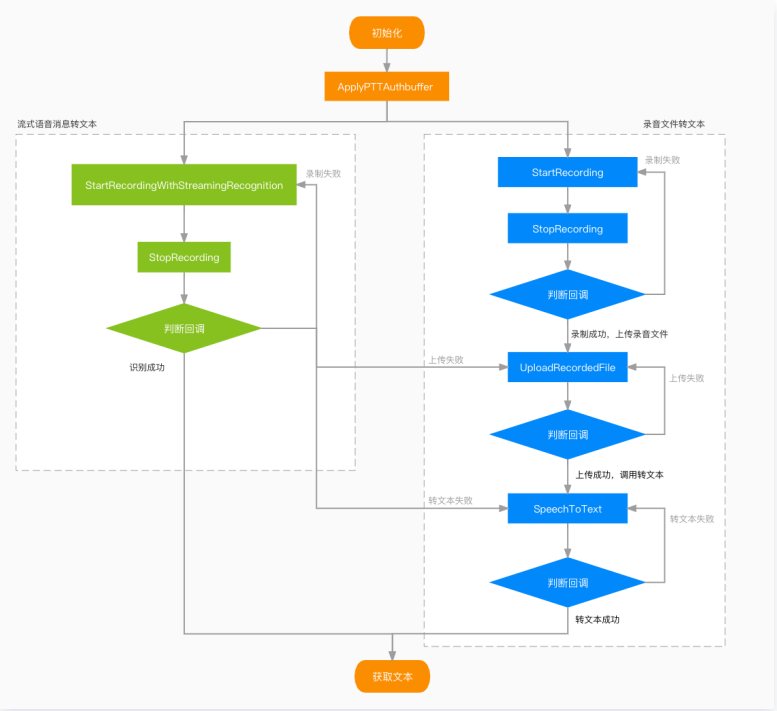
语音消息服务及转文本服务

> **说明**
>
> - 转文本服务分录音文件极速转文本以及语音消息流式转文本。
> - 使用语音消息服务不需要进入实时语音房间。
> - 语音消息最大录制时长默认为58秒，最短不能小于1秒。如果需要再加以限制，例如限制为最大录制时长为10秒，请在初始化之后调用 SetMaxMessageLength 接口进行设置。

语音消息服务使用流程



转文本服务使用流程



接口	接口含义
GenAuthBuffer	生成本地鉴权
ApplyPTTAuthbuffer	鉴权初始化
SetMaxMessageLength	限制最大语音信息时长

生成本地鉴权

生成 AuthBuffer，用于相关功能的加密和鉴权，如正式发布请使用后台部署密钥，后台部署请参见 鉴权密钥 。

接口原型

```
QAVAuthBuffer GenAuthBuffer(int appId, string roomId, string openId, string key)
```

参数	类型	含义
appId	int	来自腾讯云控制台的 AppId 号码。
roomId	string	填 null 或者空字符串。
openId	string	用户标识。与 Init 时候的 OpenId 相同。
key	string	来自腾讯云 控制台 的权限密钥。

应用鉴权

生成鉴权信息后，将鉴权赋值到 SDK 中。

接口原型

```
ITMGPTT int ApplyPTTAuthbuffer (byte[] authBuffer)
```

参数	类型	含义
authBuffer	byte[]	鉴权

示例代码

```
UserConfig.SetAppID(transform.Find ("appId").GetComponent<InputField> ().text);
UserConfig.SetUserID(transform.Find ("userId").GetComponent<InputField> ().text);
UserConfig.SetAuthKey(transform.Find("authKey").GetComponent<InputField>().text);
byte[] authBuffer = UserConfig.GetAuthBuffer(UserConfig.GetAppID(), UserConfig.GetUserID(),
null,UserConfig.GetAuthKey());
ITMGContext.GetInstance ().GetPttCtrl ().ApplyPTTAuthbuffer(authBuffer);
```

限制最大语音信息时长

限制最大语音消息的长度，最大支持58秒。

接口原型

```
ITMGPTT int SetMaxMessageLength(int msTime)
```

参数	类型	含义
msTime	int	语音时长，单位 ms，区间为 1000 < msTime < = 58000

示例代码

```
ITMGContext.GetInstance().GetPttCtrl().SetMaxMessageLength(58000);
```

流式语音识别

语音消息及转文字相关接口

接口	接口含义
StartRecordingWithStreamingRecognition	启动流式录音
StopRecording	停止录音

启动流式语音识别

此接口用于启动流式语音识别，同时在回调中会有实时的语音转文字返回，可以指定语言进行识别，也可以将语音中识别到的信息翻译成指定的语言返回。停止录音调用 停止录制 。

接口原型

```
ITMGPTT int StartRecordingWithStreamingRecognition(string filePath)
ITMGPTT int StartRecordingWithStreamingRecognition(string filePath, string speechLanguage,string
translateLanguage)
```

参数	类型	含义

filePath	String	存放的语音路径
speechLanguage	String	识别成指定文字的语言参数，参数请参见 语言参数参考列表
translateLanguage	String	填入与 speechLanguage 相同的值

示例代码

```
string recordPath = Application.persistentDataPath + string.Format("/{0}.silk", sUid++);
int ret =
ITMGContext.GetInstance().GetPttCtrl().StartRecordingWithStreamingRecognition(recordPath, "cmn-Hans-CN","cmn-Hans-CN");
```

> **注意**
> 翻译会收取额外费用，请参见 购买指南 。

流式语音识别的回调

启动流式语音识别后，需要通过在 OnStreamingSpeechComplete 或者 OnStreamingSpeechisRunning 通知中监听回调消息，事件消息分为以下两个：

- `OnStreamingSpeechComplete` 是在停止录制并完成识别后才返回文字，相当于一段话说完才会返回识别的文字。
- `OnStreamingSpeechisRunning` 是在录音过程中就会实时返回识别到的文字，相当于边说话边返回识别到的文字。

根据需求在 OnEvent 通知中对相应事件消息进行判断。传递的参数包含以下四个信息。

消息名称	含义
result	用于判断流式语音识别是否成功的返回码
text	语音转文字识别的文本
file_path	录音存放的本地地址
file_id	录音在后台的 url 地址，录音在服务器存放90天

> **注意**
> 监听 `ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_IS_RUNNING` 消息时，file_id 为空。

错误码

错误码	含义	处理方式
32775	流式语音转文本失败，但是录音成功	调用 UploadRecordedFile 接口上传录音，再调用 SpeechToText 接口进行语音转文字操作
32777	流式语音转文本失败，但是录音成功，上传成功	返回的信息中有上传成功的后台 url 地址，调用 SpeechToText 接口进行语音转文字操作
32786	流式语音转文本失败	在流式录制状态当中，请等待流式录制接口执行结果返回
32787	转文本成功，文本翻译服务未开通	需要在控制台开通文本翻译服务
32788	转文本成功，文本翻译语言参数不支持	重新检查传入参数

如果出现 4098 错误码，请参见 常见问题文档 进行解决。

示例代码

```
            //对事件进行监听：
            ITMGContext.GetInstance().GetPttCtrl().OnStreamingSpeechComplete +=new
QAVStreamingRecognitionCallback (OnStreamingSpeechComplete);
            ITMGContext.GetInstance().GetPttCtrl().OnStreamingSpeechisRunning += new
QAVStreamingRecognitionCallback (OnStreamingRecisRunning);
            //监听处理：
            void OnStreamingSpeechComplete(int code, string fileid, string filepath, string
result){
                    //启动流式语音识别的回调
            }

            void OnStreamingRecisRunning(int code, string fileid, string filePath, string
result){
                    if (code == 0)
                    {
                        setBtnText(mStreamBtn, "流式");
                        InputField field =
transform.Find("recordFilePath").GetComponent<InputField>();
                        field.text = filePath;

                        field = transform.Find("downloadUrl").GetComponent<InputField>();
                        field.text = "Stream is Running";

                        field = transform.Find("convertTextResult").GetComponent<InputField>();
                        field.text = result;
                        showWarningText("录制中");
                    }
}
```

语音消息录制

录制的流程为：录音 > 停止录音 > 录音回调返回 > 启动下一次录音。

语音消息及转文字相关接口

接口	接口含义
StartRecording	启动录音
PauseRecording	暂停录音
ResumeRecording	恢复录音
StopRecording	停止录音
CancelRecording	取消录音

启动录音

此接口用于启动录音。

接口原型

```
ITMGPTT int StartRecording(string fileDir)
```

参数	类型	含义
fileDir	string	存放的语音路径

示例代码

```
string recordPath = Application.persistentDataPath + string.Format ("/{0}.silk", sUid++);
int ret = ITMGContext.GetInstance().GetPttCtrl().StartRecording(recordPath);
```

停止录音

此接口用于停止录音。此接口为异步接口，停止录音后会有录音完成回调，成功之后录音文件才可用。

接口原型

```
ITMGPTT int StopRecording()
```

示例代码

```
ITMGContext.GetInstance().GetPttCtrl().StopRecording();
```

启动录音的回调

录音完成的回调，通过委托传递消息。
停止录音调用StopRecording。停止录音后才有启动录音的回调。

接口原型

```
public delegate void QAVRecordFileCompleteCallback(int code, string filepath);
public abstract event QAVRecordFileCompleteCallback OnRecordFileComplete;
```

参数	类型	含义
code	string	当 code 为 0 时，录制完成
filepath	string	录制的存放地址，必须是可以访问到的路径，不可将 fileid 作为路径

错误码

错误码值	原因	建议方案
4097	参数为空	检查代码中接口参数是否正确
4098	初始化错误	检查设备是否被占用，或者权限是否正常，是否初始化正常
4099	正在录制中	确保在正确的时机使用 SDK 录制功能
4100	没有采集到音频数据	检查麦克风设备是否正常

版权所有：腾讯云计算（北京）有限责任公司

4101	录音时，录制文件访问错误	确保文件存在，文件路径的合法性
4102	麦克风未授权错误	使用 SDK 需要麦克风权限，添加权限请参考对应引擎或平台的 SDK 工程配置文档
4103	录音时间太短错误	首先，限制录音时长的单位为毫秒，检查参数是否正确；其次，录音时长要1000毫秒以上才能成功录制
4104	没有启动录音操作	检查是否已经调用启动录音接口

示例代码

```
//对事件进行监听
ITMGContext.GetInstance().GetPttCtrl().OnRecordFileComplete +=  new
QAVRecordFileCompleteCallback (OnRecordFileComplete);
//监听处理
void OnRecordFileComplete(int code, string filepath){
    //启动录音的回调
}
```

暂停录音

此接口用于暂停录音。如需恢复录音请调用接口 ResumeRecording。

接口原型

```
ITMGPTT int PauseRecording()
```

示例代码

```
ITMGContext.GetInstance().GetPttCtrl().PauseRecording();
```

恢复录音

此接口用于恢复录音。

接口原型

```
ITMGPTT int ResumeRecording()
```

示例代码

```
ITMGContext.GetInstance().GetPttCtrl().ResumeRecording();
```

取消录音

调用此接口取消录音。**取消之后没有回调**。

接口原型

```
ITMGPTT int CancelRecording()
```

示例代码

```
ITMGContext.GetInstance().GetPttCtrl().CancelRecording();
```

语音消息上传、下载及播放

接口	接口含义
UploadRecordedFile	上传语音文件
DownloadRecordedFile	下载语音文件
PlayRecordedFile	播放语音
StopPlayFile	停止播放语音
GetFileSize	语音文件的大小
GetVoiceFileDuration	语音文件的时长

上传语音文件

此接口用于上传语音文件。

接口原型

```
ITMGPTT int UploadRecordedFile (string filePath)
```

参数	类型	含义
filePath	String	上传的语音路径，此路径为本地路径

示例代码

```
ITMGContext.GetInstance().GetPttCtrl().UploadRecordedFile(filePath);
```

上传语音完成的回调

上传语音完成的回调，通过委托传递消息。

接口原型

```
public delegate void QAVUploadFileCompleteCallback(int code, string filepath, string fileid);
public abstract event QAVUploadFileCompleteCallback OnUploadFileComplete;
```

参数	类型	含义
code	int	当 code 为0时，录制完成
filepath	string	录制的存放地址
fileid	string	文件的 url 路径

错误码

错误码值	原因	建议方案

8193	上传文件时，文件访问错误	确保文件存在，文件路径的合法性
8194	签名校验失败错误	检查鉴权密钥是否正确，检查是否有初始化离线语音
8195	网络错误	检查设备网络是否可以正常访问外网环境
8196	获取上传参数过程中网络失败	检查鉴权是否正确，检查设备网络是否可以正常访问外网环境
8197	获取上传参数过程中回包数据为空	检查鉴权是否正确，检查设备网络是否可以正常访问外网环境
8198	获取上传参数过程中回包解包失败	检查鉴权是否正确，检查设备网络是否可以正常访问外网环境
8200	没有设置 appinfo	检查 apply 接口是否有调用，或者入参是否为空

示例代码

```
//对事件进行监听
ITMGContext.GetInstance().GetPttCtrl().OnUploadFileComplete +=new QAVUploadFileCompleteCallback
(OnUploadFileComplete);
//监听处理
void OnUploadFileComplete(int code, string filepath, string fileid){
    //上传语音完成的回调
}
```

下载语音文件

此接口用于下载语音文件。

接口原型

```
ITMGPTT DownloadRecordedFile (string fileID, string downloadFilePath)
```

参数	类型	含义
fileID	String	文件的 url 路径
downloadFilePath	String	文件的本地保存路径，必须是可以访问到的路径，不可将 fileid 作为路径

示例代码

```
ITMGContext.GetInstance().GetPttCtrl().DownloadRecordedFile(fileId, filePath);
```

下载语音文件完成回调

下载语音文件完成回调，通过委托传递消息。

接口原型

```
public delegate void QAVDownloadFileCompleteCallback(int code, string filepath, string fileid);
public abstract event QAVDownloadFileCompleteCallback OnDownloadFileComplete;
```

参数	类型	含义
code	int	当 code 为0时，录制完成

filepath	string	录制的存放地址
fileid	string	文件的 url 路径，录音在服务器存放 90 天

错误码

错误码值	原因	建议方案
12289	下载文件时，文件访问错误	检查文件路径是否合法
12290	签名校验失败	检查鉴权密钥是否正确，检查是否有初始化离线语音
12291	网络存储系统异常	服务器获取语音文件失败，检查接口参数 fileid 是否正确，检查网络是否正常，检查 COS 文件存不存在
12292	服务器文件系统错误	检查设备网络是否可以正常访问外网环境，检查服务器上是否有此文件
12293	获取下载参数过程中，HTTP 网络失败	检查设备网络是否可以正常访问外网环境
12294	获取下载参数过程中，回包数据为空	检查设备网络是否可以正常访问外网环境
12295	获取下载参数过程中，回包解包失败	检查设备网络是否可以正常访问外网环境
12297	没有设置 appinfo	检查鉴权密钥是否正确，检查是否有初始化离线语音

示例代码

```
//对事件进行监听
ITMGContext.GetInstance().GetPttCtrl().OnDownloadFileComplete +=new
QAVDownloadFileCompleteCallback(OnDownloadFileComplete);
//监听处理
void OnDownloadFileComplete(int code, string filepath, string fileid){
    //下载语音文件完成回调
}
```

播放语音

此接口用于播放语音。

接口原型

```
ITMGPTT PlayRecordedFile(string filePath)
ITMGPTT 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，请参见 变声接入文档

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

示例代码

```
ITMGContext.GetInstance().GetPttCtrl().PlayRecordedFile(filePath);
```

播放语音的回调

播放语音的回调，通过委托传递消息。

接口原型

```
public delegate void QAVPlayFileCompleteCallback(int code, string filepath);
public abstract event QAVPlayFileCompleteCallback OnPlayFileComplete;
```

参数	类型	含义
code	int	当 code 为0时，播放完成
filepath	string	录制的存放地址

错误码

错误码值	原因	建议方案
20481	初始化错误	检查设备是否被占用，或者权限是否正常，是否初始化正常
20482	正在播放中，试图打断并播放下一个失败了（正常是可以打断的）	检查代码逻辑是否正确
20483	参数为空	检查代码中接口参数是否正确
20484	内部错误	初始化播放器错误，解码失败等问题产生此错误码，需要结合日志定位问题

示例代码

```
//对事件进行监听：
ITMGContext.GetInstance().GetPttCtrl().OnPlayFileComplete +=new
QAVPlayFileCompleteCallback(OnPlayFileComplete);
//监听处理：
void OnPlayFileComplete(int code, string filepath){
    //播放语音的回调
}
```

停止播放语音

此接口用于停止播放语音。停止播放语音也会有播放完成的回调。

接口原型

```
ITMGPTT int StopPlayFile()
```

示例代码

```
ITMGContext.GetInstance().GetPttCtrl().StopPlayFile();
```

获取语音文件的大小

通过此接口，获取语音文件的大小。

接口原型

```
ITMGPTT GetFileSize(string filePath)
```

参数	类型	含义
filePath	String	语音文件的路径，此路径为本地路径

示例代码

```
int fileSize = ITMGContext.GetInstance().GetPttCtrl().GetFileSize(filepath);
```

获取语音文件的时长

此接口用于获取语音文件的时长，单位毫秒。

接口原型

```
ITMGPTT int GetVoiceFileDuration(string filePath)
```

参数	类型	含义
filePath	String	语音文件的路径，此路径为本地路径

示例代码

```
int fileDuration = ITMGContext.GetInstance().GetPttCtrl().GetVoiceFileDuration(filepath);
```

录音文件极速转文本

接口	接口含义
SpeechToText	语音识别成文字

将指定的语音文件识别成文字

此接口用于将指定的语音文件识别成文字。

接口原型

```
ITMGPTT int SpeechToText(String fileID)
```

参数	类型	含义

fileID	String	语音文件 url

示例代码

```
ITMGContext.GetInstance().GetPttCtrl().SpeechToText(fileID);
```

将指定的语音文件翻译成文字（指定语言）

此接口可以指定语言进行识别，也可以将语音中识别到的信息翻译成指定的语言返回。

> **注意**
> 翻译会收取额外费用，请参见 购买指南 。

接口原型

```
ITMGPTT int SpeechToText(String fileID,String speechLanguage)
ITMGPTT int SpeechToText(String fileID,String speechLanguage,String translatelanguage)
```

参数	类型	含义
fileID	String	语音文件 url，录音在服务器存放90天
speechLanguage	String	识别出指定文字的语言参数，参数参见 语言参数参考列表
translatelanguage	String	翻译成指定文字的语言参数，参数参见 语言参数参考列表 中的翻译语言参数

示例代码

```
ITMGContext.GetInstance().GetPttCtrl().SpeechToText(fileID,"cmn-Hans-CN","cmn-Hans-CN");
```

识别回调

将指定的语音文件识别成文字，通过委托传递消息。

接口原型

```
public delegate void QAVSpeechToTextCallback(int code, string fileid, string result);
public abstract event QAVSpeechToTextCallback OnSpeechToTextComplete;
```

参数	类型	含义
code	int	当 code 为0时，录制完成
fileid	string	语音文件 url，录音在服务器存放 90 天
result	string	转换的文本结果

错误码

错误码值	原因	建议方案
32769	内部错误	分析日志，获取后台返回给客户端的真正错误码，并联系后台同事协助解决

32770	网络失败	检查设备网络是否可以正常访问外网环境
32772	回包解包失败	分析日志，获取后台返回给客户端的真正错误码，并联系后台同事协助解决
32774	没有设置 appinfo	检查鉴权密钥是否正确，检查是否有初始化离线语音
32776	authbuffer 校验失败	检查 authbuffer 是否正确
32784	语音转文本参数错误	检查代码中接口参数 fileid 是否为空
32785	语音转文本翻译返回错误	离线语音后台错误，请分析日志，获取后台返回给客户端的真正错误码，并联系后台同事协助解决
32787	转文本成功，文本翻译服务未开通	需要在控制台开通文本翻译服务
32788	转文本成功，文本翻译语言参数不支持	重新检查传入参数

示例代码

```
//对事件进行监听
ITMGContext.GetInstance().GetPttCtrl().OnSpeechToTextComplete += new
QAVSpeechToTextCallback(OnSpeechToTextComplete);
//监听处理
void OnSpeechToTextComplete(int code, string fileid, string result){
    //识别回调
}
```

语音消息音量相关接口

接口	接口含义
GetMicLevel	获取实时麦克风音量
SetMicVolume	设置录制音量
GetMicVolume	获取录制音量
GetSpeakerLevel	获取实时扬声器音量
SetSpeakerVolume	设置播放音量
GetSpeakerVolume	获取播放音量

获取语音消息麦克风实时音量

此接口用于获取麦克风实时音量，返回值为 int 类型，值域为0 – 200。

接口原型

```
ITMGPTT int GetMicLevel()
```

示例代码

```
ITMGContext.GetInstance().GetPttCtrl().GetMicLevel();
```

设置语音消息录制音量

此接口用于设置离线语音录制音量，值域为0 – 200。

接口原型

```
ITMGPTT int SetMicVolume(int vol)
```

示例代码

```
ITMGContext.GetInstance().GetPttCtrl().SetMicVolume(100);
```

获取语音消息录制音量

此接口用于获取离线语音录制音量。返回值为 int 类型，值域为0 – 200。

接口原型

```
ITMGPTT int GetMicVolume()
```

示例代码

```
ITMGContext.GetInstance().GetPttCtrl().GetMicVolume();
```

获取语音消息扬声器实时音量

此接口用于获取扬声器实时音量。返回值为 int 类型，值域为0 – 200。

接口原型

```
ITMGPTT int GetSpeakerLevel()
```

示例代码

```
ITMGContext.GetInstance().GetPttCtrl().GetSpeakerLevel();
```

设置语音消息播放音量

此接口用于设置离线语音播放音量，值域为0 – 200。

接口原型

```
ITMGPTT int SetSpeakerVolume(int vol)
```

示例代码

```
ITMGContext.GetInstance().GetPttCtrl().SetSpeakerVolume(100);
```

获取语音消息播放音量

此接口用于获取离线语音播放音量。返回值为 int 类型，值域为0 – 200。

接口原型

```
ITMGPTT int GetSpeakerVolume()
```

示例代码

```
ITMGContext.GetInstance().GetPttCtrl().GetSpeakerVolume();
```

高级 API

获取版本号

获取 SDK 版本号，用于分析。

接口原型

```
ITMGContext  abstract string GetSDKVersion()
```

示例代码

```
ITMGContext.GetInstance().GetSDKVersion();
```

设置打印日志等级

用于设置打印日志等级。建议保持默认等级。需要在 Init 之前调用。

接口原型

```
ITMGContext  SetLogLevel(ITMG_LOG_LEVEL levelWrite, ITMG_LOG_LEVEL levelPrint)
```

参数含义

参数	类型	含义
levelWrite	ITMG_LOG_LEVEL	设置写入日志的等级，TMG_LOG_LEVEL_NONE 表示不写入，默认为 TMG_LOG_LEVEL_INFO
levelPrint	ITMG_LOG_LEVEL	设置打印日志的等级，TMG_LOG_LEVEL_NONE 表示不打印，默认为 TMG_LOG_LEVEL_ERROR

ITMG_LOG_LEVEL 说明如下：

ITMG_LOG_LEVEL	含义
TMG_LOG_LEVEL_NONE	不打印日志
TMG_LOG_LEVEL_ERROR	打印错误日志（默认）
TMG_LOG_LEVEL_INFO	打印提示日志
TMG_LOG_LEVEL_DEBUG	打印开发调试日志
TMG_LOG_LEVEL_VERBOSE	打印高频日志

示例代码

```
ITMGContext.GetInstance().SetLogLevel(TMG_LOG_LEVEL_INFO,TMG_LOG_LEVEL_INFO);
```

设置打印日志路径

用于设置打印日志路径。默认路径如下。需要在 Init 之前调用。

平台	路径
Windows	%appdata%\Tencent\GME\ProcessName
iOS	Application/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/Documents
Android	/sdcard/Android/data/xxx.xxx.xxx/files
Mac	/Users/username/Library/Containers/xxx.xxx.xxx/Data/Documents

接口原型

```
ITMGContext  SetLogPath(string logDir)
```

参数	类型	含义
logDir	String	路径

示例代码

```
ITMGContext.GetInstance().SetLogPath(path);
```

工程导出

最近更新时间：2025-03-28 10:09:22

为方便 Unity 开发者调试和接入腾讯云游戏多媒体引擎产品 API，这里向您介绍适用于 Unity 开发的工程导出配置。

导出 iOS 平台

从 Unity 工程导出成 Xcode 工程时，需要对 GME 动态库进行处理，处理手段根据 Unity 版本的区别而有所不同。

1. 动态库处理（Unity 2019及以上版本）

配置原理

新建一个 Editor OnPostprocessBuild 脚本，利用 UnityEditor.iOS.Xcode.Extensions.PBXProjectExtensions.AddFileToEmbedFrameworks，这个 API 会自动将动态库拷贝到最终出包 Bundle 的 framework 目录下，并为其签名。
业务层可根据所需的功能对动态库进行删减，根据动态库列表来决定示例代码中导入的 framework 列表。动态库功能具体参见 动态库目录 。

```
string[] framework_names = {
    "libgme_fdkaac.framework",
    "libgme_lamemp3.framework",
    "libgme_ogg.framework",
    "libgme_soundtouch.framework"
};
```

示例代码

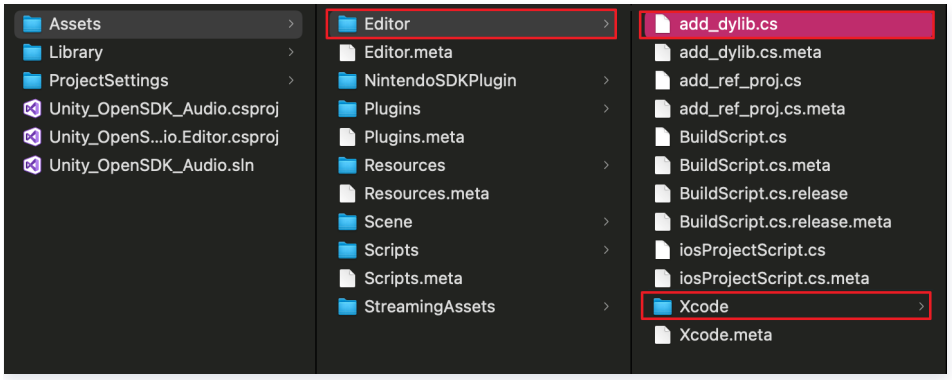
可参考 Demo 工程中的 add_dylib.cs 脚本文件，根据自己工程需求将此部分代码放在工程中 Editor 文件夹下。

```
[UnityEditor.Callbacks.PostProcessBuild(1002)]
public  static void OnPostprocessBuild (UnityEditor.BuildTarget BuildTarget, string path){
    if (BuildTarget == UnityEditor.BuildTarget.iOS) {
        UnityEngine.Debug.Log ("OnPostprocessBuild add_dylib:" + path);
#if UNITY_EDITOR_OSX || UNITY_STANDALONE_OSX
        {
            string projPath = UnityEditor.iOS.Xcode.PBXProject.GetPBXProjectPath (path);
            UnityEditor.iOS.Xcode.PBXProject proj = new UnityEditor.iOS.Xcode.PBXProject ();

            proj.ReadFromString (System.IO.File.ReadAllText (projPath));
            // string targetGuid = proj.TargetGuidByName
(UnityEditor.iOS.Xcode.PBXProject.GetUnityTargetName ()); // 2018
            string targetGuid = proj.GetUnityMainTargetGuid();  // 2019

            // 根据导入的 framework 进行删减
            string[] framework_names = {
                "libgme_fdkaac.framework",
                "libgme_lamemp3.framework",
                "libgme_ogg.framework",
                "libgme_soundtouch.framework"
            };

            for (int i = 0; i < framework_names.Length; i++)
            {
                string framework_name = framework_names[i];
```

```
                string dylibGuid = null;
                dylibGuid = proj.FindFileGuidByProjectPath("Frameworks/Plugins/iOS/" +
framework_name);

                if (dylibGuid == null) {
                    UnityEngine.Debug.LogWarning (framework_name + " guid not found");
                } else {
                    UnityEngine.Debug.LogWarning (framework_name + " guid:" + dylibGuid);
                    // proj.AddDynamicFramework (targetGuid, dylibGuid);

UnityEditor.iOS.Xcode.Extensions.PBXProjectExtensions.AddFileToEmbedFrameworks(proj, targetGuid,
dylibGuid);

                    proj.AddBuildProperty(targetGuid, "LD_RUNPATH_SEARCH_PATHS",
"@executable_path/Frameworks");
                    System.IO.File.WriteAllText (projPath, proj.WriteToString ());
                }
            }
        }
#endif
    }
}
```

2. 动态库处理（Unity 版本低于2019）

目前只有 Unity 2019 及以后版本可以使用 UnityEditor.iOS.Xcode.Extensions，如果是早期 Unity 版本，可以从高版本 Unity 导出 UnityEditor.iOS.Xcode 包给低版本 Unity 使用，或者直接参考附件 UnityEditorAV.iOS.XCode.zip 将此文件解压后放置于工程目录 Editor 文件夹下。

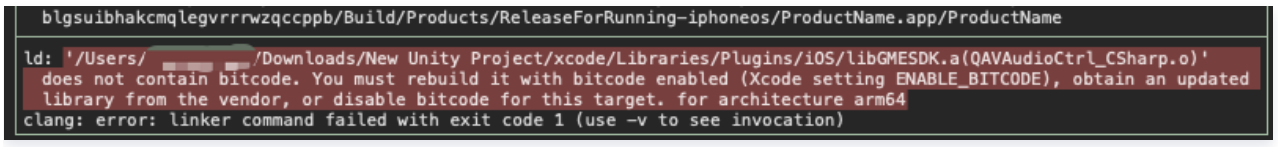


3. 导出 Xcode 工程

确保 Xcode 版本在10.0以上，在 Unity 编辑器中导出 Xcode 工程。

4. 关闭 BitCode

如果编译过程中提示下图错误，请关闭 Bitcode。关闭此设置，只需在 **Targets** > **Build Settings** 中搜索 Bitcode 即可，找到相应选项，设置为 NO。

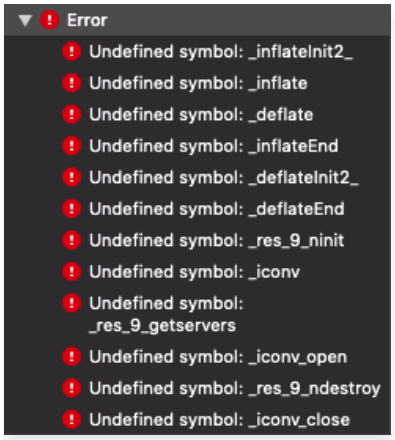


5. 添加 iOS 平台权限

- Required background modes：允许后台运行（可选）。
- Microphone Usage Description：允许麦克风权限。

6. 补充库文件

在编译过程中出现如下图错误，请将库文件补全。



库文件列表如下：

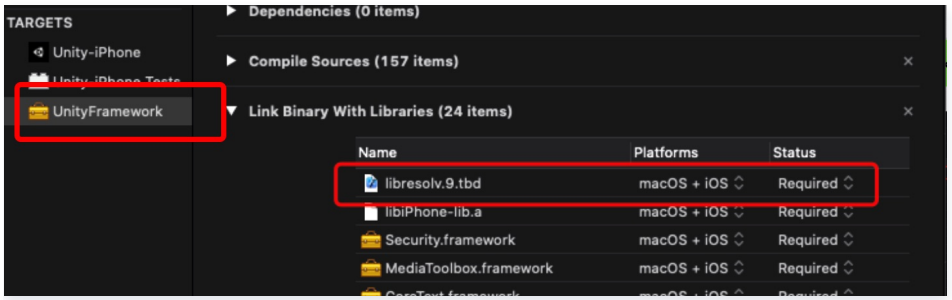
```
libc++.tbd
CoreMedia.framework
libresolv.tbd
AVFoundation.framework
Security.framework
CoreAudio.framework
AudioToolbox.framework
libiconv.tbd
libz.tbd
SystemConfiguration.framework
OpenAL.framework
```

7. 添加 libresolv.9.tbd

出现如下图错误：

```
Undefined symbols for architecture arm64:
  "_res_9_getservers", referenced from:
      tencent::av::getdnssvraddrs(tencent::av::xpstl::vector<res_9_sockaddr_union>&) in libGMESDK.a(xPlatform_open_avsdk.o)
      GMENetDiagnoseHelper::OutPutDNSServersiOSAndMAC(tencent::av::xpstl::vector<tencent::av::xp::strutf8>&) in
  libGMESDK.a(GMENetDiagnoseHelper.o)
  "_res_9_nclose", referenced from:
      GMENetDiagnoseHelper::OutPutDNSServersiOSAndMAC(tencent::av::xpstl::vector<tencent::av::xp::strutf8>&) in
  libGMESDK.a(GMENetDiagnoseHelper.o)
  "_res_9_ninit", referenced from:
      tencent::av::getdnssvraddrs(tencent::av::xpstl::vector<res_9_sockaddr_union>&) in libGMESDK.a(xPlatform_open_avsdk.o)
      GMENetDiagnoseHelper::OutPutDNSServersiOSAndMAC(tencent::av::xpstl::vector<tencent::av::xp::strutf8>&) in
  libGMESDK.a(GMENetDiagnoseHelper.o)
  "_res_9_ndestroy", referenced from:
      tencent::av::getdnssvraddrs(tencent::av::xpstl::vector<res_9_sockaddr_union>&) in libGMESDK.a(xPlatform_open_avsdk.o)
ld: symbol(s) not found for architecture arm64
clang: error: linker command failed with exit code 1 (use -v to see invocation)
```

请将 libresolv.9.tbd 添加到 **UnityFramework** 中。

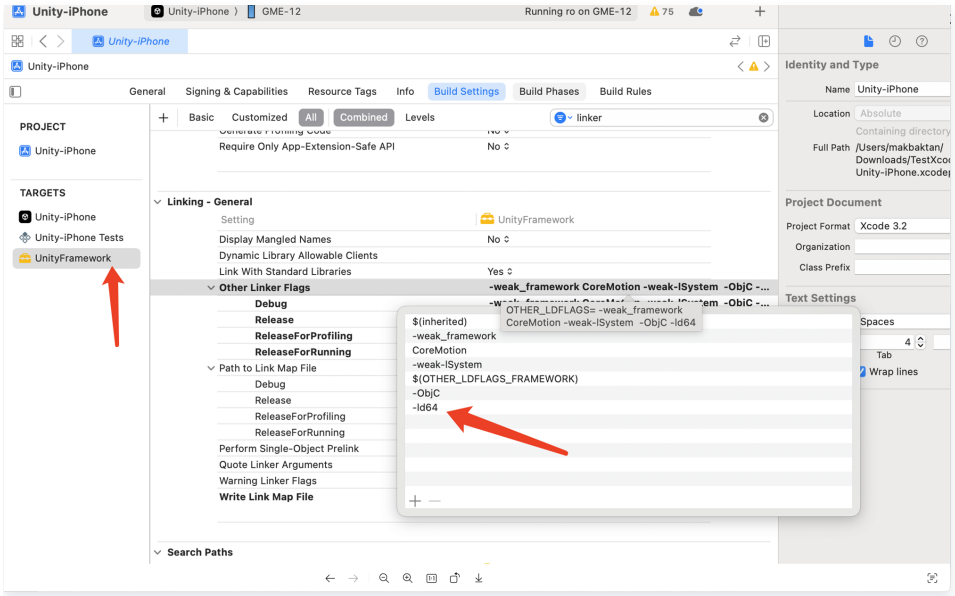


8. Xcode 15支持

如果在使用xcode 15编译Unity工程时出现:

```
error: unable to execute command: Abort trap: 6
clang: error: linker command failed due to signal (use -v to see invocation)
```

请添加 -ld64 连接选项



9. 导出问题

导出问题请参见 iOS 导出问题 进行解决。

导出 Android 平台

1. 删减 lib 文件

GME Unity SDK 默认提供 arm64-v8a、armeabi-v7a 及 x86 的 lib 文件，请根据工程项目所需进行删减。

> **架构缺失**
> 如果导出 Android 可执行文件，缺失指定架构，则会导致 Crash。

如果导出可执行文件 apk 文件后，打开后黑屏闪退，一般为缺少相应架构的 lib 文件，请根据工程进行添加或者删减。

2. 权限配置

2.1 必要权限

请务必在工程 AndroidManifest.xml 文件中添加以下权限：

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

2.2 根据需求添加权限

请根据需求在工程 AndroidManifest.xml 文件中添加以下权限：

读写权限

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。

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

蓝牙权限

需要根据以下规则判断如何添加蓝牙权限：

- 如果工程中的 targetSDKVersion 为 30 或者以下：

```
<uses-permission android:name="android.permission.BLUETOOTH"/>
```

- 如果工程中的 targetSDKVersion 为 31 或者以上：

```
<uses-permission android:name="android.permission.BLUETOOTH" android:maxSdkVersion="30"
/>
<uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />
```

3. 导出问题

导出问题请参见 Android 导出问题 进行解决。

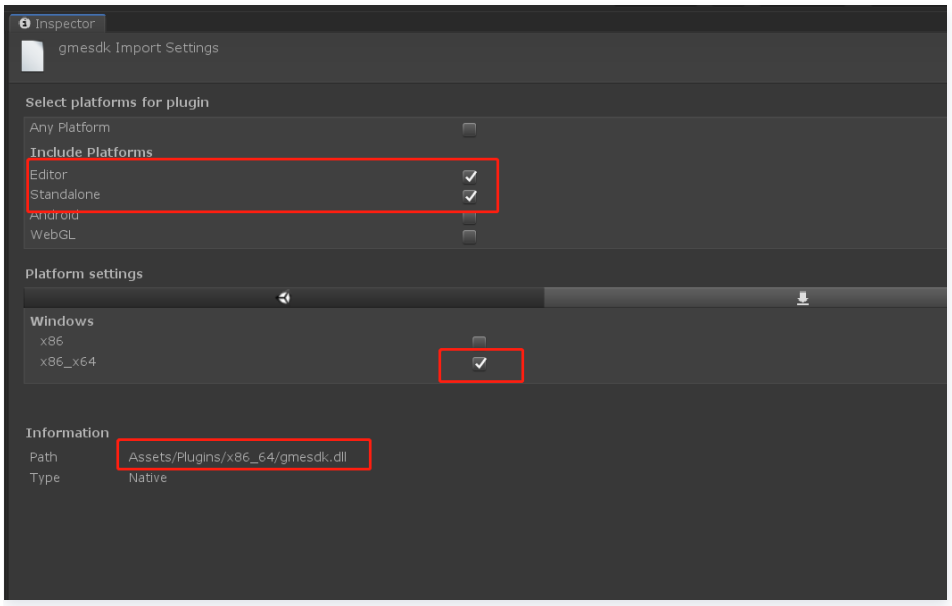
导出 Windows 平台

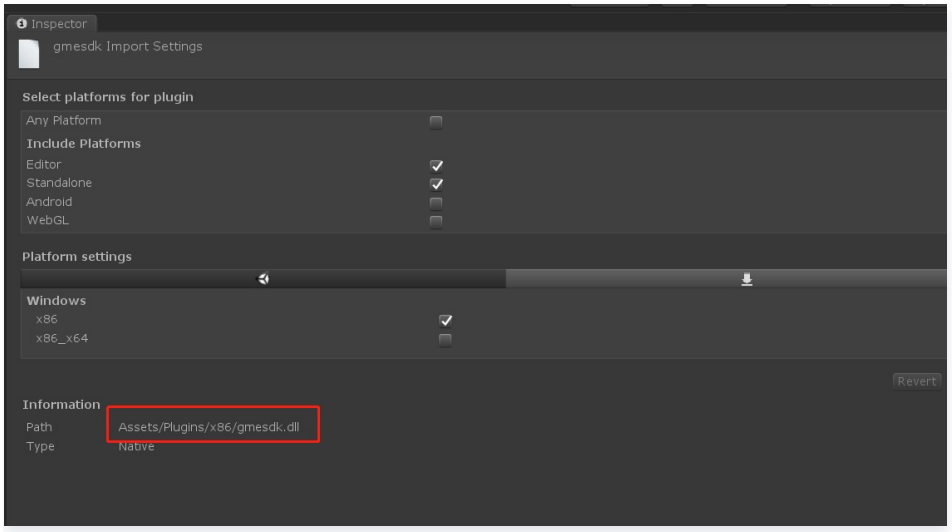
导出问题请参见 Windows 导出问题 进行解决。

导出 WebGL 平台

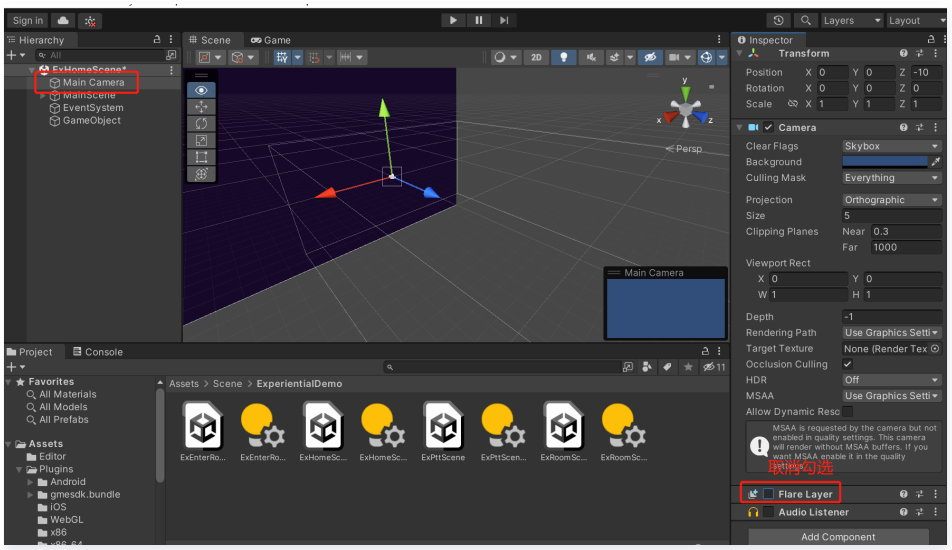
1. 配置WebGL下的plugins

设置 Windows 平台下 gmesdk.dll 的适用范围，避免它和 WebGL平台下的 gmesdk 冲突：

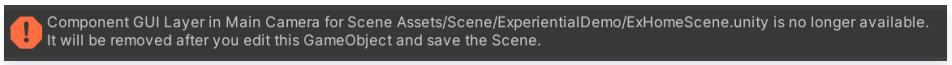




2. 取消Flare Layer（Unity 2018及以上版本）

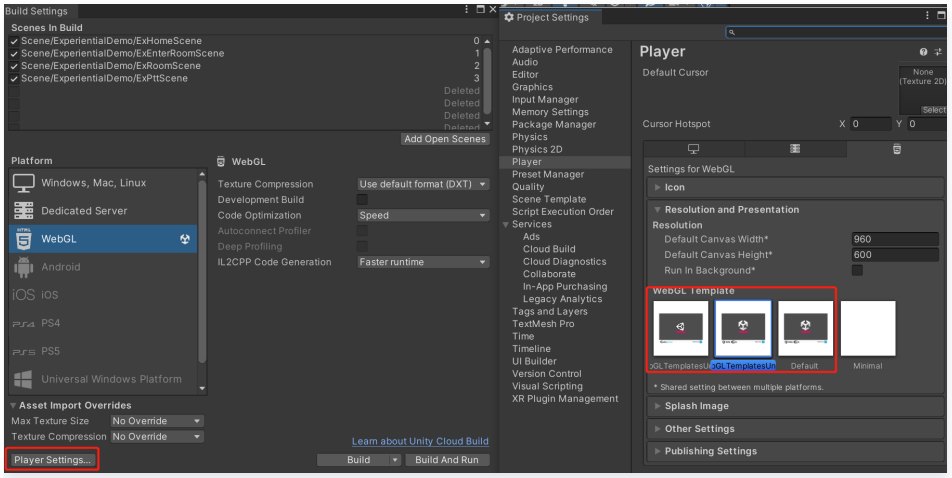


由于部分 Unity 版本不再支持 MainCamera 中的 Flare Layer 模式，因此需要在要打包的 Scene 中将 Flare Layer 取消勾选，否则会报以下错误：



3. 选择模板

导出 WebGL 平台时，选择 GME 的 WebGL 模板，这样才能保证打包成功的产物正确地导入了相关依赖库。工程默认会使用 GMEWebGLTemplatesUnity2018 模板，这个模板支持 Unity2018 和 Unity2019 版本。对于 Unity2020 和 Unity2021 版本，打包时需更改使用模板，使用 GMEWebGLTemplatesUnity2021 进行打包。



4. 引入前端库

在将 GME-WebGL 导入到自己的工程中，在使用 Unity 生成好对应的网页时，需要手动引入前端库并将前端库文件放在相应的引用位置，并添加 Audio 标签（如下图所示）。如希望每次打包 Unity 产物的时候就自动完成上述工作，可以参考 GME-WebGL demo中的做法，为自己的工程添加相应的模板。

5. 导出问题

导出问题请参见 Unity-WebGL平台导出问题 进行解决。

Unreal Engine SDK
集成 SDK

最近更新时间：2023-07-05 10:50:11

简介

为方便 Unreal Engine 开发者调试和接入腾讯云游戏多媒体引擎产品 API，这里向您介绍适用于 Unreal Engine 开发的工程配置。

下载 SDK

1. 请先下载相关 Demo 及 SDK。详细下载链接请查看 SDK 下载指引 。
2. 在界面中找到 Unreal 版本的 SDK 资源。
3. 单击**下载**。下载完的 SDK 资源解压后有以下几个部分。文件说明如下表：

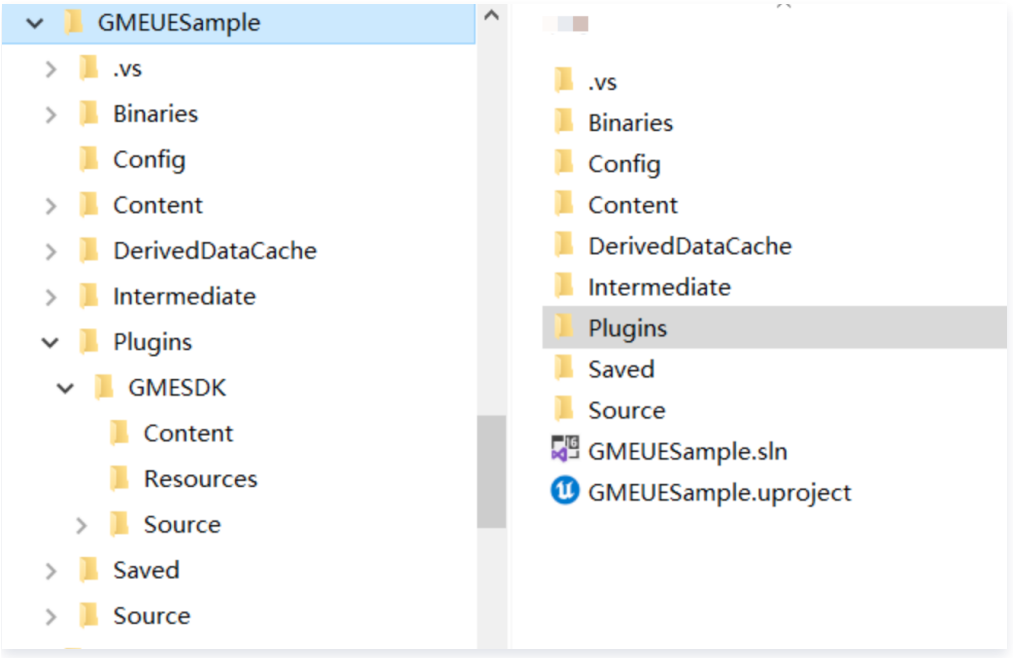
文件名	说明	作用
GMESDK.uplugin	uplugin文件	插件配置文件
Resources	插件资源文件	插件资源文件
Source	SDK 文件	SDK 各平台库文件以及代码文件（例如头文件）

> **Unreal SDK 平台支持：**
> Unreal SDK 已同时集成 Windows、Mac、Android、iOS 平台架构。如需主机平台架构，请 联系我们 。

工程配置步骤

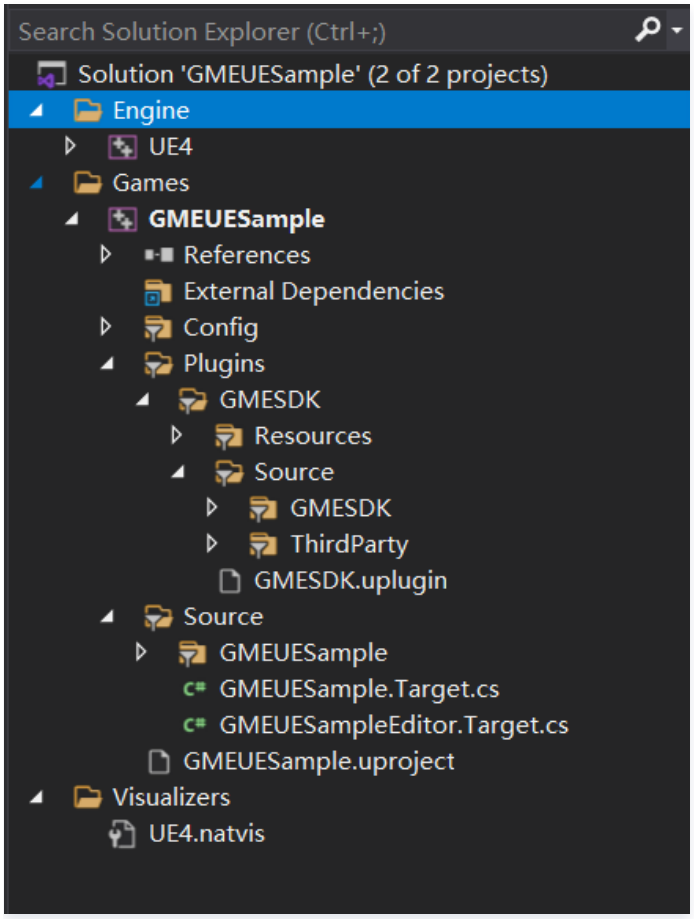
步骤1：导入 Plugins 文件

游戏工程的根目录（xxxx.uproject文件所在目录）下如果不存在 Plugins 文件夹，先新建一个，然后将 GME SDK 拷入 Plugins 文件夹，完成该步骤后的目录结构如下



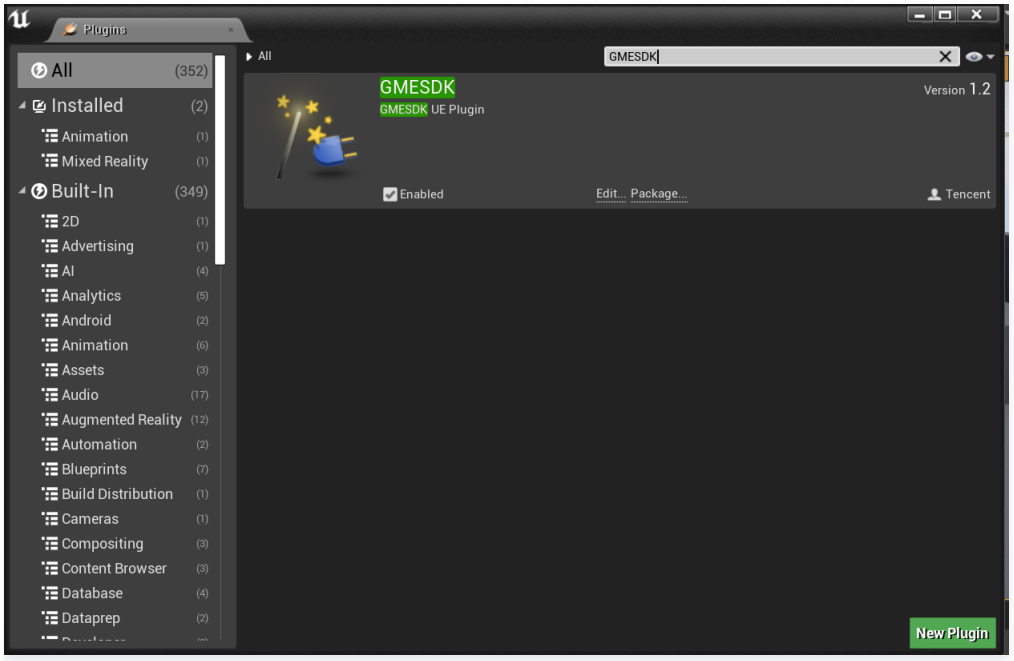
步骤2：编译插件

刷新 C++工程（Visual studio/Xcode），打开 C++工程，可以看到类似下图的目录结构，然后编译游戏。



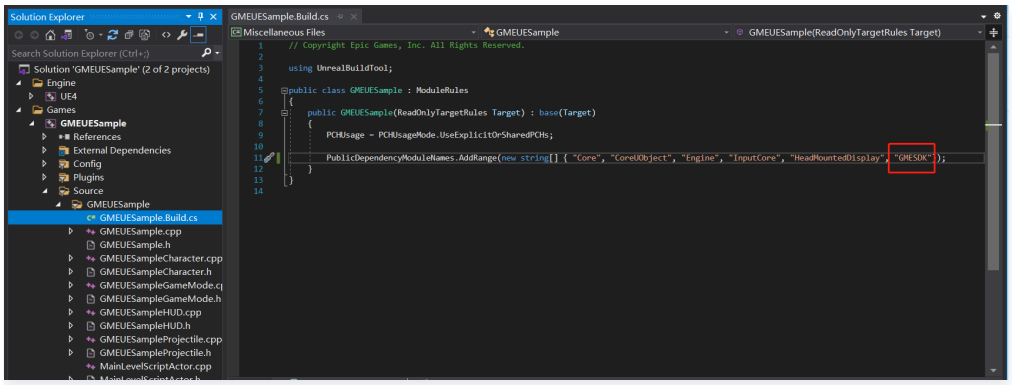
步骤3：编译完成

编译完成后重启 UE Editor，在 UE Editor 中打开插件管理器，可以看到 GME SDK 已经引入工程了，确定 GME SDK 是 Enabled 状态。



步骤4：新增 GME SDK 依赖

在游戏工程的 .build.cs 文件中新增 GME SDK 的依赖。



不同 Unreal 版本适配工作

Unreal Engine 4.21及以上版本

如果是使用 Unreal Engine 4.21及以上版本，下载使用 GME Unreal Engine 示例代码后要添加以下代码：

```
AUEDemoLevelScriptActor::AUEDemoLevelScriptActor()
{
    PrimaryActorTick.bCanEverTick = true;
}
```

> **说明**
> 在默认的情况下，tick 为关闭状态，必须手动开启。

实时语音

最近更新时间：2024-07-25 15:11:31

为方便 Unreal Engine 开发者调试和接入腾讯云游戏多媒体引擎客户端 API，本文为您介绍适用于 Unreal Engine 实时语音功能的开发接入技术文档。

使用 GME 重要事项

GME 提供实时语音服务、语音消息服务及转文本服务，使用 GME 服务都依赖 Init 和 Poll 等核心接口。

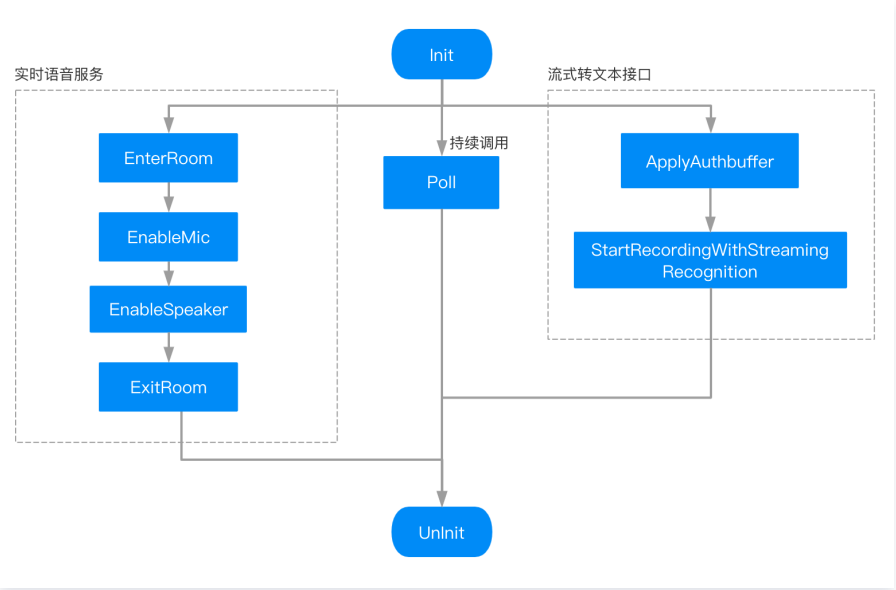
重点提示

- 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、语音消息服务以及转文本服务**。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。
- GME 的接口调用成功后返回值为 QAVError.OK，数值为 0。
- GME 的接口调用要在同一个线程下。
- GME 需要周期性的调用 Poll 接口触发事件回调。
- 错误码详情可参见 错误码 。

接入 SDK

重要步骤

接入 SDK 重要流程如下：



1. 初始化 GME
2. 周期性调用 Poll 触发回调
3. 进入实时语音房间
4. 打开麦克风
5. 打开扬声器
6. 退出语音房间
7. 反初始化 GME

C++ 类

类	含义
ITMGContext	核心接口
ITMGRoom	房间相关接口
ITMGRoomManager	房间管理接口
ITMGAudioCtrl	音频相关接口
ITMGAudioEffectCtrl	音效及伴奏相关接口

核心接口

接口	接口含义
Init	初始化 GME
Poll	触发事件回调
Pause	系统暂停
Resume	系统恢复
Uninit	反初始化 GME

引用头文件

接入 GME 首先需要引入头文件 tmg_sdk.h，头文件类继承 ITMGDelegate 以进行消息的传递及回调。

示例代码

```
#include "GMESDK/tmg_sdk.h"

class UEDEMO1_API AUEDemoLevelScriptActor : public ALevelScriptActor, public ITMGDelegate
{
public:
...
private:
...
}
```

设置单例

在 EnterRoom 函数调用之前要先获取 ITMGContext ，所有调用都从 ITMGContext 开始，由 ITMGDelegate 回调回传给应用，必须首先设置。

示例代码

```
ITMGContext* context = ITMGContextGetInstance();
context->SetTMGDelegate(this);
```

消息传递

接口类采用 Delegate 方法用于向应用程序发送回调通知，消息类型参考 ITMG_MAIN_EVENT_TYPE，data 在 Windows 平台下是 json 字符串格式， 具体 key-value 参见说明文档。

示例代码

```
//函数实现：
//UEDemoLevelScriptActor.h:
class UEDEMO1_API AUEDemoLevelScriptActor : public ALevelScriptActor, public SetTMGDelegate
{
public:
    void OnEvent(ITMG_MAIN_EVENT_TYPE eventType, const char* data);
}

//UEDemoLevelScriptActor.cpp:
void AUEDemoLevelScriptActor::OnEvent(ITMG_MAIN_EVENT_TYPE eventType, const char* data){
    //在此对 eventType 进行判断及操作
}
```

初始化 SDK

未初始化前，SDK 处于未初始化阶段，需要通过接口 Init 初始化 SDK，才可以使用实时语音服务、语音消息服务及转文本服务。调用 Init 接口的线程必须于其他接口在同一线程,建议都在主线程调用接口。

接口原型

```
ITMGContext virtual int Init(const char* sdkAppId, const char* openId)
```

参数	类型	含义
sdkAppId	const char*	来自 腾讯云控制台 的 GME 服务提供的 AppID，获取请参考 语音服务开通指引 。
openID	const char*	openID 只支持 Int64 类型（转为 const char* 传入）。规则由 App 开发者自行制定，App 内不重复即可。如需使用字符串作为 Openid 传入，可 提交工单 联系开发者。

返回值

返回值	处理
AV_OK = 0	初始化 SDK 成功
AV_ERR_SDK_NOT_FULL_UPDATE=7015	检查 SDK 文件是否完整，建议删除后重新导入 SDK

> ⚠ **关于7015错误提示**
>
> - 7015错误码是通过 md5 进行判断，在接入过程中若出现此错误，请根据提示检查 SDK 文件是否完整、SDK 文件版本是否一致。
> - 出现返回值 AV_ERR_SDK_NOT_FULL_UPDATE 时，此返回值**只有提示作用**，并不会造成初始化失败。
> - 由于第三方加固、Unity 打包机制等因素会影响库文件 md5，造成误判，所以**正式发布请在逻辑中忽略此错误**，并尽量不在 UI 中提示。

示例代码

```
std::string appid = TCHAR_TO_UTF8(CurrentWidget->editAppID->GetText().ToString().operator*());
std::string userId = TCHAR_TO_UTF8(CurrentWidget->editUserID->GetText().ToString().operator*());
ITMGContextGetInstance()->Init(appid.c_str(), userId.c_str());
```

触发事件回调

通过在 update 里面周期的调用 Poll 可以触发事件回调。Poll 是 GME 的消息泵，GME 需要周期性的调用 Poll 接口触发事件回调。如果没有调用 Poll ，将会导致整个 SDK 服务运行异常。
详情请参见 Demo 中的 UEDemoLevelScriptActor.cpp 文件。

> **务必周期性调用 Poll 接口**
> 务必周期性调用 Poll 接口且在主线程调用，以免接口回调异常。

接口原型

```
class ITMGContext {
protected:
    virtual ~ITMGContext() {}

public:
    virtual void Poll()= 0;
}
```

示例代码

```
//头文件中的声明
virtual void Tick(float DeltaSeconds);

void AUEDemoLevelScriptActor::Tick(float DeltaSeconds) {
    Super::Tick(DeltaSeconds);
    ITMGContextGetInstance()->Poll();
}
```

系统暂停

当系统发生 Pause 事件时，需要同时通知引擎进行 Pause。如果不需要后台播放房间内声音，请调用 Pause 接口暂停整个 GME 服务。

接口原型

```
ITMGContext int Pause()
```

系统恢复

当系统发生 Resume 事件时，需要同时通知引擎进行 Resume。Resume 接口只恢复实时语音。

接口原型

```
ITMGContext int Resume()
```

反初始化 SDK

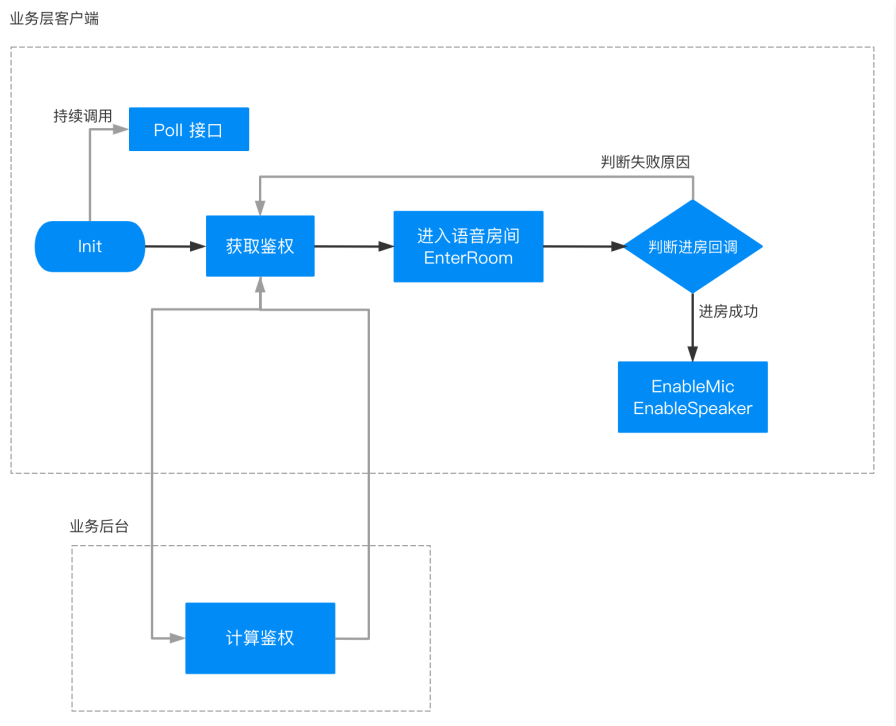
反初始化 SDK，进入未初始化状态。**如果游戏业务侧账号与 openid 是绑定的，那切换游戏账号需要反初始化 GME，再用新的 openid 初始化。**

接口原型

```
ITMGContext int Uninit()
```

实时语音房间相关接口

初始化之后，SDK 调用进房后进去了房间，才可以进行实时语音通话。
使用问题可参考 实时语音相关问题 。



接口	接口含义
GenAuthBuffer	本地鉴权计算
EnterRoom	加入房间
ExitRoom	退出房间
IsRoomEntered	判断是否已经进入房间
SwitchRoom	快速切换房间

本地鉴权计算

生成 AuthBuffer，用于相关功能的加密和鉴权，如正式发布请使用后台部署密钥，后台部署请参考 鉴权密钥 。

接口原型

```
int  QAVSDK_AuthBuffer_GenAuthBuffer(unsigned int dwSdkAppID, const char* strRoomID, const char*
strOpenID,
    const char* strKey, unsigned char* strAuthBuffer, unsigned int bufferLength);
```

参数	类型	含义
dwSdkAppID	unsigned int	来自腾讯云控制台的 AppId 号码。

strRoomID	const char*	房间号，最大支持127字符。
strOpenID	const char*	用户标识。与 Init 时候的 openID 相同。
strKey	const char*	来自腾讯云 控制台 的权限密钥。
strAuthBuffer	const char*	返回的 authbuff。
bufferLength	Int	传入的 authbuff 长度，建议为 500。

示例代码

```
unsigned int bufferLen = 512;
unsigned char retAuthBuff[512] = {0};
QAVSDK_AuthBuffer_GenAuthBuffer(atoi(SDKAPPID3RD), roomId, "10001",
AUTHKEY,retAuthBuff,bufferLen);
```

加入房间

用生成的鉴权信息进房，加入房间默认不打开麦克风及扬声器。

> **注意**
> - 加入房间事件回调结果 result 为 0 代表进房成功，进房接口 EnterRoom 返回值为 0 不代表进房成功。
> - 房间的音频类型由第一个进房的人确定，此后房间里有成员修改房间类型，将对此房间所有成员生效。例如第一个进入房间的人使用的房间音频类型是流畅音质，第二个进房的是即使进房时候调用接口的音频类型参数是高清音质，进入房间之后也会变成流畅音质。需要有成员调用 ChangeRoomType 才会修改房间的音频类型。

接口原型

```
ITMGContext virtual int EnterRoom(const char*  roomID, ITMG_ROOM_TYPE roomType, const char*
authBuff, int buffLen)
```

参数	类型	含义
roomID	const char*	房间号，最大支持127字符。
roomType	ITMG_ROOM_TYPE	房间类型，游戏建议使用 ITMG_ROOM_TYPE_FLUENCY。房间音频类型请参考 音质选择。
authBuffer	const char*	鉴权码。
buffLen	Int	鉴权码长度。

示例代码

```
ITMGContext* context = ITMGContextGetInstance();
context->EnterRoom(roomID, ITMG_ROOM_TYPE_FLUENCY, (char*)retAuthBuff,bufferLen);
```

加入房间事件回调

加入房间完成后会发送信息 ITMG_MAIN_EVENT_TYPE_ENTER_ROOM，在 OnEvent 函数中进行判断回调后处理。如果回调为成功，即此时进房成功，开始进行计费。

计费问题参考：

- 购买指南。
- 计费相关问题。
- 使用实时语音后，如果客户端掉线了，是否还会继续计费？

示例代码

```
void UBaseViewController::OnEvent(ITMG_MAIN_EVENT_TYPE eventType, const char *data) {

    FString jsonData = FString(UTF8_TO_TCHAR(data));
    TSharedPtr<FJsonObject> JsonObject;
    TSharedRef<TJsonReader<>> Reader =
TJsonReaderFactory<>::Create(FString(UTF8_TO_TCHAR(data)));
    FJsonSerializer::Deserialize(Reader, JsonObject);


    if (eventType == ITMG_MAIN_EVENT_TYPE_ENTER_ROOM) {
        int32 result = JsonObject->GetIntegerField(TEXT("result"));
        FString error_info = JsonObject->GetStringField(TEXT("error_info"));
        if (result == 0) {
            GEngine->AddOnScreenDebugMessage(INDEX_NONE, 20.0f, FColor::Yellow, TEXT("Enter room
success."));
        }
        else {
            FString msg = FString::Printf(TEXT("Enter room failed. result=%d, info = %ls"),
result, *error_info);
            GEngine->AddOnScreenDebugMessage(INDEX_NONE, 20.0f, FColor::Yellow, *msg);
        }
        onEnterRoomCompleted(result, error_info);
    }
}
```

Data 详情

消息	Data	例子
ITMG_MAIN_EVENT_TYPE_ENTER_ROOM	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_ROOM_DISCONNECT	result; error_info	{"error_info":"waiting timeout, please check your network","result":0}

如果断网，将会有断网的回调提示 `ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVENT_TYPE_ROOM_DISCONNECT` **，此时 SDK 会自动进行重连，回调是** `ITMG_MAIN_EVENT_TYPE_RECONNECT_START` **，当重连成功时，会有** `ITMG_MAIN_EVENT_TYPE_RECONNECT_SUCCESS` **回调。**

错误码

错误码值	原因及建议方案
7006	鉴权失败原因。 • AppID 不存在或者错误 • authbuff 鉴权错误 • 鉴权过期

	• OpenId 不符合规范
7007	已经在其它房间。
1001	已经在进房过程中，然后又重复了此操作。建议在进房回调返回之前不要再调用进房接口。
1003	已经进房了在房间中，又调用一次进房接口。
1101	确保已经初始化 SDK，确保 OpenId 是否符合规则，或者确保在同一线程调用接口，以及确保 Poll 接口正常调用。

退出房间

通过调用此接口可以退出所在房间。这是一个异步接口，返回值为 AV_OK 的时候代表异步投递成功。如果应用中有退房后立即进房的场景，在接口调用流程上，开发者无需要等待 ExitRoom 的回调 RoomExitComplete 通知，只需直接调用接口。

接口原型

```
ITMGContext virtual int ExitRoom()
```

示例代码

```
ITMGContext* context = ITMGContextGetInstance();
context->ExitRoom();
```

退出房间事件回调

退出房间完成后会有回调，消息为 ITMG_MAIN_EVENT_TYPE_EXIT_ROOM。

示例代码

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

Data详情

消息	Data	例子
ITMG_MAIN_EVENT_TYPE_EXIT_ROOM	result; error_info	{"error_info":"","result":0}

判断是否已经进入房间

通过调用此接口可以判断是否已经进入房间，返回值为 bool 类型。请勿在进房过程中调用。

接口原型

```
ITMGContext virtual bool IsRoomEntered()
```

示例代码

```
ITMGContext* context = ITMGContextGetInstance();
context->IsRoomEntered();
```

快速切换房间

调用此接口快速切换实时语音房间。此接口在进房后调用。切换房间后，不重置设备，即如果在此房间已经是打开麦克风状态，在切换房间后也会是打开麦克风状态。
快速切换房间的回调是 ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVENT_TYPE_SWITCH_ROOM，字段是 error_info 以及 result。

接口原型

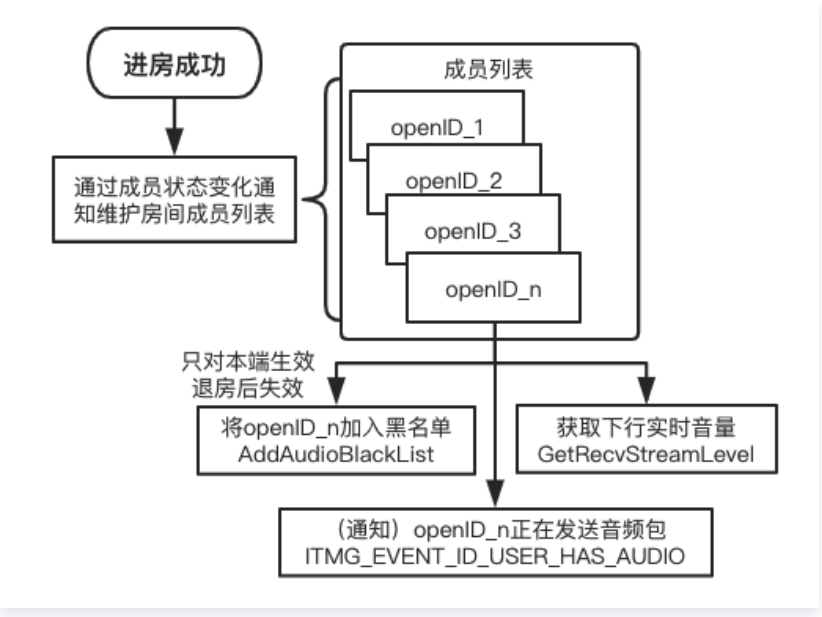
```
ITMGContext virtual int SwitchRoom(const char* targetRoomID, const char* authBuff, int buffLen);
```

类型说明

参数	类型	含义
targetRoomID	const char*	将要进入的房间号。
authBuffer	const char*	用将要进入的房间号生成的新鉴权。
buffLen	Int	鉴权码长度。

房间内状态维护

此部分接口用于业务层显示说话成员、进退房成员，以及将房间内某成员禁言等功能。



接口/通知	含义
ITMG_MAIN_EVNET_TYPE_USER_UPDATE	成员状态变化通知。
AddAudioBlackList	房间中禁言某成员。
RemoveAudioBlackList	移除禁言。

成员进房、开关麦克风、说话状态通知事件

此接口适用于获取房间中说话的人并在 UI 中展示，以及有人进入、退出语音房间的一个通知。
该事件在状态变化才通知，状态不变化的情况下不通知。如需实时获取成员状态，请在上层收到通知时缓存，事件消息为 ITMG_MAIN_EVNET_TYPE_USER_UPDATE，其中 data 包含两个信息，event_id 及 user_list，在 OnEvent 函数中对事件消息进行判断。
音频事件的通知有一个阈值，超过这个阈值才会发送通知。超过两秒没有收到音频包才通知“有成员停止发送音频包”消息。

event_id	含义	应用侧维护内容
ITMG_EVENT_ID_USER_ENTER	有成员进入房间，返回此时进房的 openid	应用侧维护成员列表
ITMG_EVENT_ID_USER_EXIT	有成员退出房间，返回此时退房的 openid	应用侧维护成员列表
ITMG_EVENT_ID_USER_HAS_AUDIO	有成员发送音频包，返回此时房间内说话的 openid，通过此事件可以判断用户是否说话，并展示声纹效果	应用侧维护通话成员列表
ITMG_EVENT_ID_USER_NO_AUDIO	有成员停止发送音频包，返回此时房间内停止说话的 openid	应用侧维护通话成员列表
ITMG_EVENT_ID_USER_MIC_OPENED	有成员打开麦克风设备（2.9.10及以后版本支持）	应用侧维护通话成员列表
ITME_EVENT_ID_USER_MIC_CLOSED	有成员关闭麦克风设备（2.9.10及以后版本支持）	应用侧维护通话成员列表

示例代码

```
void TMGTestScene::OnEvent(ITMG_MAIN_EVENT_TYPE eventType,const char* data){
    switch (eventType) {
            case ITMG_MAIN_EVNET_TYPE_USER_UPDATE:
        {
        //进行处理
        //开发者对参数进行解析，得到信息 eventID 及 user_list
            switch (eventID)
             {
             case ITMG_EVENT_ID_USER_ENTER:
                  //有成员进入房间
                 break;
             case ITMG_EVENT_ID_USER_EXIT:
                  //有成员退出房间
                break;
            case ITMG_EVENT_ID_USER_HAS_AUDIO:
                //有成员发送音频包
                break;
            case ITMG_EVENT_ID_USER_NO_AUDIO:
                //有成员停止发送音频包
                break;
            case ITMG_EVENT_ID_USER_MIC_OPENED:
                //有成员开启麦克风设备
                break;
            case ITME_EVENT_ID_USER_MIC_CLOSED:
                //有成员关闭麦克风设备
                break;
             default:
                break;
```

```
            }
        break;
        }
    }
}
```

房间中禁言某成员

将某个 ID 加入音频数据黑名单，即不接受某人的语音， 只对本端生效，不会影响其他端。返回值为 0 表示调用成功。例如： A、B、C 都在同一个房间开麦说话：

- 如果 A 设置了 C 的黑名单， 则 A 只能听见 B 的声音。
- B 因为没有设置黑名单， 仍旧可以听见 A 和 C 的声音。
- C 同样因为没有设置黑名单， 可以听见 A 和 B 的声音。

此接口适用于在语音房间中将某用户禁言的场景。

接口原型

```
ITMGContext ITMGAudioCtrl int AddAudioBlackList(const char* openId)
```

参数	类型	含义
openId	char*	需添加黑名单的用户 openid

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->AddAudioBlackList(openId);
```

移除禁言

将某个 Id 移除音频数据黑名单。返回值为0表示调用成功。

接口原型

```
ITMGContext ITMGAudioCtrl int RemoveAudioBlackList(const char* openId)
```

参数	类型	含义
openId	char*	需移除黑名单的 ID

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->RemoveAudioBlackList(openId);
```

实时语音采集相关接口

- 初始化 SDK 之后进房，在房间中，才可以调用实时音频语音相关接口。
- 当用户界面单击打开/关闭麦克风/扬声器按钮时，建议采用 EnableMic 以及 EnableSpeaker 接口进行调用。
- 当用户界面按住麦克风按钮时发言，放开按钮不发言，建议采用进房时候调用 EnableAudioCaptureDevice 一次，后续按住发言调用 EnableAudioSend 来实现。

接口	接口含义

EnableMic	开关麦克风
GetMicState	获取麦克风状态
EnableAudioCaptureDevice	开关采集设备
IsAudioCaptureDeviceEnabled	获取采集设备状态
EnableAudioSend	打开关闭音频上行
IsAudioSendEnabled	获取音频上行状态
GetMicLevel	获取实时麦克风音量
GetSendStreamLevel	获取音频上行实时音量
SetMicVolume	设置麦克风音量
GetMicVolume	获取麦克风音量

开启或关闭麦克风

此接口用来开启关闭麦克风。加入房间默认不打开麦克风及扬声器。**EnableMic = EnableAudioCaptureDevice + EnableAudioSend**

接口原型

```
ITMGAudioCtrl virtual int EnableMic(bool bEnabled)
```

参数	类型	含义
bEnabled	Bool	如果需要打开麦克风，则传入的参数为 true，如果关闭麦克风，则参数为 false

示例代码

```
void UBaseViewController::OnEvent(ITMG_MAIN_EVENT_TYPE eventType, const char *data) {

    FString jsonData = FString(UTF8_TO_TCHAR(data));
    TSharedPtr<FJsonObject> JsonObject;
    TSharedRef<TJsonReader<>> Reader =
TJsonReaderFactory<>::Create(FString(UTF8_TO_TCHAR(data)));
    FJsonSerializer::Deserialize(Reader, JsonObject);


    if (eventType == ITMG_MAIN_EVENT_TYPE_ENTER_ROOM) {
        int32 result = JsonObject->GetIntegerField(TEXT("result"));
        FString error_info = JsonObject->GetStringField(TEXT("error_info"));
        if (result == 0) {
            GEngine->AddOnScreenDebugMessage(INDEX_NONE, 20.0f, FColor::Yellow, TEXT("Enter room
success."));
            //打开麦克风
            ITMGContextGetInstance()->GetAudioCtrl()->EnableMic(true);
        }
        else {
            FString msg = FString::Printf(TEXT("Enter room failed. result=%d, info = %ls"),
result, *error_info);
```

```
            GEngine->AddOnScreenDebugMessage(INDEX_NONE, 20.0f, FColor::Yellow, *msg);
        }
        onEnterRoomCompleted(result, error_info);
    }
}
```

麦克风状态获取

此接口用于获取麦克风状态，返回值0为关闭麦克风状态，返回值1为打开麦克风状态。

接口原型

```
ITMGAudioCtrl virtual int GetMicState()
```

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->GetMicState();
```

开启或关闭采集设备

此接口用来开启/关闭采集设备。加入房间默认不打开设备。

- 只能在进房后调用此接口，退房会自动关闭设备。
- 在移动端，打开采集设备通常会伴随权限申请，音量类型调整等操作。

接口原型

```
ITMGAudioCtrl virtual int EnableAudioCaptureDevice(bool enable)
```

参数	类型	含义
enable	Bool	如果需要打开采集设备，则传入的参数为 true，如果关闭采集设备，则参数为 false

示例代码

```
//打开采集设备
ITMGContextGetInstance()->GetAudioCtrl()->EnableAudioCaptureDevice(true);
```

采集设备状态获取

此接口用于采集设备状态获取。

接口原型

```
ITMGContext virtual bool IsAudioCaptureDeviceEnabled()
```

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->IsAudioCaptureDeviceEnabled();
```

打开或关闭音频上行

此接口用于打开/关闭音频上行。如果采集设备已经打开，那么会发送采集到的音频数据。如果采集设备没有打开，那么仍旧无声。采集设备的打开关闭请参见接口 EnableAudioCaptureDevice。

接口原型

```
ITMGContext  virtual int EnableAudioSend(bool bEnable)
```

参数	类型	含义
bEnable	Bool	如果需要打开音频上行，则传入的参数为 true，如果关闭音频上行，则参数为 false

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->EnableAudioSend(true);
```

音频上行状态获取

此接口用于音频上行状态获取。

接口原型

```
ITMGContext virtual bool IsAudioSendEnabled()
```

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->IsAudioSendEnabled();
```

获取麦克风实时音量

此接口用于获取麦克风实时音量，返回值为 int 类型。建议20ms获取一次。值域为0 – 100，通过此接口可以获取到麦克风采集到的实时音量情况。

接口原型

```
ITMGAudioCtrl virtual int GetMicLevel()
```

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->GetMicLevel();
```

获取音频上行实时音量

此接口用于获取自己音频上行实时音量，返回值为 int 类型，取值范围为0 – 100。

接口原型

```
ITMGAudioCtrl virtual int GetSendStreamLevel()
```

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->GetSendStreamLevel();
```

版权所有：腾讯云计算（北京）有限责任公司

设置麦克风软件音量

此接口用于设置麦克风的音量。参数 volume 用于设置麦克风的音量，相当于对采集的声音做衰减或增益。

接口原型

```
ITMGAudioCtrl virtual int SetMicVolume(int vol)
```

参数	类型	含义
vol	Int	取值范围为 0-200，数值为0的时候表示静音，当数值为100的时候表示音量不增不减，默认数值为100。

示例代码

```
int micVol = (int)(value * 100);
ITMGContextGetInstance()->GetAudioCtrl()->SetMicVolume(vol);
```

获取麦克风软件音量

此接口用于获取麦克风的音量。返回值为一个int类型数值，返回值为101代表没调用过接口 SetMicVolume。

接口原型

```
ITMGAudioCtrl virtual int GetMicVolume()
```

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->GetMicVolume();
```

实时语音播放相关接口

接口	接口含义
EnableSpeaker	开关扬声器
GetSpeakerState	获取扬声器状态
EnableAudioPlayDevice	开关播放设备
IsAudioPlayDeviceEnabled	获取播放设备状态
EnableAudioRecv	打开关闭音频下行
IsAudioRecvEnabled	获取音频下行状态
GetSpeakerLevel	获取实时扬声器音量
GetRecvStreamLevel	获取房间内其他成员下行实时音量
SetSpeakerVolume	设置扬声器音量
GetSpeakerVolume	获取扬声器音量
TrackingVolume	开始监听房间内用户的音量变化
StopTrackingVolume	停止监听房间内用户的音量变化

版权所有：腾讯云计算（北京）有限责任公司

开启或关闭扬声器

此接口用于开启关闭扬声器。EnableSpeaker = EnableAudioPlayDevice + EnableAudioRecv

接口原型

```
ITMGAudioCtrl virtual int EnableSpeaker(bool enable)
```

参数	类型	含义
enable	Bool	如果需要关闭扬声器，则传入的参数为 false，如果打开扬声器，则参数为 true

示例代码

```
//打开扬声器
ITMGContextGetInstance()->GetAudioCtrl()->EnableSpeaker(true);
```

扬声器状态获取

此接口用于扬声器状态获取。返回值0为关闭扬声器状态，返回值1为打开扬声器状态。

接口原型

```
ITMGAudioCtrl virtual int GetSpeakerState()
```

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->GetSpeakerState();
```

开启或关闭播放设备

此接口用于开启关闭播放设备。

接口原型

```
ITMGAudioCtrl virtual int EnableAudioPlayDevice(bool enable)
```

参数	类型	含义
enable	Bool	如果需要关闭播放设备，则传入的参数为 false，如果打开播放设备，则参数为 true

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->EnableAudioPlayDevice(true);
```

播放设备状态获取

此接口用于播放设备状态获取。

接口原型

```
ITMGAudioCtrl virtual bool IsAudioPlayDeviceEnabled()
```

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->IsAudioPlayDeviceEnabled();
```

打开或关闭音频下行

此接口用于打开/关闭音频下行。如果播放设备已经打开，那么会播放房间里其他人的音频数据。如果播放设备没有打开，那么仍旧无声。播放设备的打开关闭参见接口请参见 EnableAudioPlayDevice。

接口原型

```
ITMGAudioCtrl virtual int EnableAudioRecv(bool enable)
```

参数	类型	含义
enable	Bool	如果需要打开音频下行，则传入的参数为 true，如果关闭音频下行，则参数为 false

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->EnableAudioRecv(true);
```

音频下行状态获取

此接口用于音频下行状态获取。

接口原型

```
ITMGAudioCtrl virtual bool IsAudioRecvEnabled()
```

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->IsAudioRecvEnabled();
```

获取扬声器实时音量

此接口用于获取扬声器实时音量。返回值为 int 类型数值，表示扬声器实时音量。建议20ms获取一次。

接口原型

```
ITMGAudioCtrl virtual int GetSpeakerLevel()
```

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->GetSpeakerLevel();
```

获取房间内其他成员下行实时音量

此接口用于获取房间内其他成员下行实时音量，返回值为 int 类型，取值范围为0 - 200。

接口原型

```
ITMGAudioCtrl virtual int GetRecvStreamLevel(const char* openId)
```

参数	类型	含义
openId	char*	房间其他成员的 openId

示例代码

```
iter->second.level = ITMGContextGetInstance()->GetAudioCtrl()->GetRecvStreamLevel(iter->second.openid.c_str());
```

设置扬声器的音量

此接口用于设置扬声器的音量。

接口原型

```
ITMGAudioCtrl virtual int SetSpeakerVolume(int vol)
```

参数	类型	含义
vol	Int	设置音量，范围0 – 200，当数值为0时，表示静音，当数值为100时，表示音量不增不减，默认数值为100。

示例代码

```
int vol = 100;
ITMGContextGetInstance()->GetAudioCtrl()->SetSpeakerVolume(vol);
```

获取扬声器的音量

此接口用于获取扬声器的音量。返回值为 int 类型数值，代表扬声器的音量，返回值为101代表没调用过接口 SetSpeakerVolume。Level 是实时音量，Volume 是扬声器的音量，最终声音音量 = Level × Volume %。例如实时音量是数值是100，此时 Volume 的数值是60，那么最终发出来的声音数值也是60。

接口原型

```
ITMGAudioCtrl virtual int GetSpeakerVolume()
```

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->GetSpeakerVolume();
```

开始监听房间内用户的音量变化

开始监听房间类用户（包括自己）的音量变化，当音量有变化时会触发 ITMG_MAIN_EVENT_TYPE_USER_VOLUMES 时间回调。

接口原型

```
ITMGAudioCtrl virtual int TrackingVolume(float trackingTimeS);
```

参数	类型	含义

trackingTimeS	Float	音量变化事件回调的间隔，单位为秒

消息	含义	
ITMG_MAIN_EVENT_TYPE_USER_VOLUMES	房间内用户音量发生变化	[{"open_id":"abc", "volume":2}, {"open_id":"def", "volume":2}]

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->TrackingVolume(1.0);
```

停止监听房间内用户的音量变化

停止监听房间内用户（包括自己）的音量变化。

接口原型

```
ITMGAudioCtrl virtual int StopTrackingVolume()
```

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->StopTrackingVolume();
```

设备选择相关接口

设备选择相关接口只能在 PC 端上使用。

接口	接口含义
GetMicListCount	获取麦克风设备数量
GetMicList	枚举麦克风设备
GetSpeakerListCount	获取扬声器设备数量
GetSpeakerList	枚举扬声器设备
SelectMic	选定麦克风设备
SelectSpeaker	选定扬声器设备

获取麦克风设备数量

此接口用来获取麦克风设备数量。

函数原型

```
ITMGAudioCtrl virtual int GetMicListCount()
```

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->GetMicListCount();
```

枚举麦克风设备

此接口用来枚举麦克风设备。配合 GetMicListCount 接口使用。

函数原型

```
ITMGAudioCtrl virtual int GetMicList(TMGAudioDeviceInfo* ppDeviceInfoList, int nCount)

class TMGAudioDeviceInfo
{
public:
    const char* pDeviceID;
    const char* pDeviceName;
};
```

参数	类型	含义
ppDeviceInfoList	TMGAudioDeviceInfo	设备列表
nCount	Int	获取的麦克风设备数量

TMGAudioDeviceInfo 参数	类型	含义
pDeviceID	const char*	设备名称
pDeviceName	const char*	设备 ID

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->GetMicList(ppDeviceInfoList,nCount);
```

选中麦克风设备

此接口用来选中麦克风设备。如果不调用或者传入 "DEVICEID_DEFAULT"，则选中系统默认设备。
GetMicList接口中返回的第0个设备 id 为默认设备，未选中设备时通话设备为默认设备，选中后业务层维护通话设备。如果该通话设备被拔出，则此时通话设备为默认设备，拔出的通话设备插入后，此时通话设备恢复为插入的通话设备。

函数原型

```
ITMGAudioCtrl virtual int SelectMic(const char* pMicID)
```

参数	类型	含义
pMicID	const char*	麦克风设备 ID，设备 ID 来自于 GetMicList 返回列表。

示例代码

```
const char* pMicID ="{0.0.1.00000000}.{7b0b712d-3b46-4f7a-bb83-bf9be4047f0d}";
ITMGContextGetInstance()->GetAudioCtrl()->SelectMic(pMicID);
```

获取扬声器设备数量

此接口用来获取扬声器设备数量。

函数原型

```
ITMGAudioCtrl virtual int GetSpeakerListCount()
```

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->GetSpeakerListCount();
```

枚举扬声器设备

此接口用来枚举扬声器设备。配合 GetSpeakerListCount 接口使用。

函数原型

```
ITMGAudioCtrl virtual int GetSpeakerList(TMGAudioDeviceInfo* ppDeviceInfoList, int nCount)
```

参数	类型	含义
ppDeviceInfoList	TMGAudioDeviceInfo	设备列表
nCount	Int	获取的扬声器设备数量

TMGAudioDeviceInfo 参数	类型	含义
pDeviceID	const char*	设备名称
pDeviceName	const char*	设备 ID

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->GetSpeakerList(ppDeviceInfoList,nCount);
```

选中扬声器设备

此接口用来选中播放设备。如果不调用或者传入 "DEVICEID_DEFAULT"，则选中系统默认播放设备。

函数原型

```
ITMGAudioCtrl virtual int SelectSpeaker(const char* pSpeakerID)
```

参数	类型	含义
pSpeakerID	const char*	扬声器设备 ID，设备 ID 来自于 GetSpeakerList 返回列表

示例代码

```
const char* pSpeakerID ="{0.0.1.00000000}.{7b0b712d-3b46-4f7a-bb83-bf9be4047f0d}";
ITMGContextGetInstance()->GetAudioCtrl()->SelectSpeaker(pSpeakerID);
```

高级 API

启动耳返

此接口用于启动耳返，需要 EnableLoopBack+EnableSpeaker 才可以听到自己声音。

接口原型

```
ITMGAudioCtrl virtual int EnableLoopBack(bool enable)
```

参数	类型	含义
enable	Bool	设置是否启动

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->EnableLoopBack(true);
```

获取用户房间音频类型

此接口用于获取用户房间音频类型，返回值为房间音频类型，返回值为0时代表获取用户房间音频类型发生错误，房间音频类型参考 EnterRoom 接口。

接口原型

```
class ITMGRoom {
public:
    virtual ~ITMGRoom() {} ;
    virtual int GetRoomType() = 0;

};
```

示例代码

```
ITMGContext* context = ITMGContextGetInstance();
ITMGContextGetInstance()->GetRoom()->GetRoomType();
```

房间类型修改

此接口用于修改用户房间音频类型，结果参见回调事件，事件类型为 ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE。房间的音频类型由第一个进房的人确定，此后房间里有成员修改房间类型，将对此房间所有成员生效。

接口原型

```
IITMGContext TMGRoom public int ChangeRoomType((ITMG_ROOM_TYPE roomType)
```

参数	类型	含义
roomType	ITMG_ROOM_TYPE	房间切换成的目标类型，房间音频类型参考 EnterRoom 接口

示例代码

```
ITMGContext* context = ITMGContextGetInstance();
ITMGContextGetInstance()->GetRoom()->ChangeRoomType(ITMG_ROOM_TYPE_FLUENCY);
```

Data 详情

消息	Data	例子
ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE	result; error_info; new_room_type	{"error_info":"","new_room_type":0,"result":0}

房间类型完成回调

房间类型设置完成后，回调的事件消息为 ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE，返回的参数为 result、error_info 及 new_room_type，new_room_type 代表的信息如下，在 OnEvent 函数中对事件消息进行判断。

事件子类型	代表参数	含义
ITMG_ROOM_CHANGE_EVENT_ENTERROOM	1	表示在进房的过程中，自带的音频类型与房间不符合，被修改为所进入房间的音频类型
ITMG_ROOM_CHANGE_EVENT_START	2	表示已经在房间内，音频类型开始切换（例如调用 ChangeRoomType 接口后切换音频类型 ）
ITMG_ROOM_CHANGE_EVENT_COMPLETE	3	表示已经在房间，音频类型切换完成
ITMG_ROOM_CHANGE_EVENT_REQUEST	4	表示房间成员调用 ChangeRoomType 接口，请求切换房间音频类型

示例代码

```
void TMGTestScene::OnEvent(ITMG_MAIN_EVENT_TYPE eventType,const char* data) {
    if (ITMGContext.ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE == type)
        {
        //对房间类型事件进行处理
     }
}
```

房间通话质量监控事件

质量监控事件，在进房后触发，2秒回调一次，事件消息为 ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_QUALITY，返回的参数为 weight、loss 及 delay，代表的信息如下：

参数	类型	含义
weight	Int	范围是1 – 50，数值为50是音质评分极好，数值为1是音质评分很差，几乎不能使用，数值为0代表初始值，无含义。一般数值在30以下就可以提醒用户网络较差，建议切换网络。
loss	Double	上行丢包率。
delay	Int	音频触达延迟时间（ms）。

获取版本号

获取 SDK 版本号，用于分析。

接口原型

```
ITMGContext virtual const char* GetSDKVersion()
```

示例代码

```
ITMGContextGetInstance()->GetSDKVersion();
```

设置打印日志等级

用于设置打印日志等级。建议保持默认等级。需要在 Init 之前调用。

接口原型

```
ITMGContext int SetLogLevel(ITMG_LOG_LEVEL levelWrite, ITMG_LOG_LEVEL levelPrint)
```

参数含义

参数	类型	含义
levelWrite	ITMG_LOG_LEVEL	设置写入日志的等级，TMG_LOG_LEVEL_NONE 表示不写入，默认为 TMG_LOG_LEVEL_INFO
levelPrint	ITMG_LOG_LEVEL	设置打印日志的等级，TMG_LOG_LEVEL_NONE 表示不打印，默认为 TMG_LOG_LEVEL_ERROR

ITMG_LOG_LEVEL 说明如下：

ITMG_LOG_LEVEL	含义
TMG_LOG_LEVEL_NONE	不打印日志
TMG_LOG_LEVEL_ERROR	打印错误日志（默认）
TMG_LOG_LEVEL_INFO	打印提示日志
TMG_LOG_LEVEL_DEBUG	打印开发调试日志
TMG_LOG_LEVEL_VERBOSE	打印高频日志

示例代码

```
ITMGContextGetInstance()->SetLogLevel(TMG_LOG_LEVEL_INFO,TMG_LOG_LEVEL_INFO);
```

设置打印日志路径

用于设置打印日志路径。默认路径如下。需要在 Init 之前调用。

平台	路径
Windows	%appdata%\GMEGLOBAL\GME\ProcessName
iOS	Application/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/Documents
Android	/sdcard/Android/data/xxx.xxx.xxx/files
Mac	/Users/username/Library/Containers/xxx.xxx.xxx/Data/Documents

接口原型

```
ITMGContext virtual int SetLogPath(const char* logDir)
```

参数	类型	含义
logDir	const char*	路径

示例代码

```
cosnt char* logDir = ""//自行设置路径

ITMGContext* context = ITMGContextGetInstance();
context->SetLogPath(logDir);
```

获取诊断信息

获取音视频通话的实时通话质量的相关信息。该接口主要用来查看实时通话质量、排查问题等，业务侧可以忽略。

接口原型

```
ITMGRoom virtual const char* GetQualityTips()
```

示例代码

```
ITMGContextGetInstance()->GetRoom()->GetQualityTips();
```

回调消息

消息	含义	Data	例子
ITMG_MAIN_EVENT_TYPE_ENTER_ROOM	进入音频房间消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_EXIT_ROOM	退出音频房间消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_ROOM_DISCONNECT	房间因为网络等原因断开消息	result; error_info	{"error_info":"waiting timeout, please check your network","result":0}
ITMG_MAIN_EVNET_TYPE_USER_UPDATE	房间成员更新消息	user_list; event_id	{"event_id":1,"user_list":["0"]}
ITMG_MAIN_EVENT_TYPE_RECONNECT_START	房间重连开始消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_RECONNECT_SUCCESS	房间重连成功消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_SWITCH_ROOM	快速切换房间消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE	房间状态切换消息	result; error_info; sub_event_type; new_room_type	{"error_info":"","new_room_type":0,"result":0}

ITMG_MAIN_EVENT_TYPE_ROOM_SHARING_START	开始跨房连麦消息	result;	{"result":0}
ITMG_MAIN_EVENT_TYPE_ROOM_SHARING_STOP	跨房连麦停止消息	result;	{"result":0}
ITMG_MAIN_EVENT_TYPE_SPEAKER_DEFAULT_DEVICE_CHANGED	默认扬声器设备修改消息	result; error_info	{"deviceID":"{0.0.0.00000000}.{a4f1e8be-49fa-43e2-b8cf-dd00542b47ae}","deviceName":"扬声器 (Realtek High Definition Audio)","error_info":"","isNewDevice":true,"isUsedDevice":false,"result":0}
ITMG_MAIN_EVENT_TYPE_SPEAKER_NEW_DEVICE	新增扬声器设备消息	result; error_info	{"deviceID":"{0.0.0.00000000}.{a4f1e8be-49fa-43e2-b8cf-dd00542b47ae}","deviceName":"扬声器 (Realtek High Definition Audio)","error_info":"","isNewDevice":true,"isUsedDevice":false,"result":0}
ITMG_MAIN_EVENT_TYPE_SPEAKER_LOST_DEVICE	丢失扬声器设备消息	result; error_info	{"deviceID":"{0.0.0.00000000}.{a4f1e8be-49fa-43e2-b8cf-dd00542b47ae}","deviceName":"扬声器 (Realtek High Definition Audio)","error_info":"","isNewDevice":false,"isUsedDevice":false,"result":0}
ITMG_MAIN_EVENT_TYPE_MIC_NEW_DEVICE	新增麦克风设备消息	result; error_info	{"deviceID":"{0.0.1.00000000}.{5fdf1a5b-f42d-4ab2-890a-7e454093f229}","deviceName":"麦克风 (Realtek High Definition Audio)","error_info":"","isNewDevice":true,"isUsedDevice":true,"result":0}
ITMG_MAIN_EVENT_TYPE_MIC_LOST_DEVICE	丢失麦克风设备消息	result; error_info	{"deviceID":"{0.0.1.00000000}.{5fdf1a5b-f42d-4ab2-890a-7e454093f229}","deviceName":"麦克风 (Realtek High Definition Audio)","error_info":"","isNewDevice":false,"isUsedDevice":true,"result":0}
ITMG_MAIN_EVENT_TYPE_MIC_DEFAULT_DEVICE_CHANGED	默认麦克风设备修改消息	result; error_info	{"deviceID":"{0.0.1.00000000}.{5fdf1a5b-f42d-4ab2-890a-7e454093f229}","deviceName":"麦克风 (Realtek High Definition Audio)","error_info":"","isNewDevice":false,"isUsedDevice":true,"result":0}
ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_QUALITY	房间质量消息	weight; loss; delay	{"weight":5,"loss":0.1,"delay":1}
ITMG_MAIN_EVNET_TYPE_PTT_RECORD_COMPLETE	语音消息录制完成消息	result; file_path	{"file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_UPLOAD_COMPL	语音消息上传完成消息	result; file_path;file_i	{"file_id":"","file_path":"","result":0}

ETE		d	
ITMG_MAIN_EVNET_TYPE_PTT_DOWNLOAD_COMPLETE	语音消息下载完成消息	result;file_path;file_id	{"file_id":"","file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_PLAY_COMPLETE	语音消息播放完成消息	result;file_path	{"file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_SPEECH2TEXT_COMPLETE	语音消息极速转文本完成消息	result;text;file_id	{"file_id":"","text":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_COMPLETE	语音消息流式转文本完成消息	result;file_path;text;file_id	{"file_id":"","file_path":","text":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_IS_RUNNING	语音消息正在流式转文本中	result;file_path;text;file_id	{"file_id":"","file_path":","text":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_TEXT2SPEECH_COMPLETE	文本转语音完成消息	result;text;file_id	{"file_id":"","text":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_TRANSLATE_TEXT_COMPLETE	文本翻译完成消息	result;text;file_id	{"file_id":"","text":"","result":0}

版权所有：腾讯云计算（北京）有限责任公司

语音消息及转文本

最近更新时间：2024-12-17 17:39:22

为方便 Unreal Engine 开发者调试和接入腾讯云游戏多媒体引擎客户端 API，本文为您介绍适用于 Unreal Engine 语音消息服务及转文本服务的接入技术。

使用 GME 重要事项

GME 提供实时语音服务、语音消息服务及转文本服务，使用 GME 服务都依赖 Init 和 Poll 等核心接口。

重点提示

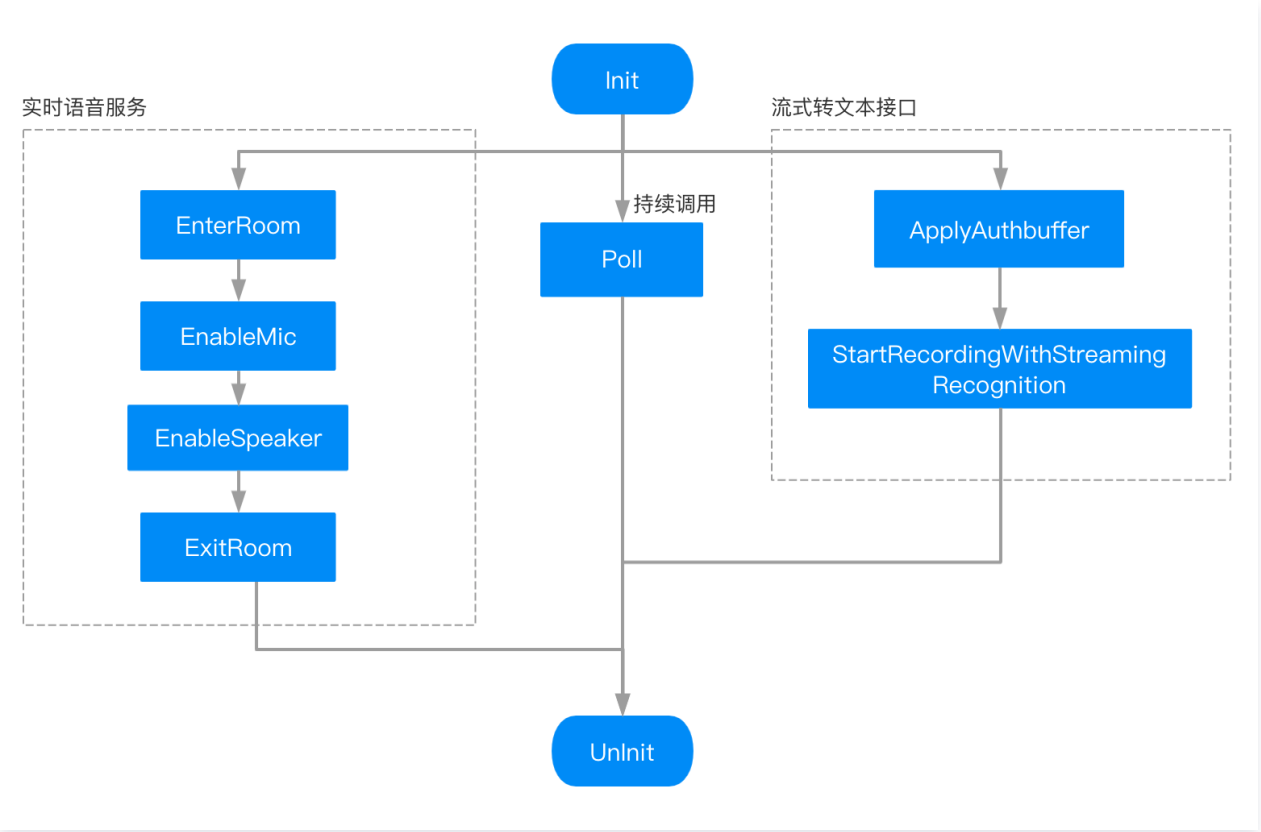
- 已完成 GME 应用创建，并获取 SDK AppID 和 Key。请参见 服务开通指引 。
- 已开通 **GME 实时语音服务、语音消息服务以及转文本服务**。请参见 服务开通指引 。
- GME 使用前请对工程进行配置，否则 SDK 不生效。
- GME 的接口调用成功后返回值为 QAVError.OK，数值为 0。
- GME 的接口调用要在同一个线程下。
- GME 需要周期性的调用 Poll 接口触发事件回调。
- 错误码详情可参考 错误码 。

> ⚠ **注意**
> 语音转文本相关接口有默认频率限制，限额范围内计费方式请参见 计费文档；若需提升接口频率限额或了解超额计费方式，请联系商务或 提交工单咨询 。
> - 语音消息非流式转文本接口 ***SpeechToText()***：默认单账号限制并发数为10路
> - 语音消息流式转文本接口 ***StartRecordingWithStreamingRecognition()***：默认单账号限制并发数为50路
> - 实时语音流式转文本接口 ***StartRealTimeASR()***：默认单账号限制并发数为50路

接入 SDK

重要步骤

接入 SDK 重要流程如下：



1. 初始化 GME
2. 周期性调用 Poll 触发回调
3. 鉴权初始化
4. 启动流式语音识别
5. 停止录制
6. 反初始化 GME

C++ 类

类	含义
ITMGContext	核心接口
ITMGPTT	语音消息及转文本相关接口

核心接口

接口	接口含义
Init	初始化 GME
Poll	触发事件回调
Pause	系统暂停
Resume	系统恢复
Uninit	反初始化 GME

准备工作

接入 GME 首先需要引入头文件 tmg_sdk.h，头文件类继承 ITMGDelegate 以进行消息的传递及回调。

示例代码 /

```
#include "GMESDK/tmg_sdk.h"

class UEDEMO1_API AUEDemoLevelScriptActor : public ALevelScriptActor, public ITMGDelegate
{
public:
...
private:
...
}
```

设置单例

在 EnterRoom 函数调用之前要先获取 ITMGContext ，所有调用都从 ITMGContext 开始，由ITMGDelegate 回调回传给应用，必须首先设置。

示例代码

```
ITMGContext* context = ITMGContextGetInstance();
context->SetTMGDelegate(this);
```

消息传递

接口类采用 Delegate 方法用于向应用程序发送回调通知，消息类型参考 ITMG_MAIN_EVENT_TYPE，data 在 Windows 平台下是 json 字符串格式， 具体 key-value 参见说明文档。

示例代码

```
//函数实现：
//UEDemoLevelScriptActor.h:
class UEDEMO1_API AUEDemoLevelScriptActor : public ALevelScriptActor, public SetTMGDelegate
{
public:
    void OnEvent(ITMG_MAIN_EVENT_TYPE eventType, const char* data);
}

//UEDemoLevelScriptActor.cpp:
void AUEDemoLevelScriptActor::OnEvent(ITMG_MAIN_EVENT_TYPE eventType, const char* data){
    //在此对eventType进行判断及操作
}
```

初始化 SDK

未初始化前，SDK 处于未初始化阶段，**需要通过接口 Init 初始化 SDK**，才可以使用实时语音服务、语音消息服务及转文本服务。调用 Init 接口的线程必须于其他接口在同一线程,建议都在主线程调用接口。

接口原型

```
ITMGContext virtual int Init(const char* sdkAppId, const char* openId)
```

参数	类型	含义
sdkAppId	const char*	来自 腾讯云控制台 的 GME 服务提供的 AppID，获取请参见 语音服务开通指引 。
openID	const char*	openID 只支持 Int64 类型（转为 const char* 传入）。规则由 App 开发者自行制定，App 内不重复即可。如需使用字符串作为 Openid 传入，可 提交工单 联系开发者。

返回值

返回值	处理
AV_OK = 0	初始化 SDK 成功
AV_ERR_SDK_NOT_FULL_UPDATE=7015	检查 SDK 文件是否完整，建议删除后重新导入 SDK

⚠ 关于7015错误提示

- 7015错误码是通过 md5 进行判断，在接入过程中若出现此错误，请根据提示检查 SDK 文件是否完整、SDK 文件版本是否一致。
- 出现返回值 AV_ERR_SDK_NOT_FULL_UPDATE 时，此返回值**只有提示作用**，并不会造成初始化失败。
- 由于第三方加固、Unity 打包机制等因素会影响库文件 md5，造成误判，所以**正式发布请在逻辑中忽略此错误**，并尽量不在 UI 中提示。

示例代码

```
ITMGContextGetInstance()->Init("14000XXXXX", "100000");
```

触发事件回调

通过在 update 里面周期的调用 Poll 可以触发事件回调。Poll 是 GME 的消息泵，GME 需要周期性的调用 Poll 接口触发事件回调。如果没有调用 Poll ，将会导致整个 SDK 服务运行异常。详情请参见 Sample Project 中的 EnginePollHelper 文件。

⚠ 注意

务必周期性调用 Poll 接口且在主线程调用，以免接口回调异常。

接口原型

```
class ITMGContext {
protected:
    virtual ~ITMGContext() {}

public:
    virtual void Poll()= 0;
}
```

示例代码

```
//头文件中的声明
virtual void Tick(float DeltaSeconds);

void AUEDemoLevelScriptActor::Tick(float DeltaSeconds) {
    Super::Tick(DeltaSeconds);
    ITMGContextGetInstance()->Poll();
```

```
}
```

系统暂停

当系统发生 Pause 事件时，需要同时通知引擎进行 Pause。如果不需要后台播放房间内声音，请调用 Pause 接口暂停整个 GME 服务。

接口原型

```
ITMGContext int Pause()
```

系统恢复

当系统发生 Resume 事件时，需要同时通知引擎进行 Resume。Resume 接口只恢复实时语音。

接口原型

```
ITMGContext int Resume()
```

反初始化 SDK

反初始化 SDK，进入未初始化状态。**如果游戏业务侧账号与 openid 是绑定的，那切换游戏账号需要反初始化 GME，再用新的 openid 初始化。**

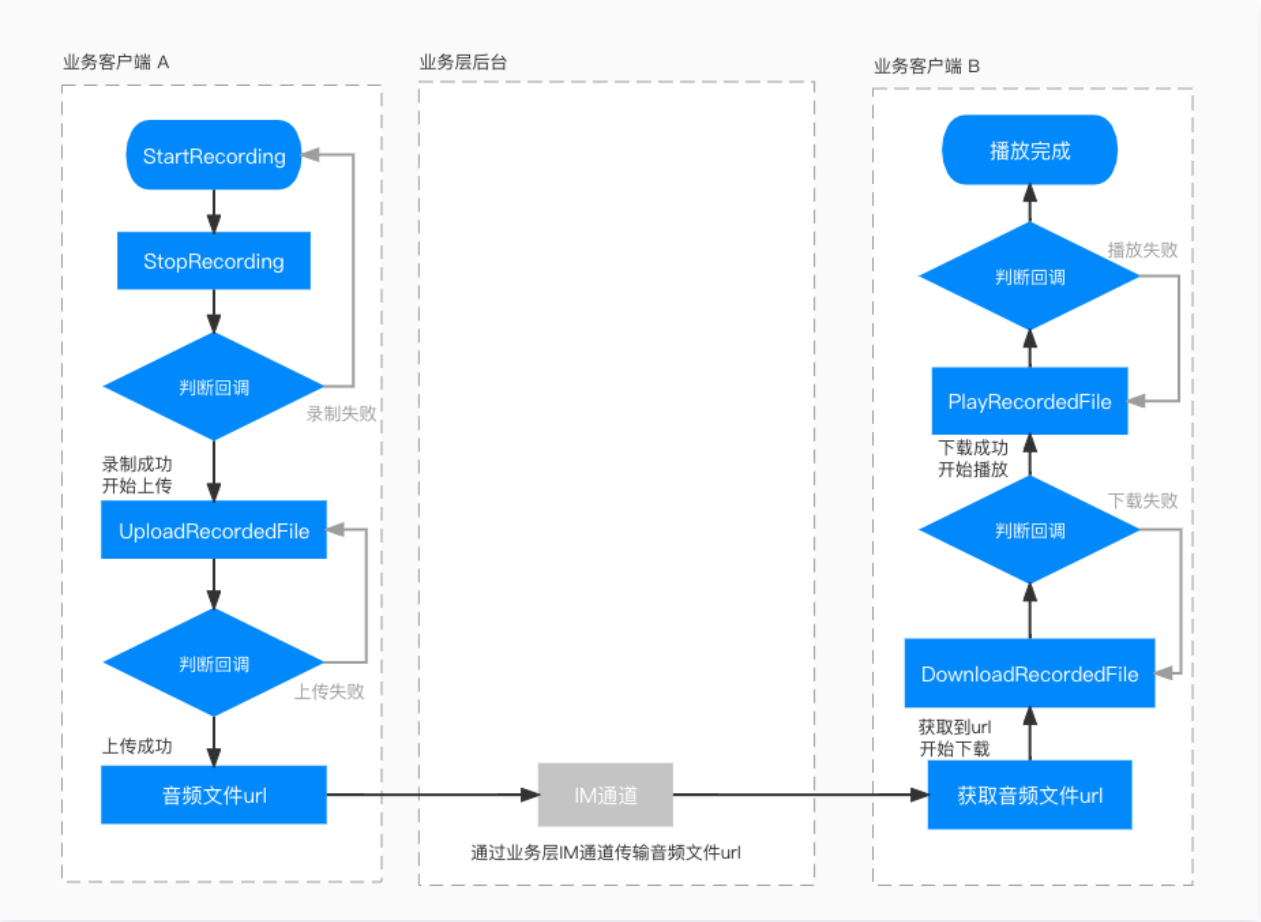
接口原型

```
ITMGContext int Uninit()
```

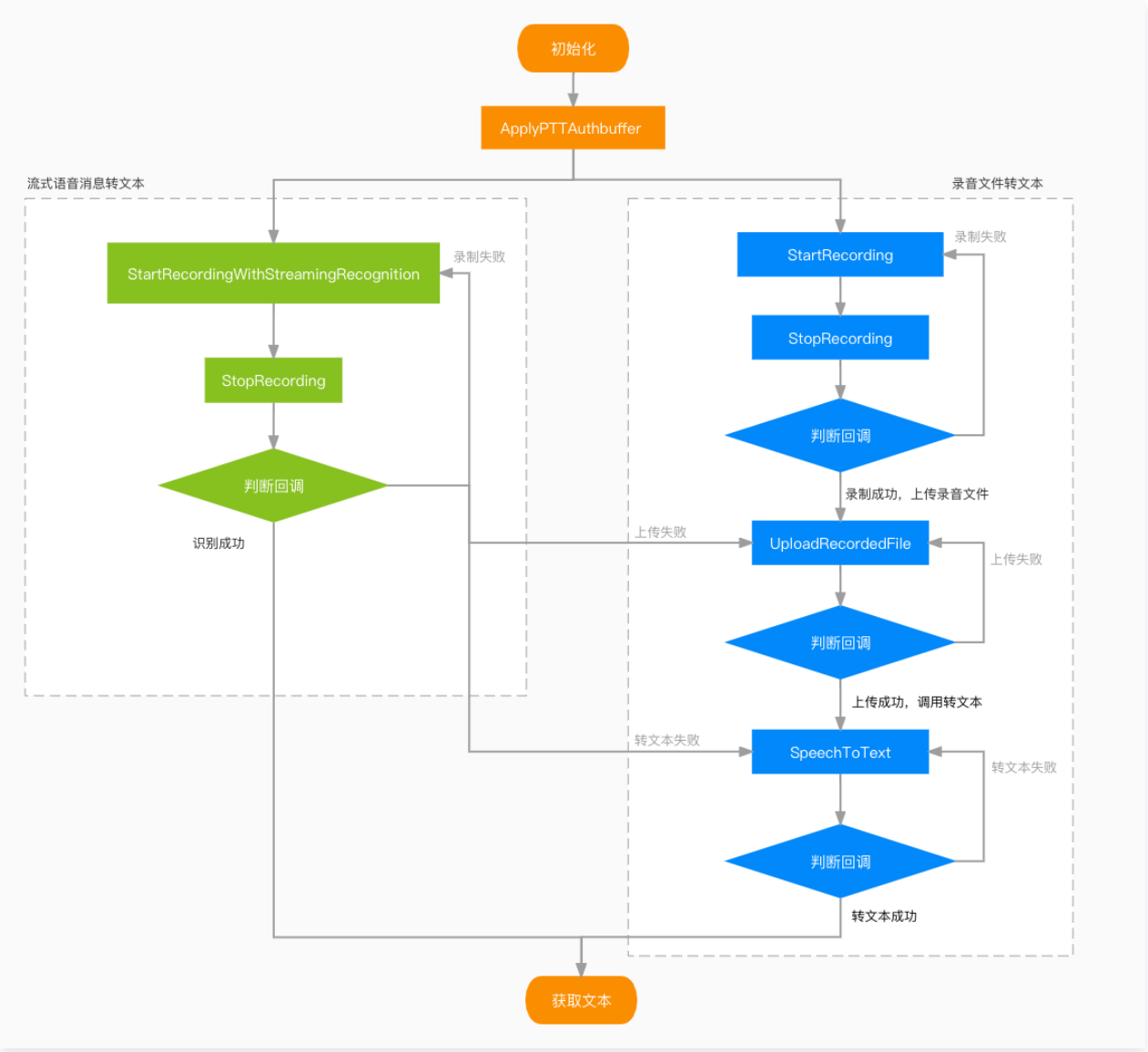
语音消息服务及转文本服务

> **说明**
> - 转文本服务分录音文件极速转文本以及语音消息流式转文本。
> - 使用语音消息服务不需要进入实时语音房间。
> - 语音消息最大录制时长默认为58秒，最短不能小于1秒。如果需要再加以限制，例如限制为最大录制时长为10秒，请在初始化之后调用 SetMaxMessageLength 接口进行设置。

语音消息服务使用流程


业务客户端 A
业务层后台
业务客户端 B
StartRecording
StopRecording
判断回调
录制失败
录制成功
开始上传
UploadRecordedFile
判断回调
上传失败
上传成功
音频文件url
IM通道
通过业务层IM通道传输音频文件url
播放完成
判断回调
播放失败
PlayRecordedFile
下载成功
开始播放
判断回调
下载失败
DownloadRecordedFile
获取到url
开始下载
获取音频文件url

转文本服务使用流程



接口	接口含义
GenAuthBuffer	生成本地鉴权
ApplyPTTAuthbuffer	鉴权初始化
SetMaxMessageLength	限制最大语音信息时长

生成本地鉴权

生成 AuthBuffer，用于相关功能的加密和鉴权，如正式发布请使用后台部署密钥，后台部署请参见 鉴权密钥 。

接口原型

```
int  QAVSDK_AuthBuffer_GenAuthBuffer(unsigned int dwSdkAppID, const char* strRoomID, const char*
strOpenID,
    const char* strKey, unsigned char* strAuthBuffer, unsigned int bufferLength);
```

参数	类型	含义

dwSdkAppID	int	来自腾讯云控制台的 AppId 号码。
strRoomID	const char*	填 null 或者空字符串。
strOpenID	const char*	用户标识。与 Init 时候的 openID相同。
strKey	const char*	来自腾讯云 控制台 的权限密钥。
strAuthBuffer	const char*	返回的 authbuff
bufferLength	int	传入的 authbuff 长度，建议为 500

应用鉴权

生成鉴权信息后，将鉴权赋值到 SDK 中。

接口原型

```
ITMGPTT virtual int ApplyPTTAuthbuffer(const char* authBuffer, int authBufferLen)
```

参数	类型	含义
authBuffer	const char*	鉴权
authBufferLen	int	鉴权长度

示例代码

```
ITMGContextGetInstance()->GetPTT()->ApplyPTTAuthbuffer(authBuffer,authBufferLen);
```

限制最大语音信息时长

限制最大语音消息的长度，最大支持58秒。

接口原型

```
ITMGPTT virtual int SetMaxMessageLength(int msTime)
```

参数	类型	含义
msTime	int	语音时长，单位 ms，区间为 1000 < msTime < = 58000

示例代码

```
int msTime = 10000;
ITMGContextGetInstance()->GetPTT()->SetMaxMessageLength(msTime);
```

流式语音识别

语音消息及转文字相关接口

接口	接口含义
StartRecordingWithStreamingRecognition	启动流式录音

StopRecording	停止录音

启动流式语音识别

此接口用于启动流式语音识别，同时在回调中会有实时的语音转文字返回，可以指定语言进行识别，也可以将语音中识别到的信息翻译成指定的语言返回。**停止录音调用** 停止录制 。

接口原型

```
ITMGPTT virtual int StartRecordingWithStreamingRecognition(const char* filePath)
ITMGPTT virtual int StartRecordingWithStreamingRecognition(const char* filePath,const char*
translateLanguage,const char* translateLanguage)
```

参数	类型	含义
filePath	const char*	存放的语音路径
speechLanguage	const char*	识别成指定文字的语言参数，参数请参见 语言参数参考列表
translateLanguage	const char*	翻译成指定文字的语言参数，参数请参见 语言参数参考列表

示例代码

```
ITMGContextGetInstance()->GetPTT()->StartRecordingWithStreamingRecognition(filePath,"cmn-Hans-
CN","cmn-Hans-CN");
```

> **注意**
> 翻译会收取额外费用，请参见 购买指南 。

流式语音识别的回调

启动流式语音识别后，需要在 OnEvent 通知中监听回调消息，事件消息分为以下两个：

- `ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_COMPLETE` 是在停止录制并完成识别后才返回文字，相当于一段话说完才会返回识别的文字。
- `ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_IS_RUNNING` 是在录音过程中就会实时返回识别到的文字，相当于边说话边返回识别到的文字。

根据需求在 OnEvent 通知中对相应事件消息进行判断。传递的参数包含以下四个信息。

消息名称	含义
result	用于判断流式语音识别是否成功的返回码
text	语音转文字识别的文本
file_path	录音存放的本地地址
file_id	录音在后台的 url 地址，录音在服务器存放90天

> **注意**
> 监听 `ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_IS_RUNNING` 消息时，file_id 为空。

错误码

错误码	含义	处理方式
32775	流式语音转文本失败，但是录音成功	调用 UploadRecordedFile 接口上传录音，再调用 SpeechToText 接口进行语音转文字操作
32777	流式语音转文本失败，但是录音成功，上传成功	返回的信息中有上传成功的后台 url 地址，调用 SpeechToText 接口进行语音转文字操作
32786	流式语音转文本失败	在流式录制状态当中，请等待流式录制接口执行结果返回
32787	转文本成功，文本翻译服务未开通	需要在控制台开通文本翻译服务
32788	转文本成功，文本翻译语言参数不支持	重新检查传入参数

如果出现 4098 错误码，请参见 常见问题文档 进行解决。

示例代码

```
void TMGTestScene::OnEvent(ITMG_MAIN_EVENT_TYPE eventType,const char* data){
                switch (eventType) {
                    case ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_COMPLETE:
                    {
                        HandleSTREAM2TEXTComplete(data,true);
                        break;
                        }
                    ...
                            case ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_IS_RUNNING:
                    {
                        HandleSTREAM2TEXTComplete(data, false);
                        break;
                    }
                }
}

void CTMGSDK_For_AudioDlg::HandleSTREAM2TEXTComplete(const char* data, bool isComplete)
{
                    std::string strText = "STREAM2TEXT: ret=";
                    strText += data;
                    m_EditMonitor.SetWindowText(MByteToWChar(strText).c_str());
                    Json::Reader reader;
                    Json::Value root;
                    bool parseRet = reader.parse(data, root);
                    if (!parseRet) {
                        ::SetWindowText(m_EditInfo.GetSafeHwnd(),MByteToWChar(std::string("parse
result Json error")).c_str());
                    }
                    else
                    {
                        if (isComplete) {
                            ::SetWindowText(m_EditUpload.GetSafeHwnd(),
MByteToWChar(root["file_id"].asString()).c_str());
                        }
```

```
                        else {
                            std::string isruning = "STREAMINGRECOGNITION_IS_RUNNING";
                            ::SetWindowText(m_EditUpload.GetSafeHwnd(),
MByteToWChar(isruning).c_str());
                        }
                    }
}
```

语音消息录制

录制的流程为：录音 > 停止录音 > 录音回调返回 > 启动下一次录音。

语音消息及转文字相关接口

接口	接口含义
StartRecording	启动录音
PauseRecording	暂停录音
ResumeRecording	恢复录音
StopRecording	停止录音
CancelRecording	取消录音

启动录音

此接口用于启动录音。

接口原型

```
ITMGPTT virtual int StartRecording(const char* fileDir)
```

参数	类型	含义
fileDir	const char*	存放的语音路径

示例代码

```
char buffer[256]={0};
snprintf(buffer, sizeof(buffer), "%sunreal_ptt_local.file", getFilePath().c_str());
ITMGContextGetInstance()->GetPTT()->StartRecording(buffer);
```

停止录音

此接口用于停止录音。此接口为异步接口，停止录音后会有录音完成回调，成功之后录音文件才可用。

接口原型

```
ITMGPTT virtual int StopRecording()
```

示例代码

```
ITMGContextGetInstance()->GetPTT()->StopRecording();
```

启动录音的回调

启动录音的结果会通过回调返回。
停止录音调用StopRecording。停止录音后才有启动录音的回调。

参数	类型	含义
result	int32	当 code 为 0 时，录制完成
filepath	FString	录制的存放地址，必须是可以访问到的路径，不可将 fileid 作为路径

错误码

错误码值	原因	建议方案
4097	参数为空	检查代码中接口参数是否正确
4098	初始化错误	检查设备是否被占用，或者权限是否正常，是否初始化正常
4099	正在录制中	确保在正确的时机使用 SDK 录制功能
4100	没有采集到音频数据	检查麦克风设备是否正常
4101	录音时，录制文件访问错误	确保文件存在，文件路径的合法性
4102	麦克风未授权错误	使用 SDK 需要麦克风权限，添加权限请参考对应引擎或平台的 SDK 工程配置文档
4103	录音时间太短错误	首先，限制录音时长的单位为毫秒，检查参数是否正确；其次，录音时长要1000毫秒以上才能成功录制
4104	没有启动录音操作	检查是否已经调用启动录音接口

示例代码

```
void TMGTestScene::OnEvent(ITMG_MAIN_EVENT_TYPE eventType,const char* data){
    switch (eventType) {
    ...
        else if (eventType == ITMG_MAIN_EVNET_TYPE_PTT_RECORD_COMPLETE) {
            int32 result = JsonObject->GetIntegerField(TEXT("result"));
            FString filepath = JsonObject->GetStringField(TEXT("file_path"));
            std::string path = TCHAR_TO_UTF8(filepath.operator*());
            int duration = 0;
            int filesize = 0;
            if (result == 0) {
                duration = ITMGContextGetInstance()->GetPTT()-
>GetVoiceFileDuration(path.c_str());
                filesize = ITMGContextGetInstance()->GetPTT()->GetFileSize(path.c_str());
            }
            onPttRecordFileCompleted(result, filepath, duration, filesize);
        }
    }
}
```

暂停录音

此接口用于暂停录音。如需恢复录音请调用接口 ResumeRecording。

接口原型

```
ITMGPTT virtual int PauseRecording()
```

示例代码

```
ITMGContextGetInstance()->GetPTT()->PauseRecording();
```

恢复录音

此接口用于恢复录音。

接口原型

```
ITMGPTT virtual int ResumeRecording()
```

示例代码

```
ITMGContextGetInstance()->GetPTT()->ResumeRecording();
```

取消录音

调用此接口取消录音。**取消之后没有回调。**

接口原型

```
ITMGPTT virtual int CancelRecording()
```

示例代码

```
ITMGContextGetInstance()->GetPTT()->CancelRecording();
```

语音消息上传、下载及播放

接口	接口含义
UploadRecordedFile	上传语音文件
DownloadRecordedFile	下载语音文件
PlayRecordedFile	播放语音
StopPlayFile	停止播放语音
GetFileSize	语音文件的大小
GetVoiceFileDuration	语音文件的时长

上传语音文件

此接口用于上传语音文件。

接口原型

```
ITMGPTT virtual int UploadRecordedFile(const char* filePath)
```

参数	类型	含义
filePath	const char*	上传的语音路径，此路径为本地路径

示例代码

```
ITMGContextGetInstance()->GetPTT()->UploadRecordedFile(filePath);
```

上传语音完成的回调

上传语音完成后，事件消息为 ITMG_MAIN_EVNET_TYPE_PTT_UPLOAD_COMPLETE，在 OnEvent 函数中对事件消息进行判断。传递的参数包含三个信息，result，file_path 和 file_id。

参数	类型	含义
result	int32	当 code 为0时，录制完成
filepath	FString	录制的存放地址
fileid	FString	文件的 url 路径

错误码

错误码值	原因	建议方案
8193	上传文件时，文件访问错误	确保文件存在，文件路径的合法性
8194	签名校验失败错误	检查鉴权密钥是否正确，检查是否有初始化离线语音
8195	网络错误	检查设备网络是否可以正常访问外网环境
8196	获取上传参数过程中网络失败	检查鉴权是否正确，检查设备网络是否可以正常访问外网环境
8197	获取上传参数过程中回包数据为空	检查鉴权是否正确，检查设备网络是否可以正常访问外网环境
8198	获取上传参数过程中回包解包失败	检查鉴权是否正确，检查设备网络是否可以正常访问外网环境
8200	没有设置 appinfo	检查 apply 接口是否有调用，或者入参是否为空

示例代码

```
void TMGTestScene::OnEvent(ITMG_MAIN_EVENT_TYPE eventType,const char* data){
    switch (eventType) {
    ...
        else if (eventType == ITMG_MAIN_EVNET_TYPE_PTT_UPLOAD_COMPLETE) {
            int32 result = JsonObject->GetIntegerField(TEXT("result"));
            FString filepath = JsonObject->GetStringField(TEXT("file_path"));
            FString fileid = JsonObject->GetStringField(TEXT("file_id"));
            onPttUploadFileCompleted(result, filepath, fileid);
        }
    }
```

```
}
```

下载语音文件

此接口用于下载语音文件。

接口原型

```
ITMGPTT virtual int DownloadRecordedFile(const char* fileId, const char* filePath)
```

参数	类型	含义
fileId	const char*	文件的 url 路径
filePath	const char*	文件的本地保存路径

示例代码

```
ITMGContextGetInstance()->GetPTT()->DownloadRecordedFile(fileID,filePath);
```

下载语音文件完成回调

下载语音完成后，事件消息为 ITMG_MAIN_EVNET_TYPE_PTT_DOWNLOAD_COMPLETE，在 OnEvent 函数中对事件消息进行判断。
传递的参数包含三个信息，result、file_path 和 file_id。

参数	类型	含义
result	int32	当 code 为0时，录制完成
filepath	FString	录制的存放地址
fileid	FString	文件的 url 路径，录音在服务器存放 90 天

错误码

错误码值	原因	建议方案
12289	下载文件时，文件访问错误	检查文件路径是否合法
12290	签名校验失败	检查鉴权密钥是否正确，检查是否有初始化离线语音
12291	网络存储系统异常	服务器获取语音文件失败，检查接口参数 fileid 是否正确，检查网络是否正常，检查 COS 文件存不存在
12292	服务器文件系统错误	检查设备网络是否可以正常访问外网环境，检查服务器上是否有此文件
12293	获取下载参数过程中，HTTP 网络失败	检查设备网络是否可以正常访问外网环境
12294	获取下载参数过程中，回包数据为空	检查设备网络是否可以正常访问外网环境
12295	获取下载参数过程中，回包解包失败	检查设备网络是否可以正常访问外网环境

12297	没有设置 appinfo	检查鉴权密钥是否正确，检查是否有初始化离线语音

示例代码

```
void TMGTestScene::OnEvent(ITMG_MAIN_EVENT_TYPE eventType,const char* data){
    switch (eventType) {
    ...
        else if (eventType == ITMG_MAIN_EVNET_TYPE_PTT_DOWNLOAD_COMPLETE) {
            int32 result = JsonObject->GetIntegerField(TEXT("result"));
            FString filepath = JsonObject->GetStringField(TEXT("file_path"));
            FString fileid = JsonObject->GetStringField(TEXT("file_id"));
            onPttDownloadFileCompleted(result, filepath, fileid);
        }
    }
}
```

播放语音

此接口用于播放语音。

接口原型

```
ITMGPTT virtual int PlayRecordedFile(const char* filePath)
ITMGPTT virtual int PlayRecordedFile(const char* filePath, nt voiceType)
```

参数	类型	含义
filePath	const char*	本地语音文件的路径
voicetype	int	变声类型，请参见 变声接入文档

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

示例代码

```
ITMGContextGetInstance()->GetPTT()->PlayRecordedFile(filePath);
```

播放语音的回调

播放语音的回调，事件消息为 ITMG_MAIN_EVNET_TYPE_PTT_PLAY_COMPLETE， 在 OnEvent 函数中对事件消息进行判断。传递的参数包含两个信息，一个是 result，另一个是 file_path。

参数	类型	含义
code	int	当 code 为0时，播放完成
filepath	FString	录制的存放地址

错误码

错误码值	原因	建议方案
20481	初始化错误	检查设备是否被占用，或者权限是否正常，是否初始化正常
20482	正在播放中，试图打断并播放下一个失败了（正常是可以打断的）	检查代码逻辑是否正确
20483	参数为空	检查代码中接口参数是否正确
20484	内部错误	初始化播放器错误，解码失败等问题产生此错误码，需要结合日志定位问题

示例代码

```
void TMGTestScene::OnEvent(ITMG_MAIN_EVENT_TYPE eventType,const char* data){
    switch (eventType) {
    ...
        else if (eventType == ITMG_MAIN_EVNET_TYPE_PTT_PLAY_COMPLETE) {
            int32 result = JsonObject->GetIntegerField(TEXT("result"));
            FString filepath = JsonObject->GetStringField(TEXT("file_path"));
            onPttPlayFileCompleted(result, filepath);
        }
    }
}
```

停止播放语音

此接口用于停止播放语音。停止播放语音也会有播放完成的回调。

接口原型

```
ITMGPTT virtual int StopPlayFile()
```

示例代码

```
ITMGContextGetInstance()->GetPTT()->StopPlayFile();
```

获取语音文件的大小

通过此接口，获取语音文件的大小。

接口原型

```
ITMGPTT virtual int GetFileSize(const char* filePath)
```

参数	类型	含义
filePath	const char*	语音文件的路径，此路径为本地路径

示例代码

```
ITMGContextGetInstance()->GetPTT()->GetFileSize(filePath);
```

获取语音文件的时长

此接口用于获取语音文件的时长，单位毫秒。

接口原型

```
ITMGPTT virtual int GetVoiceFileDuration(const char* filePath)
```

参数	类型	含义
filePath	const char*	语音文件的路径，此路径为本地路径

示例代码

```
ITMGContextGetInstance()->GetPTT()->GetVoiceFileDuration(filePath);
```

录音文件极速转文本

接口	接口含义
SpeechToText	语音识别成文字

将指定的语音文件识别成文字

此接口用于将指定的语音文件识别成文字。

接口原型

```
ITMGPTT virtual void SpeechToText(const char* fileID)
```

参数	类型	含义
fileID	const char*	语音文件 url

示例代码

```
ITMGContextGetInstance()->GetPTT()->SpeechToText(fileID);
```

将指定的语音文件翻译成文字（指定语言）

此接口可以指定语言进行识别，也可以将语音中识别到的信息翻译成指定的语言返回。

> ⚠ **注意**
> 翻译会收取额外费用，请参见 购买指南 。

接口原型

```
ITMGPTT virtual int SpeechToText(const char* fileID,const char* speechLanguage)
ITMGPTT virtual int SpeechToText(const char* fileID,const char* speechLanguage,const char*
translateLanguage)
```

参数	类型	含义

fileID	const char*	语音文件 url，录音在服务器存放90天
speechLanguage	const char*	识别出指定文字的语言参数，参数参见 语言参数参考列表
translatelanguage	const char*	翻译成指定文字的语言参数，参数参见 言参数参考列表

示例代码

```
ITMGContextGetInstance()->GetPTT()->SpeechToText(filePath,"cmn-Hans-CN","cmn-Hans-CN");
```

识别回调

将指定的语音文件识别成文字的回调，事件消息为 ITMG_MAIN_EVNET_TYPE_PTT_SPEECH2TEXT_COMPLETE， 在 OnEvent 函数中对事件消息进行判断。
传递的参数包含三个信息，result、file_path 和 text，其中 text 为识别的文本。

参数	类型	含义
result	int32	当 code 为0时，录制完成
fileid	FString	语音文件 url，录音在服务器存放 90 天
text	FString	转换的文本结果

错误码

错误码值	原因	建议方案
32769	内部错误	分析日志，获取后台返回给客户端的真正错误码，并联系后台同事协助解决
32770	网络失败	检查设备网络是否可以正常访问外网环境
32772	回包解包失败	分析日志，获取后台返回给客户端的真正错误码，并联系后台同事协助解决
32774	没有设置 appinfo	检查鉴权密钥是否正确，检查是否有初始化离线语音
32776	authbuffer 校验失败	检查 authbuffer 是否正确
32784	语音转文本参数错误	检查代码中接口参数 fileid 是否为空
32785	语音转文本翻译返回错误	离线语音后台错误，请分析日志，获取后台返回给客户端的真正错误码，并联系后台同事协助解决
32787	转文本成功，文本翻译服务未开通	需要在控制台开通文本翻译服务
32788	转文本成功，文本翻译语言参数不支持	重新检查传入参数

示例代码

```
void TMGTestScene::OnEvent(ITMG_MAIN_EVENT_TYPE eventType,const char* data){
    switch (eventType) {
    ...
        else if (eventType == ITMG_MAIN_EVNET_TYPE_PTT_SPEECH2TEXT_COMPLETE) {
            int32 result = JsonObject->GetIntegerField(TEXT("result"));
            FString text = JsonObject->GetStringField(TEXT("text"));
            FString fileid = JsonObject->GetStringField(TEXT("file_id"));
```

```
            onPttSpeech2TextCompleted(result, fileid, text);
        }
    }
}
```

语音消息音量相关接口

接口	接口含义
GetMicLevel	获取实时麦克风音量
SetMicVolume	设置录制音量
GetMicVolume	获取录制音量
GetSpeakerLevel	获取实时扬声器音量
SetSpeakerVolume	设置播放音量
GetSpeakerVolume	获取播放音量

获取语音消息麦克风实时音量

此接口用于获取麦克风实时音量，返回值为 int 类型，值域为0 – 200。

接口原型

```
ITMGPTT virtual int GetMicLevel()
```

示例代码

```
ITMGContext.GetInstance(this).GetPTT().GetMicLevel();
```

设置语音消息录制音量

此接口用于设置离线语音录制音量，值域为0 – 200。

接口原型

```
ITMGPTT virtual int SetMicVolume(int vol)
```

示例代码

```
ITMGContextGetInstance()->GetPTT()->SetMicVolume(100);
```

获取语音消息录制音量

此接口用于获取离线语音录制音量。返回值为 int 类型，值域为0 – 200。

接口原型

```
ITMGPTT virtual int GetMicVolume()
```

示例代码

```
ITMGContextGetInstance()->GetPTT()->GetMicVolume();
```

获取语音消息扬声器实时音量

此接口用于获取扬声器实时音量。返回值为 int 类型，值域为0 – 200。

接口原型

```
ITMGPTT virtual int GetSpeakerLevel()
```

示例代码

```
ITMGContextGetInstance()->GetPTT()->GetSpeakerLevel();
```

设置语音消息播放音量

此接口用于设置离线语音播放音量，值域为0 – 200。

接口原型

```
ITMGPTT virtual int SetSpeakerVolume(int vol)
```

示例代码

```
ITMGContextGetInstance()->GetPTT()->SetSpeakerVolume(100);
```

获取语音消息播放音量

此接口用于获取离线语音播放音量。返回值为 int 类型，值域为0 – 200。

接口原型

```
ITMGPTT virtual int GetSpeakerVolume()
```

示例代码

```
ITMGContextGetInstance()->GetPTT()->GetSpeakerVolume();
```

高级 API

获取版本号

获取 SDK 版本号，用于分析 SDK 使用情况。

接口原型

```
ITMGContext virtual const char* GetSDKVersion()
```

示例代码

```
ITMGContextGetInstance()->GetSDKVersion();
```

设置打印日志等级

用于设置打印日志等级。建议保持默认等级。需要在 Init 之前调用。

接口原型

```
ITMGContext int SetLogLevel(ITMG_LOG_LEVEL levelWrite, ITMG_LOG_LEVEL levelPrint)
```

参数含义

参数	类型	含义
levelWrite	ITMG_LOG_LEVEL	设置写入日志的等级，TMG_LOG_LEVEL_NONE 表示不写入，默认为 TMG_LOG_LEVEL_INFO
levelPrint	ITMG_LOG_LEVEL	设置打印日志的等级，TMG_LOG_LEVEL_NONE 表示不打印，默认为 TMG_LOG_LEVEL_ERROR

ITMG_LOG_LEVEL 说明如下：

ITMG_LOG_LEVEL	含义
TMG_LOG_LEVEL_NONE	不打印日志
TMG_LOG_LEVEL_ERROR	打印错误日志（默认）
TMG_LOG_LEVEL_INFO	打印提示日志
TMG_LOG_LEVEL_DEBUG	打印开发调试日志
TMG_LOG_LEVEL_VERBOSE	打印高频日志

示例代码

```
ITMGContextGetInstance()->SetLogLevel(TMG_LOG_LEVEL_INFO,TMG_LOG_LEVEL_INFO);
```

设置打印日志路径

用于设置打印日志路径。默认路径如下。需要在 Init 之前调用。

平台	路径
Windows	%appdata%\Tencent\GME\ProcessName
iOS	Application/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/Documents
Android	/sdcard/Android/data/xxx.xxx.xxx/files
Mac	/Users/username/Library/Containers/xxx.xxx.xxx/Data/Documents

接口原型

```
ITMGContext virtual int SetLogPath(const char* logDir)
```

参数	类型	含义
logDir	const char*	路径

示例代码

```
cosnt char* logDir = ""//自行设置路径
ITMGContext* context = ITMGContextGetInstance();
context->SetLogPath(logDir);
```

回调消息

消息	含义	Data	例子
ITMG_MAIN_EVENT_TYPE_ENTER_ROOM	进入音频房间消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_EXIT_ROOM	退出音频房间消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_ROOM_DISCONNECT	房间因为网络等原因断开消息	result; error_info	{"error_info":"waiting timeout, please check your network","result":0}
ITMG_MAIN_EVNET_TYPE_USER_UPDATE	房间成员更新消息	user_list; event_id	{"event_id":1,"user_list":["0"]}
ITMG_MAIN_EVENT_TYPE_RECONNECT_START	房间重连开始消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_RECONNECT_SUCCESS	房间重连成功消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_SWITCH_ROOM	快速切换房间消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE	房间状态切换消息	result; error_info; sub_event_type; new_room_type	{"error_info":"","new_room_type":0,"result":0}
ITMG_MAIN_EVENT_TYPE_ROOM_SHARING_START	开始跨房连麦消息	result;	{"result":0}
ITMG_MAIN_EVENT_TYPE_ROOM_SHARING_STOP	跨房连麦停止消息	result;	{"result":0}
ITMG_MAIN_EVENT_TYPE_SPEAKER_DEF	默认扬声器设备修	result; error_info	{"deviceID":"{0.0.0.00000000}.{a4f1e8be-49fa-43e2-b8cf-dd00542b47ae}","deviceName":"扬声器 (Realtek

AULT_DEVICE_CHANGED	改消息		High Definition Audio)","error_info":"","isNewDevice":true,"isUsedDevice":false,"result":0}
ITMG_MAIN_EVENT_TYPE_SPEAKER_NEW_DEVICE	新增扬声器设备消息	result; error_info	{"deviceID":"{0.0.0.00000000}.{a4f1e8be-49fa-43e2-b8cf-dd00542b47ae}","deviceName":"扬声器 (Realtek High Definition Audio)","error_info":"","isNewDevice":true,"isUsedDevice":false,"result":0}
ITMG_MAIN_EVENT_TYPE_SPEAKER_LOST_DEVICE	丢失扬声器设备消息	result; error_info	{"deviceID":"{0.0.0.00000000}.{a4f1e8be-49fa-43e2-b8cf-dd00542b47ae}","deviceName":"扬声器 (Realtek High Definition Audio)","error_info":"","isNewDevice":false,"isUsedDevice":false,"result":0}
ITMG_MAIN_EVENT_TYPE_MIC_NEW_DEVICE	新增麦克风设备消息	result; error_info	{"deviceID":"{0.0.1.00000000}.{5fdf1a5b-f42d-4ab2-890a-7e454093f229}","deviceName":"麦克风 (Realtek High Definition Audio)","error_info":"","isNewDevice":true,"isUsedDevice":true,"result":0}
ITMG_MAIN_EVENT_TYPE_MIC_LOST_DEVICE	丢失麦克风设备消息	result; error_info	{"deviceID":"{0.0.1.00000000}.{5fdf1a5b-f42d-4ab2-890a-7e454093f229}","deviceName":"麦克风 (Realtek High Definition Audio)","error_info":"","isNewDevice":false,"isUsedDevice":true,"result":0}
ITMG_MAIN_EVENT_TYPE_MIC_DEFAULT_DEVICE_CHANGED	默认麦克风设备修改消息	result; error_info	{"deviceID":"{0.0.1.00000000}.{5fdf1a5b-f42d-4ab2-890a-7e454093f229}","deviceName":"麦克风 (Realtek High Definition Audio)","error_info":"","isNewDevice":false,"isUsedDevice":true,"result":0}
ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_QUALITY	房间质量消息	weight; loss; delay	{"weight":5,"loss":0.1,"delay":1}
ITMG_MAIN_EVNET_TYPE_PTT_RECORD_COMPLETE	语音消息录制完成消息	result; file_path	{"file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_UPLOAD_COMPLETE	语音消息上传完成消息	result; file_path;file_id	{"file_id":"","file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_DOWNLOAD_COMPLETE	语音消息下载完成消息	result; file_path;file_id	{"file_id":"","file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_PLAY_COMPLETE	语音消息播放完成消息	result; file_path	{"file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_SPEECH2TEXT_COMPLETE	语音消息极速转文本完成消息	result; text;file_id	{"file_id":"","text":"","result":0}

ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_COMPLETE	语音消息流式转文本完成消息	result;file_path;text;file_id	{"file_id":"","file_path":","text":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_IS_RUNNING	语音消息正在流式转文本中	result;file_path;text;file_id	{"file_id":"","file_path":","text":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_TEXT2SPEECH_COMPLETE	文本转语音完成消息	result;text;file_id	{"file_id":"","text":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_TRANSLATE_TEXT_COMPLETE	文本翻译完成消息	result;text;file_id	{"file_id":"","text":"","result":0}

版权所有：腾讯云计算（北京）有限责任公司

Cocos2D SDK 工程配置

最近更新时间：2023-03-20 15:36:29

为方便 Cocos2D 开发者调试和接入腾讯云游戏多媒体引擎产品 API，本文档主要为您介绍适用于 Cocos2D 开发的工程配置指引。

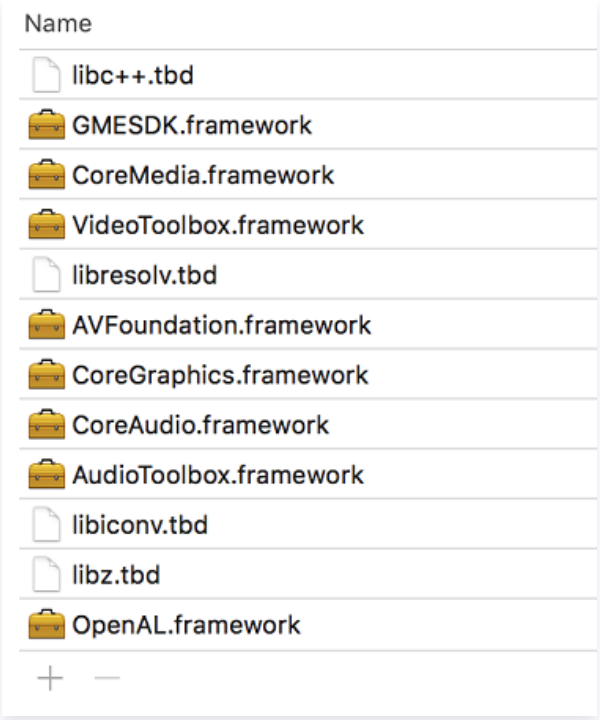
SDK 准备

1. 请下载相关 Demo 及 SDK，详情请参见 下载指引 。
2. 解压获取到的 SDK 资源。
3. 文件夹内容如下：
   - GMESDK：游戏音视频 SDK framework 文件。
   - GMECocosDemo：游戏音视频 SDK 示例工程。

> **说明**
> SDK 支持在 Mac 系统上编译。

iOS Xcode 配置

1. 将 framework 添加到 Xcode 工程中并设置头文件引用位置。（GMESDK 文件夹中的 framework 文件，必须添加到工程中）
2. 添加依赖库，参考下图进行：



Android 配置

1. 将 gmesdk.jar 加入到 libs 库中。
2. 在 Activity 中导入 so 文件。示例如下：

```
public class AppActivity extends Cocos2dxActivity {
 static final String TAG = "AppActivity";
 static OpensdkGameWrapper gameWrapper ;
```

```
 static {
     OpensdkGameWrapper.loadSdkLibrary();
 }
}
```

3. 在 oncreate 函数中进行初始化，顺序不能出错。

```
protected void onCreate(Bundle savedInstanceState) {
     super.setEnableVirtualButton(false);
     super.onCreate(savedInstanceState);
     //初始化， 顺序不能错
     gameWrapper = new OpensdkGameWrapper(this);
     runOnGLThread(new Runnable() {
         @Override
         public void run() {
             gameWrapper.initOpensdk();
         }
     });
}
```

4. 将自己的工程进行编译选项配置。请参考 GME Cocos Demo 中的 Android.mk。
   - 路径：GMECocos/GMECocosDemo/proj.android-studio/app/jni/Android.mk。
   - preBuild.mk 文件的路径：/Users/username/Downloads/GMECocos/GMESDK/android/bin/preBuild.mk。

导出不同平台

从 Cocos2D 引擎导出不同平台，需要做相应的工程配置。

Android 平台

配置必要权限

在工程 AndroidManifest.xml 文件中添加以下权限：

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

如果使用离线语音，请在清单文件 application 节点下添加：

```
<application android:usesCleartextTraffic="true" >
```

根据需求添加权限

请根据需求在工程 AndroidManifest.xml 文件中添加以下权限：

读写权限
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。

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

蓝牙权限

需要根据以下规则判断如何添加蓝牙权限：

- **如果工程中的 targetSDKVersion 为 30 或者以下：**

```
<uses-permission android:name="android.permission.BLUETOOTH"/>
```

- **如果工程中的 targetSDKVersion 为 31 或者以上：**

```
<uses-permission android:name="android.permission.BLUETOOTH" android:maxSdkVersion="30"
/>
<uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />
```

iOS 平台

添加权限：

- Microphone Usage Description：允许麦克风权限。
- 增加 Allow Arbitrary Loads 权限。如下图所示：

Key	Type	Value
▼Information Property List	Dictionary	(19 items)
Localization native development re...	String	$(DEVELOPMENT_LANGUAGE)
Executable file	String	$(EXECUTABLE_NAME)
Bundle identifier	String	$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Bundle name	String	$(PRODUCT_NAME)
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.0
Bundle version	String	1
Application requires iPhone enviro...	Boolean	YES
▼App Transport Security Settings	Dictionary	(1 item)
Allow Arbitrary Loads	Boolean	YES
Privacy - Camera Usage Description	String	Privacy - Camera Usage Description
Privacy - Microphone Usage Desc...	String	NSMicrophoneUsageDescription
▶Required background modes	Array	(1 item)
Application supports iTunes file sh...	Boolean	YES
Launch screen interface file base...	String	LaunchScreen
Main storyboard file base name	String	Main
▶Required device capabilities	Array	(1 item)
▶Supported interface orientations	Array	(3 items)
▶Supported interface orientations (i...	Array	(4 items)

Windows 平台

需要到 下载指引 中下载 Windows SDK 后导入到工程中。

快速入门

最近更新时间：2023-03-20 15:36:29

为方便 Cocos2D 开发者调试和接入腾讯云游戏多媒体引擎产品 API，这里向您介绍适用于 Cocos2D 开发的快速接入文档。
GME 快速入门文档只提供最主要的接入接口，更多详细接口请参考 相关接口文档 。

重要接口	接口含义
Init	初始化 GME
Poll	触发事件回调
EnterRoom	进房
EnableMic	开麦克风
EnableSpeaker	开扬声器

说明

- GME 使用前请对工程进行配置，否则 SDK 不生效。
- GME 的接口调用成功后返回值为 AV_OK，数值为 0。
- GME 的接口调用要在同一个线程下。
- GME 需要周期性的调用 Poll 接口触发事件回调。
- GME 回调信息参考回调消息列表。
- 设备的操作要在进房成功之后。
- 错误码详情可参考 错误码 。

快速接入步骤

1、获取单例

在使用语音功能时，需要首先获取 ITMGContext 对象。

示例代码

```
ITMGContext* context = ITMGContextGetInstance();
context->SetTMGDelegate(this);
```

2、初始化 SDK

参数获取请查看 接入指引 。
此接口需要来自腾讯云控制台的 AppID 号码作为参数，再加上 openID，这个 openID 是唯一标识一个用户，规则由 App 开发者自行制定，App 内不重复即可（目前只支持 INT64）。

注意
初始化 SDK 之后才可以进房。

函数原型

```
ITMGContext virtual int Init(const char* sdkAppId, const char* openID)
```

参数	类型	含义
sdkAppId	char*	来自腾讯云控制台的 AppId 号码。
openId	char*	OpenID 只支持 Int64 类型（转为 char* 传入），数值必须大于 10000，用于标识用户

示例代码

```
#define SDKAPPID3RD "1400089356"
cosnt char* openId="10001";
ITMGContext* context = ITMGContextGetInstance();
context->Init(SDKAPPID3RD, openId);
```

3、触发事件回调

通过在 update 里面周期的调用 Poll 可以触发事件回调。

函数原型

```
class ITMGContext {
protected:
    virtual ~ITMGContext() {}

public:
    virtual void Poll()= 0;
}
```

示例代码

```
ITMGContextGetInstance()->Poll();
```

4、鉴权信息

生成 AuthBuffer，用于相关功能的加密和鉴权，如正式发布请使用后台部署密钥，后台部署请参考 鉴权密钥 。离线语音获取鉴权时，房间号参数必须填 null。

函数原型

```
int  QAVSDK_AuthBuffer_GenAuthBuffer(unsigned int dwSdkAppID, const char* strRoomID, const char*
strOpenID,
    const char* strKey, unsigned char* strAuthBuffer, unsigned int bufferLength);
```

参数	类型	含义
dwSdkAppID	int	来自腾讯云控制台的 AppId 号码。
strRoomID	char*	房间号，最大支持127字符（离线语音房间号参数必须填 null）
strOpenID	char*	用户标识。与 Init 时候的 openID相同。
strKey	char*	来自腾讯云 控制台 的权限密钥。
strAuthBuffer	char*	返回的 authbuff
bufferLength	int	传入的 authbuff 长度，建议为 500

示例代码

```
unsigned int bufferLen = 512;
unsigned char retAuthBuff[512] = {0};
QAVSDK_AuthBuffer_GenAuthBuffer(atoi(SDKAPPID3RD), roomId, "10001",
AUTHKEY,retAuthBuff,bufferLen);
```

5、加入房间

用生成的鉴权信息进房，会收到消息为 ITMG_MAIN_EVENT_TYPE_ENTER_ROOM 的回调。加入房间默认不打开麦克风及扬声器。返回值为 AV_OK 的时候代表成功。
如果普通语音进房，业务方面无涉及范围语音需求，则使用普通进房接口。详细信息请查看 范围语音 。

函数原型

```
ITMGContext virtual int EnterRoom(const char*  roomID, ITMG_ROOM_TYPE roomType, const char*
authBuff, int buffLen)
```

参数	类型	含义
roomID	char*	房间号，最大支持127字符
roomType	ITMG_ROOM_TYPE	房间音频类型
authBuffer	char*	鉴权码
buffLen	int	鉴权码长度

房间音频类型请参考 音质选择 。

示例代码

```
ITMGContext* context = ITMGContextGetInstance();
context->EnterRoom(roomID, ITMG_ROOM_TYPE_STANDARD, (char*)retAuthBuff,bufferLen);
```

6、加入房间事件的回调

加入房间完成后会发送信息 ITMG_MAIN_EVENT_TYPE_ENTER_ROOM，在 OnEvent 函数中进行判断。

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;
        }
    }
}
```

7、开启关闭麦克风

此接口用来开启关闭麦克风。加入房间默认不打开麦克风及扬声器。

函数原型

```
ITMGAudioCtrl virtual int EnableMic(bool bEnabled)
```

参数	类型	含义
bEnabled	bool	如果需要打开麦克风，则传入的参数为 true，如果关闭麦克风，则参数为 false

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->EnableMic(true);
```

8、开启关闭扬声器

此接口用于开启关闭扬声器。

函数原型

```
ITMGAudioCtrl virtual int EnableSpeaker(bool enabled)
```

参数	类型	含义
enable	bool	如果需要关闭扬声器，则传入的参数为 false，如果打开扬声器，则参数为 true

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->EnableSpeaker(true);
```

实时语音

最近更新时间：2023-03-20 15:36:29

为方便 Cocos2d 开发者调试和接入腾讯云游戏多媒体引擎客户端 API，本文为您介绍适用于 Cocos2d 实时语音功能的开发接入技术文档。

使用 GME 重要事项

GME 提供实时语音服务、语音消息服务及转文本服务，使用 GME 服务都依赖 Init 和 Poll 等核心接口。

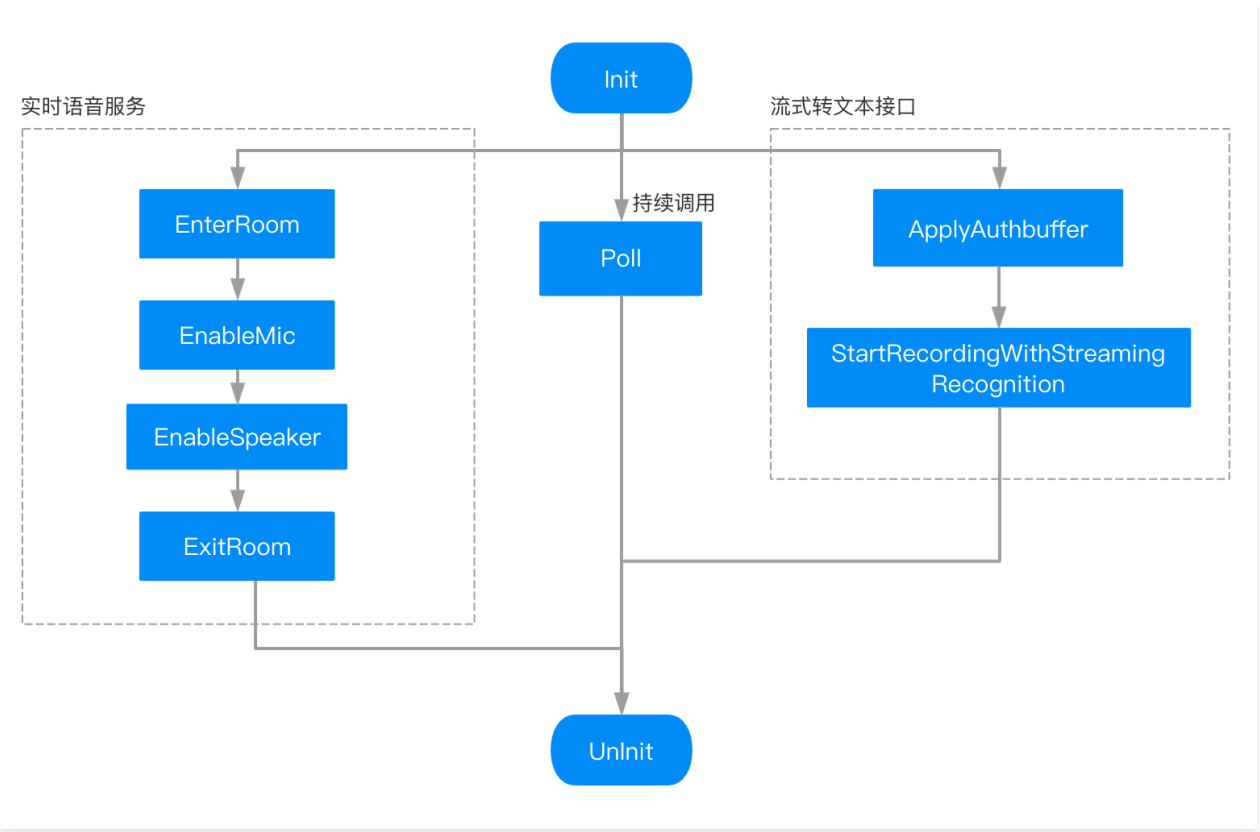
重点提示

- 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、语音消息服务以及转文本服务**。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。
- GME 的接口调用成功后返回值为 QAVError.OK，数值为 0。
- GME 的接口调用要在同一个线程下。
- GME 需要周期性的调用 Poll 接口触发事件回调。
- 错误码详情可参考 错误码 。

接入 SDK

重要步骤

接入 SDK 重要流程如下：



1. 初始化 GME
2. 周期性调用 Poll 触发回调
3. 进入实时语音房间
4. 打开麦克风

5. 打开扬声器
6. 退出语音房间
7. 反初始化 GME

C++ 类

类	含义
ITMGContext	核心接口
ITMGRoom	房间相关接口
ITMGRoomManager	房间管理接口
ITMGAudioCtrl	音频相关接口
ITMGAudioEffectCtrl	音效及伴奏相关接口

核心接口

接口	接口含义
Init	初始化 GME
Poll	触发事件回调
Pause	系统暂停
Resume	系统恢复
Uninit	反初始化 GME

准备工作

接入 GME 首先需要引入头文件 tmg_sdk.h，头文件类继承 ITMGDelegate 以进行消息的传递及回调。

示例代码

```
#include "tmg_sdk.h"

class TMGTestScene : public cocos2d::Scene,public ITMGDelegate
{
public:
...
private:
...
}
```

设置单例

在 EnterRoom 函数调用之前要先获取 ITMGContext ，所有调用都从 ITMGContext 开始，由ITMGDelegate 回调回传给应用，必须首先设置。

示例代码

```
ITMGContext* context = ITMGContextGetInstance();
context->SetTMGDelegate(this);
```

消息传递

接口类采用 Delegate 方法用于向应用程序发送回调通知，消息类型参考 ITMG_MAIN_EVENT_TYPE，data 在 Windows 平台下是 json 字符串格式， 具体 key-value 参见说明文档。

示例代码

```
//函数实现：
//TMGTestScene.h:
class TMGTestScene : public cocos2d::Scene,public ITMGDelegate
{
public:
    void OnEvent(ITMG_MAIN_EVENT_TYPE eventType, const char* data);
}

//TMGTestScene.cpp:
void TMGTestScene::OnEvent(ITMG_MAIN_EVENT_TYPE eventType,const char* data){
    //在此对eventType进行判断及操作
}
```

初始化 SDK

未初始化前，SDK 处于未初始化阶段，**需要通过接口 Init 初始化 SDK**，才可以使用实时语音服务、语音消息服务及转文本服务。调用 Init 接口的线程必须于其他接口在同一线程,建议都在主线程调用接口。

接口原型

```
ITMGContext virtual int Init(const char* sdkAppId, const char* openId)
```

参数	类型	含义
sdkAppId	const char*	来自 腾讯云控制台 的 GME 服务提供的 AppID，获取请参考 语音服务开通指引 。
openID	const char*	openID 只支持 Int64 类型（转为 const char* 传入）。规则由 App 开发者自行制定，App 内不重复即可。如需使用字符串作为 Openid 传入，可 提交工单 联系开发者。

返回值

返回值	处理
AV_OK = 0	初始化 SDK 成功
AV_ERR_SDK_NOT_FULL_UPDATE=7015	检查 SDK 文件是否完整，建议删除后重新导入 SDK

⚠ 关于7015错误提示

- 7015错误码是通过 md5 进行判断，在接入过程中若出现此错误，请根据提示检查 SDK 文件是否完整、SDK 文件版本是否一致。
- 出现返回值 AV_ERR_SDK_NOT_FULL_UPDATE 时，此返回值**只有提示作用**，并不会造成初始化失败。
- 由于第三方加固、Unity 打包机制等因素会影响库文件 md5，造成误判，所以**正式发布请在逻辑中忽略此错误**，并尽量不在 UI 中提示。

示例代码

```
#define SDKAPPID3RD "14000xxxxx"
cosnt char* openId="10001";
ITMGContext* context = ITMGContextGetInstance();
context->Init(SDKAPPID3RD, openId);
```

触发事件回调

通过在 update 里面周期的调用 Poll 可以触发事件回调。Poll 是 GME 的消息泵，GME 需要周期性的调用 Poll 接口触发事件回调。如果没有调用 Poll ，将会导致整个 SDK 服务运行异常。
可参考 Demo 中的 EnginePollHelper.cpp 文件。

> **务必周期性调用 Poll 接口**
> 务必周期性调用 Poll 接口且在主线程调用，以免接口回调异常。

接口原型

```
class ITMGContext {
protected:
    virtual ~ITMGContext() {}

public:
    virtual void Poll()= 0;
}
```

示例代码

```
//头文件中的声明
class TMGTestScene : public cocos2d::Scene,public ITMGDelegate
{
void update(float delta);
}

//代码实现
void TMGTestScene::update(float delta)
{
    ITMGContextGetInstance()->Poll();
}
```

系统暂停

当系统发生 Pause 事件时，需要同时通知引擎进行 Pause。如果不需要后台播放房间内声音，请调用 Pause 接口暂停整个 GME 服务。

接口原型

```
ITMGContext int Pause()
```

系统恢复

当系统发生 Resume 事件时，需要同时通知引擎进行 Resume。Resume 接口只恢复实时语音。

接口原型

```
ITMGContext int Resume()
```

反初始化 SDK

反初始化 SDK，进入未初始化状态。**如果游戏业务侧账号与 openid 是绑定的，那切换游戏账号需要反初始化 GME，再用新的 openid 初始化。**

接口原型

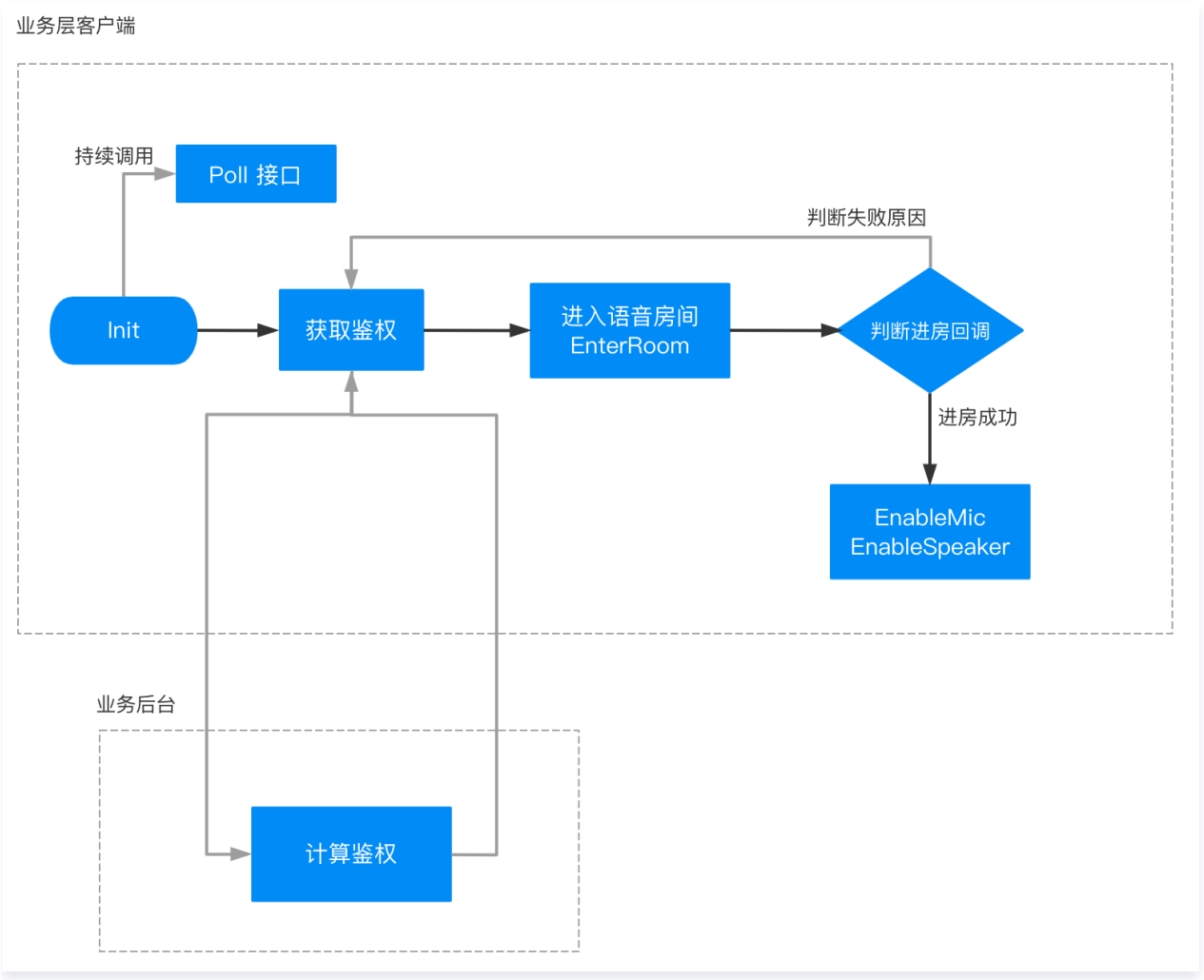
```
ITMGContext int Uninit()
```

示例代码

```
ITMGContext* context = ITMGContextGetInstance();
context->Uninit();
```

实时语音房间相关接口

初始化之后，SDK 调用进房后进去了房间，才可以进行实时语音通话。
使用问题可参考 实时语音相关问题 。



接口	接口含义

GenAuthBuffer	本地鉴权计算
EnterRoom	加入房间
ExitRoom	退出房间
IsRoomEntered	判断是否已经进入房间
SwitchRoom	快速切换房间

本地鉴权计算

生成 AuthBuffer，用于相关功能的加密和鉴权，如正式发布请使用后台部署密钥，后台部署请参考 鉴权密钥 。

接口原型

```
int  QAVSDK_AuthBuffer_GenAuthBuffer(unsigned int dwSdkAppID, const char* strRoomID, const char*
strOpenID,
    const char* strKey, unsigned char* strAuthBuffer, unsigned int bufferLength);
```

参数	类型	含义
dwSdkAppID	unsigned int	来自腾讯云控制台的 AppId 号码
strRoomID	const char*	房间号，最大支持127字符
strOpenID	const char*	用户标识。与 Init 时候的 openID相同。
strKey	const char*	来自腾讯云 控制台 的权限密钥
strAuthBuffer	const char*	返回的 authbuff
bufferLength	int	传入的 authbuff 长度，建议为 500

示例代码

```
unsigned int bufferLen = 512;
unsigned char retAuthBuff[512] = {0};
QAVSDK_AuthBuffer_GenAuthBuffer(atoi(SDKAPPID3RD), roomId, "10001",
AUTHKEY,retAuthBuff,bufferLen);
```

加入房间

用生成的鉴权信息进房，加入房间默认不打开麦克风及扬声器。

注意

- 加入房间事件回调结果 result 为 0 代表进房成功，进房接口 EnterRoom 返回值为 0 不代表进房成功。
- 房间的音频类型由第一个进房的人确定，此后房间里有成员修改房间类型，将对此房间所有成员生效。例如第一个进入房间的人使用的房间音频类型是流畅音质，第二个进房的是即使进房时候调用接口的音频类型参数是高清音质，进入房间之后也会变成流畅音质。需要有成员调用 ChangeRoomType 才会修改房间的音频类型。

接口原型

```
ITMGContext virtual int EnterRoom(const char*  roomID, ITMG_ROOM_TYPE roomType, const char*
```

```
authBuff, int buffLen)
```

参数	类型	含义
roomID	const char*	房间号，最大支持127字符
roomType	ITMG_ROOM_TYPE	房间类型，游戏建议使用 ITMG_ROOM_TYPE_FLUENCY。房间音频类型请参考 音质选择 。
authBuffer	const char*	鉴权码
buffLen	int	鉴权码长度

示例代码

```
ITMGContext* context = ITMGContextGetInstance();
context->EnterRoom(roomID, ITMG_ROOM_TYPE_FLUENCY, (char*)retAuthBuff,bufferLen);
```

加入房间事件回调

加入房间完成后会发送信息 ITMG_MAIN_EVENT_TYPE_ENTER_ROOM，在 OnEvent 函数中进行判断回调后处理。如果回调为成功，即此时进房成功，开始进行**计费**。

> **计费问题参考：**
> - 购买指南。
> - 计费相关问题。
> - 使用实时语音后，如果客户端掉线了，是否还会继续计费？

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;
        }
    }
}
```

Data 详情

消息	Data	例子
ITMG_MAIN_EVENT_TYPE_ENTER_ROOM	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_ROOM_DISCONNECT	result; error_info	{"error_info":"waiting timeout, please check your network","result":0}

如果断网，将会有断网的回调提示 `ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVENT_TYPE_ROOM_DISCONNECT`，此时 SDK 会自动进行重连，回调是 `ITMG_MAIN_EVENT_TYPE_RECONNECT_START`，当重连成功时，会有 `ITMG_MAIN_EVENT_TYPE_RECONNECT_SUCCESS` 回调。

错误码

错误码值	原因及建议方案
7006	鉴权失败原因。 • AppID 不存在或者错误 • authbuff 鉴权错误 • 鉴权过期 • OpenId 不符合规范
7007	已经在其它房间
1001	已经在进房过程中，然后又重复了此操作。建议在进房回调返回之前不要再调用进房接口
1003	已经进房了在房间中，又调用一次进房接口
1101	确保已经初始化 SDK，确保 OpenId 是否符合规则，或者确保在同一线程调用接口，以及确保 Poll 接口正常调用

退出房间

通过调用此接口可以退出所在房间。这是一个异步接口，返回值为 AV_OK 的时候代表异步投递成功。如果应用中有退房后立即进房的场景，在接口调用流程上，开发者无需要等待 ExitRoom 的回调 RoomExitComplete 通知，只需直接调用接口。

接口原型

```
ITMGContext virtual int ExitRoom()
```

示例代码

```
ITMGContext* context = ITMGContextGetInstance();
context->ExitRoom();
```

退出房间事件回调

退出房间完成后会有回调，消息为 ITMG_MAIN_EVENT_TYPE_EXIT_ROOM。

示例代码

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

Data详情

消息	Data	例子
ITMG_MAIN_EVENT_TYPE_EXIT_ROOM	result; error_info	{"error_info":"","result":0}

判断是否已经进入房间

通过调用此接口可以判断是否已经进入房间，返回值为 bool 类型。请勿在进房过程中调用。

接口原型

```
ITMGContext virtual bool IsRoomEntered()
```

示例代码

```
ITMGContext* context = ITMGContextGetInstance();
context->IsRoomEntered();
```

快速切换房间

调用此接口快速切换实时语音房间。此接口在进房后调用。切换房间后，不重置设备，即如果在此房间已经是打开麦克风状态，在切换房间后也会是打开麦克风状态。
快速切换房间的回调是 ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVENT_TYPE_SWITCH_ROOM，字段是 error_info 以及 result。

接口原型

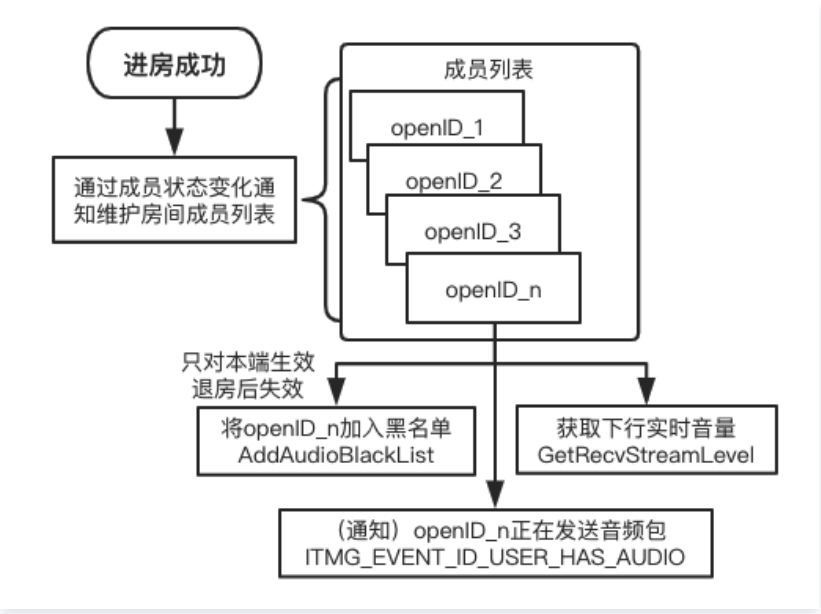
```
ITMGContext virtual int SwitchRoom(const char* targetRoomID, const char* authBuff, int buffLen);
```

类型说明

参数	类型	含义
targetRoomID	const char*	将要进入的房间号
authBuffer	const char*	用将要进入的房间号生成的新鉴权
buffLen	int	鉴权码长度

房间内状态维护

此部分接口用于业务层显示说话成员、进退房成员，以及将房间内某成员禁言等功能。



接口/通知	含义
ITMG_MAIN_EVNET_TYPE_USER_UPDATE	成员状态变化通知
AddAudioBlackList	房间中禁言某成员
RemoveAudioBlackList	移除禁言

成员进房、说话状态通知事件

此接口适用于获取房间中说话的人并在 UI 中展示，以及有人进入、退出语音房间的一个通知。
该事件在状态变化才通知，状态不变化的情况下不通知。如需实时获取成员状态，请在上层收到通知时缓存，事件消息为 ITMG_MAIN_EVNET_TYPE_USER_UPDATE，其中 data 包含两个信息，event_id 及 user_list，在 OnEvent 函数中对事件消息进行判断。
音频事件的通知有一个阈值，超过这个阈值才会发送通知。超过两秒没有收到音频包才通知“有成员停止发送音频包”消息。

event_id	含义	应用侧维护内容
ITMG_EVENT_ID_USER_ENTER	有成员进入房间，返回此时进房的 openid	应用侧维护成员列表
ITMG_EVENT_ID_USER_EXIT	有成员退出房间，返回此时退房的 openid	应用侧维护成员列表
ITMG_EVENT_ID_USER_HAS_AUDIO	有成员发送音频包，返回此时房间内说话的 openid，通过此事件可以判断用户是否说话，并展示声纹效果	应用侧维护通话成员列表
ITMG_EVENT_ID_USER_NO_AUDIO	有成员停止发送音频包，返回此时房间内停止说话的 openid	应用侧维护通话成员列表

示例代码

```
void TMGTestScene::OnEvent(ITMG_MAIN_EVENT_TYPE eventType,const char* data){
    switch (eventType) {
            case ITMG_MAIN_EVNET_TYPE_USER_UPDATE:
        {
        //进行处理
        //开发者对参数进行解析，得到信息 eventID 及 user_list
            switch (eventID)
             {
             case ITMG_EVENT_ID_USER_ENTER:
                  //有成员进入房间
                 break;
             case ITMG_EVENT_ID_USER_EXIT:
                  //有成员退出房间
                break;
            case ITMG_EVENT_ID_USER_HAS_AUDIO:
                //有成员发送音频包
                break;
            case ITMG_EVENT_ID_USER_NO_AUDIO:
                //有成员停止发送音频包
                break;
             default:
                break;
            }
        break;
        }
```

```
    }
}
```

房间中禁言某成员

将某个 ID 加入音频数据黑名单，即不接受某人的语音， 只对本端生效，不会影响其他端。返回值为 0 表示调用成功。例如： A、B、C 都在同一个房间开麦说话：

- 如果 A 设置了 C 的黑名单， 则 A 只能听见 B 的声音。
- B 因为没有设置黑名单， 仍旧可以听见 A 和 C 的声音。
- C 同样因为没有设置黑名单， 可以听见 A 和 B 的声音。

此接口适用于在语音房间中将某用户禁言的场景。

接口原型

```
ITMGContext ITMGAudioCtrl int AddAudioBlackList(const char* openId)
```

参数	类型	含义
openId	char*	需添加黑名单的用户 openid

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->AddAudioBlackList(openId);
```

移除禁言

将某个 Id 移除音频数据黑名单。返回值为0表示调用成功。

接口原型

```
ITMGContext ITMGAudioCtrl int RemoveAudioBlackList(const char* openId)
```

参数	类型	含义
openId	char*	需移除黑名单的 ID

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->RemoveAudioBlackList(openId);
```

实时语音采集相关接口

- 初始化 SDK 之后进房，在房间中，才可以调用实时音频语音相关接口。
- 当用户界面单击打开/关闭麦克风/扬声器按钮时，建议采用 EnableMic 以及 EnableSpeaker 接口进行调用。
- 当用户界面按住麦克风按钮时发言，放开按钮不发言，建议采用进房时候调用 EnableAudioCaptureDevice 一次，后续按住发言调用 EnableAudioSend 来实现。

接口	接口含义
EnableMic	开关麦克风
GetMicState	获取麦克风状态

EnableAudioCaptureDevice	开关采集设备
IsAudioCaptureDeviceEnabled	获取采集设备状态
EnableAudioSend	打开关闭音频上行
IsAudioSendEnabled	获取音频上行状态
GetMicLevel	获取实时麦克风音量
GetSendStreamLevel	获取音频上行实时音量
SetMicVolume	设置麦克风音量
GetMicVolume	获取麦克风音量

开启或关闭麦克风

此接口用来开启关闭麦克风。加入房间默认不打开麦克风及扬声器。EnableMic = EnableAudioCaptureDevice + EnableAudioSend

接口原型

```
ITMGAudioCtrl virtual int EnableMic(bool bEnabled)
```

参数	类型	含义
bEnabled	bool	如果需要打开麦克风，则传入的参数为 true，如果关闭麦克风，则参数为 false

示例代码

```
//打开麦克风
ITMGContextGetInstance()->GetAudioCtrl()->EnableMic(true);
```

麦克风状态获取

此接口用于获取麦克风状态，返回值0为关闭麦克风状态，返回值1为打开麦克风状态。

接口原型

```
ITMGAudioCtrl virtual int GetMicState()
```

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->GetMicState();
```

开启或关闭采集设备

此接口用来开启/关闭采集设备。加入房间默认不打开设备。

- 只能在进房后调用此接口，退房会自动关闭设备。
- 在移动端，打开采集设备通常会伴随权限申请，音量类型调整等操作。

接口原型

```
ITMGAudioCtrl virtual int EnableAudioCaptureDevice(bool enable)
```

参数	类型	含义
enable	bool	如果需要打开采集设备，则传入的参数为 true，如果关闭采集设备，则参数为 false

示例代码

```
//打开采集设备
ITMGContextGetInstance()->GetAudioCtrl()->EnableAudioCaptureDevice(true);
```

采集设备状态获取

此接口用于采集设备状态获取。

接口原型

```
ITMGContext virtual bool IsAudioCaptureDeviceEnabled()
```

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->IsAudioCaptureDeviceEnabled();
```

打开或关闭音频上行

此接口用于打开/关闭音频上行。如果采集设备已经打开，那么会发送采集到的音频数据。如果采集设备没有打开，那么仍旧无声。采集设备的打开关闭请参见接口 EnableAudioCaptureDevice。

接口原型

```
ITMGContext  virtual int EnableAudioSend(bool bEnable)
```

参数	类型	含义
bEnable	bool	如果需要打开音频上行，则传入的参数为 true，如果关闭音频上行，则参数为 false

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->EnableAudioSend(true);
```

音频上行状态获取

此接口用于音频上行状态获取。

接口原型

```
ITMGContext virtual bool IsAudioSendEnabled()
```

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->IsAudioSendEnabled();
```

获取麦克风实时音量

此接口用于获取麦克风实时音量，返回值为 int 类型。建议20ms获取一次。值域为0 – 100，通过此接口可以获取到麦克风采集到的实时音量情况。

接口原型

```
ITMGAudioCtrl virtual int GetMicLevel()
```

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->GetMicLevel();
```

获取音频上行实时音量

此接口用于获取自己音频上行实时音量，返回值为 int 类型，取值范围为0 – 100。

接口原型

```
ITMGAudioCtrl virtual int GetSendStreamLevel()
```

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->GetSendStreamLevel();
```

设置麦克风软件音量

此接口用于设置麦克风的音量。参数 volume 用于设置麦克风的音量，相当于对采集的声音做衰减或增益。

接口原型

```
ITMGAudioCtrl virtual int SetMicVolume(int vol)
```

参数	类型	含义
vol	int	取值范围为 0–200，数值为0的时候表示静音，当数值为100的时候表示音量不增不减，默认数值为100。

示例代码

```
int micVol = (int)(value * 100);
ITMGContextGetInstance()->GetAudioCtrl()->SetMicVolume(vol);
```

获取麦克风软件音量

此接口用于获取麦克风的音量。返回值为一个int类型数值，返回值为101代表没调用过接口 SetMicVolume。

接口原型

```
ITMGAudioCtrl virtual int GetMicVolume()
```

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->GetMicVolume();
```

实时语音播放相关接口

接口	接口含义
EnableSpeaker	开关扬声器
GetSpeakerState	获取扬声器状态
EnableAudioPlayDevice	开关播放设备
IsAudioPlayDeviceEnabled	获取播放设备状态
EnableAudioRecv	打开关闭音频下行
IsAudioRecvEnabled	获取音频下行状态
GetSpeakerLevel	获取实时扬声器音量
GetRecvStreamLevel	获取房间内其他成员下行实时音量
SetSpeakerVolume	设置扬声器音量
GetSpeakerVolume	获取扬声器音量

开启或关闭扬声器

此接口用于开启关闭扬声器。**EnableSpeaker = EnableAudioPlayDevice + EnableAudioRecv**

接口原型

```
ITMGAudioCtrl virtual int EnableSpeaker(bool enable)
```

参数	类型	含义
enable	bool	如果需要关闭扬声器，则传入的参数为 false，如果打开扬声器，则参数为 true

示例代码

```
//打开扬声器
ITMGContextGetInstance()->GetAudioCtrl()->EnableSpeaker(true);
```

扬声器状态获取

此接口用于扬声器状态获取。返回值0为关闭扬声器状态，返回值1为打开扬声器状态。

接口原型

```
ITMGAudioCtrl virtual int GetSpeakerState()
```

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->GetSpeakerState();
```

开启或关闭播放设备

此接口用于开启关闭播放设备。

接口原型

```
ITMGAudioCtrl virtual int EnableAudioPlayDevice(bool enable)
```

参数	类型	含义
enable	bool	如果需要关闭播放设备，则传入的参数为 false，如果打开播放设备，则参数为 true

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->EnableAudioPlayDevice(true);
```

播放设备状态获取

此接口用于播放设备状态获取。

接口原型

```
ITMGAudioCtrl virtual bool IsAudioPlayDeviceEnabled()
```

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->IsAudioPlayDeviceEnabled();
```

打开或关闭音频下行

此接口用于打开/关闭音频下行。如果播放设备已经打开，那么会播放房间里其他人的音频数据。如果播放设备没有打开，那么仍旧无声。播放设备的打开关闭参见接口请参见 EnableAudioPlayDevice。

接口原型

```
ITMGAudioCtrl virtual int EnableAudioRecv(bool enable)
```

参数	类型	含义
enable	bool	如果需要打开音频下行，则传入的参数为 true，如果关闭音频下行，则参数为 false

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->EnableAudioRecv(true);
```

音频下行状态获取

此接口用于音频下行状态获取。

接口原型

```
ITMGAudioCtrl virtual bool IsAudioRecvEnabled()
```

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->IsAudioRecvEnabled();
```

获取扬声器实时音量

此接口用于获取扬声器实时音量。返回值为 int 类型数值，表示扬声器实时音量。建议20ms获取一次。

接口原型

```
ITMGAudioCtrl virtual int GetSpeakerLevel()
```

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->GetSpeakerLevel();
```

获取房间内其他成员下行实时音量

此接口用于获取房间内其他成员下行实时音量，返回值为 int 类型，取值范围为0 – 200。

接口原型

```
ITMGAudioCtrl virtual int GetRecvStreamLevel(const char* openId)
```

参数	类型	含义
openId	char*	房间其他成员的openId

示例代码

```
iter->second.level = ITMGContextGetInstance()->GetAudioCtrl()->GetRecvStreamLevel(iter-
>second.openid.c_str());
```

设置扬声器的音量

此接口用于设置扬声器的音量。

接口原型

```
ITMGAudioCtrl virtual int SetSpeakerVolume(int vol)
```

参数	类型	含义
vol	int	设置音量，范围0 – 200，当数值为0时，表示静音，当数值为100时，表示音量不增不减，默认数值为100。

示例代码

```
int vol = 100;
ITMGContextGetInstance()->GetAudioCtrl()->SetSpeakerVolume(vol);
```

获取扬声器的音量

此接口用于获取扬声器的音量。返回值为 int 类型数值，代表扬声器的音量，返回值为101代表没调用过接口 SetSpeakerVolume。
Level 是实时音量，Volume 是扬声器的音量，最终声音音量 = Level × Volume %。例如实时音量是数值是100，此时 Volume 的数值是

60，那么最终发出来的声音数值也是60。

接口原型

```
ITMGAudioCtrl virtual int GetSpeakerVolume()
```

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->GetSpeakerVolume();
```

设备选择相关接口

设备选择相关接口只能在 PC 端上使用。

接口	接口含义
GetMicListCount	获取麦克风设备数量
GetMicList	枚举麦克风设备
GetSpeakerListCount	获取扬声器设备数量
GetSpeakerList	枚举扬声器设备
SelectMic	选定麦克风设备
SelectSpeaker	选定扬声器设备

获取麦克风设备数量

此接口用来获取麦克风设备数量。

函数原型

```
ITMGAudioCtrl virtual int GetMicListCount()
```

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->GetMicListCount();
```

枚举麦克风设备

此接口用来枚举麦克风设备。配合 GetMicListCount 接口使用。

函数原型

```
ITMGAudioCtrl virtual int GetMicList(TMGAudioDeviceInfo* ppDeviceInfoList, int nCount)

class TMGAudioDeviceInfo
{
public:
    const char* pDeviceID;
    const char* pDeviceName;
};
```

参数	类型	含义
ppDeviceInfoList	TMGAudioDeviceInfo	设备列表
nCount	int	获取的麦克风设备数量

TMGAudioDeviceInfo 参数	类型	含义
pDeviceID	const char*	设备名称
pDeviceName	const char*	设备 ID

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->GetMicList(ppDeviceInfoList,nCount);
```

选中麦克风设备

此接口用来选中麦克风设备。如果不调用或者传入 "DEVICEID_DEFAULT"，则选中系统默认设备。
GetMicList接口中返回的第0个设备 id 为默认设备，未选中设备时通话设备为默认设备，选中后业务层维护通话设备。如果该通话设备被拔出，则此时通话设备为默认设备，拔出的通话设备插入后，此时通话设备恢复为插入的通话设备。

函数原型

```
ITMGAudioCtrl virtual int SelectMic(const char* pMicID)
```

参数	类型	含义
pMicID	const char*	麦克风设备 ID，设备 ID 来自于 GetMicList 返回列表。

示例代码

```
const char* pMicID ="{0.0.1.00000000}.{7b0b712d-3b46-4f7a-bb83-bf9be4047f0d}";
ITMGContextGetInstance()->GetAudioCtrl()->SelectMic(pMicID);
```

获取扬声器设备数量

此接口用来获取扬声器设备数量。

函数原型

```
ITMGAudioCtrl virtual int GetSpeakerListCount()
```

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->GetSpeakerListCount();
```

枚举扬声器设备

此接口用来枚举扬声器设备。配合 GetSpeakerListCount 接口使用。

函数原型

```
ITMGAudioCtrl virtual int GetSpeakerList(TMGAudioDeviceInfo* ppDeviceInfoList, int nCount)
```

参数	类型	含义
ppDeviceInfoList	TMGAudioDeviceInfo	设备列表
nCount	int	获取的扬声器设备数量

TMGAudioDeviceInfo 参数	类型	含义
pDeviceID	const char*	设备名称
pDeviceName	const char*	设备 ID

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->GetSpeakerList(ppDeviceInfoList,nCount);
```

选中扬声器设备

此接口用来选中播放设备。如果不调用或者传入 "DEVICEID_DEFAULT"，则选中系统默认播放设备。

函数原型

```
ITMGAudioCtrl virtual int SelectSpeaker(const char* pSpeakerID)
```

参数	类型	含义
pSpeakerID	const char*	扬声器设备 ID，设备 ID 来自于 GetSpeakerList 返回列表。

示例代码

```
const char* pSpeakerID ="{0.0.1.00000000}.{7b0b712d-3b46-4f7a-bb83-bf9be4047f0d}";
ITMGContextGetInstance()->GetAudioCtrl()->SelectSpeaker(pSpeakerID);
```

高级 API

启动耳返

此接口用于启动耳返，需要 EnableLoopBack+EnableSpeaker 才可以听到自己声音。

接口原型

```
ITMGAudioCtrl virtual int EnableLoopBack(bool enable)
```

参数	类型	含义

enable	bool	设置是否启动

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->EnableLoopBack(true);
```

获取用户房间音频类型

此接口用于获取用户房间音频类型，返回值为房间音频类型，返回值为0时代表获取用户房间音频类型发生错误，房间音频类型参考 EnterRoom 接口。

接口原型

```
class ITMGRoom {
public:
    virtual ~ITMGRoom() {} ;
    virtual int GetRoomType() = 0;

};
```

示例代码

```
ITMGContext* context = ITMGContextGetInstance();
ITMGContextGetInstance()->GetRoom()->GetRoomType();
```

房间类型修改

此接口用于修改用户房间音频类型，结果参见回调事件，事件类型为 ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE。房间的音频类型由第一个进房的人确定，此后房间里有成员修改房间类型，将对此房间所有成员生效。

接口原型

```
IITMGContext TMGRoom public int ChangeRoomType((ITMG_ROOM_TYPE roomType)
```

参数	类型	含义
roomType	ITMG_ROOM_TYPE	房间切换成的目标类型，房间音频类型参考 EnterRoom 接口

示例代码

```
ITMGContext* context = ITMGContextGetInstance();
ITMGContextGetInstance()->GetRoom()->ChangeRoomType(ITMG_ROOM_TYPE_FLUENCY);
```

Data 详情

消息	Data	例子
ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE	result; error_info; new_room_type	{"error_info":"","new_room_type":0,"result":0}

房间类型完成回调

房间类型设置完成后，回调的事件消息为 ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE，返回的参数为 result、error_info 及 new_room_type，new_room_type 代表的信息如下，在 OnEvent 函数中对事件消息进行判断。

事件子类型	代表参数	含义
ITMG_ROOM_CHANGE_EVENT_ENTERROOM	1	表示在进房的过程中，自带的音频类型与房间不符合，被修改为所进入房间的音频类型
ITMG_ROOM_CHANGE_EVENT_START	2	表示已经在房间内，音频类型开始切换（例如调用 ChangeRoomType 接口后切换音频类型 ）
ITMG_ROOM_CHANGE_EVENT_COMPLETE	3	表示已经在房间，音频类型切换完成
ITMG_ROOM_CHANGE_EVENT_REQUEST	4	表示房间成员调用 ChangeRoomType 接口，请求切换房间音频类型

示例代码

```
void TMGTestScene::OnEvent(ITMG_MAIN_EVENT_TYPE eventType,const char* data) {
    if (ITMGContext.ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE == type)
        {
        //对房间类型事件进行处理
     }
}
```

房间通话质量监控事件

质量监控事件，在进房后触发，2秒回调一次，事件消息为 ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_QUALITY，返回的参数为 weight、loss 及 delay，代表的信息如下：

参数	类型	含义
weight	int	范围是1 – 50，数值为50是音质评分极好，数值为1是音质评分很差，几乎不能使用，数值为0代表初始值，无含义。一般数值在30以下就可以提醒用户网络较差，建议切换网络。
loss	double	上行丢包率。
delay	int	音频触达延迟时间（ms）。

获取版本号

获取 SDK 版本号，用于分析。

接口原型

```
ITMGContext virtual const char* GetSDKVersion()
```

示例代码

```
ITMGContextGetInstance()->GetSDKVersion();
```

设置打印日志等级

用于设置打印日志等级。建议保持默认等级。需要在 Init 之前调用。

接口原型

```
ITMGContext int SetLogLevel(ITMG_LOG_LEVEL levelWrite, ITMG_LOG_LEVEL levelPrint)
```

参数含义

参数	类型	含义
levelWrite	ITMG_LOG_LEVEL	设置写入日志的等级，TMG_LOG_LEVEL_NONE 表示不写入，默认为 TMG_LOG_LEVEL_INFO
levelPrint	ITMG_LOG_LEVEL	设置打印日志的等级，TMG_LOG_LEVEL_NONE 表示不打印，默认为 TMG_LOG_LEVEL_ERROR

ITMG_LOG_LEVEL 说明如下：

ITMG_LOG_LEVEL	含义
TMG_LOG_LEVEL_NONE	不打印日志
TMG_LOG_LEVEL_ERROR	打印错误日志（默认）
TMG_LOG_LEVEL_INFO	打印提示日志
TMG_LOG_LEVEL_DEBUG	打印开发调试日志
TMG_LOG_LEVEL_VERBOSE	打印高频日志

示例代码

```
ITMGContextGetInstance()->SetLogLevel(TMG_LOG_LEVEL_INFO,TMG_LOG_LEVEL_INFO);
```

设置打印日志路径

用于设置打印日志路径。默认路径如下。需要在 Init 之前调用。

平台	路径
Windows	%appdata%\Tencent\GME\ProcessName
iOS	Application/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/Documents
Android	/sdcard/Android/data/xxx.xxx.xxx/files
Mac	/Users/username/Library/Containers/xxx.xxx.xxx/Data/Documents

接口原型

```
ITMGContext virtual int SetLogPath(const char* logDir)
```

参数	类型	含义

logDir	const char*	路径

示例代码

```
cosnt char* logDir = ""//自行设置路径

ITMGContext* context = ITMGContextGetInstance();
context->SetLogPath(logDir);
```

获取诊断信息

获取音视频通话的实时通话质量的相关信息。该接口主要用来查看实时通话质量、排查问题等，业务侧可以忽略。

接口原型

```
ITMGRoom virtual const char* GetQualityTips()
```

示例代码

```
ITMGContextGetInstance()->GetRoom()->GetQualityTips();
```

回调消息

消息	含义	Data	例子
ITMG_MAIN_EVENT_TYPE_ENTER_ROOM	进入音频房间消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_EXIT_ROOM	退出音频房间消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_ROOM_DISCONNECT	房间因为网络等原因断开消息	result; error_info	{"error_info":"waiting timeout, please check your network","result":0}
ITMG_MAIN_EVNET_TYPE_USER_UPDATE	房间成员更新消息	user_list; event_id	{"event_id":1,"user_list":["0"]}
ITMG_MAIN_EVENT_TYPE_RECONNECT_START	房间重连开始消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_RECONNECT_SUCCESS	房间重连成功消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_SWITCH_ROOM	快速切换房间消息	result; error_info	{"error_info":"","result":0}

ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE	房间状态切换消息	result; error_info; sub_event_type; new_room_type	{"error_info":"","new_room_type":0,"result":0}
ITMG_MAIN_EVENT_TYPE_ROOM_SHARING_START	开始跨房连麦消息	result;	{"result":0}
ITMG_MAIN_EVENT_TYPE_ROOM_SHARING_STOP	跨房连麦停止消息	result;	{"result":0}
ITMG_MAIN_EVENT_TYPE_SPEAKER_DEFAULT_DEVICE_CHANGED	默认扬声器设备修改消息	result; error_info	{"deviceID":"{0.0.0.00000000}.{a4f1e8be-49fa-43e2-b8cf-dd00542b47ae}","deviceName":"扬声器 (Realtek High Definition Audio)","error_info":"","isNewDevice":true,"isUsedDevice":false,"result":0}
ITMG_MAIN_EVENT_TYPE_SPEAKER_NEW_DEVICE	新增扬声器设备消息	result; error_info	{"deviceID":"{0.0.0.00000000}.{a4f1e8be-49fa-43e2-b8cf-dd00542b47ae}","deviceName":"扬声器 (Realtek High Definition Audio)","error_info":"","isNewDevice":true,"isUsedDevice":false,"result":0}
ITMG_MAIN_EVENT_TYPE_SPEAKER_LOST_DEVICE	丢失扬声器设备消息	result; error_info	{"deviceID":"{0.0.0.00000000}.{a4f1e8be-49fa-43e2-b8cf-dd00542b47ae}","deviceName":"扬声器 (Realtek High Definition Audio)","error_info":"","isNewDevice":false,"isUsedDevice":false,"result":0}
ITMG_MAIN_EVENT_TYPE_MIC_NEW_DEVICE	新增麦克风设备消息	result; error_info	{"deviceID":"{0.0.1.00000000}.{5fdf1a5b-f42d-4ab2-890a-7e454093f229}","deviceName":"麦克风 (Realtek High Definition Audio)","error_info":"","isNewDevice":true,"isUsedDevice":true,"result":0}
ITMG_MAIN_EVENT_TYPE_MIC_LOST_DEVICE	丢失麦克风设备消息	result; error_info	{"deviceID":"{0.0.1.00000000}.{5fdf1a5b-f42d-4ab2-890a-7e454093f229}","deviceName":"麦克风 (Realtek High Definition Audio)","error_info":"","isNewDevice":false,"isUsedDevice":true,"result":0}
ITMG_MAIN_EVENT_TYPE_MIC_DEFAULT_DEVICE_CHANGED	默认麦克风设备修改消息	result; error_info	{"deviceID":"{0.0.1.00000000}.{5fdf1a5b-f42d-4ab2-890a-7e454093f229}","deviceName":"麦克风 (Realtek High Definition Audio)","error_info":"","isNewDevice":false,"isUsedDevice":true,"result":0}
ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_QUALITY	房间质量消息	weight; loss; delay	{"weight":5,"loss":0.1,"delay":1}
ITMG_MAIN_EVNET_TYPE_PTT_RECORD_COMPLETE	语音消息录制完成消息	result; file_path	{"file_path":"","result":0}

ITMG_MAIN_EVNET_TYPE_PTT_UPLOAD_COMPLETE	语音消息上传完成消息	result; file_path;file_id	{"file_id":"","file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_DOWNLOAD_COMPLETE	语音消息下载完成消息	result; file_path;file_id	{"file_id":"","file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_PLAY_COMPLETE	语音消息播放完成消息	result; file_path	{"file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_SPEECH2TEXT_COMPLETE	语音消息极速转文本完成消息	result; text;file_id	{"file_id":"","text":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_COMPLETE	语音消息流式转文本完成消息	result; file_path; text;file_id	{"file_id":"","file_path":","text":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_IS_RUNNING	语音消息正在流式转文本中	result; file_path; text;file_id	{"file_id":"","file_path":","text":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_TEXT2SPEECH_COMPLETE	文本转语音完成消息	result; text;file_id	{"file_id":"","text":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_TRANSLATE_TEXT_COMPLETE	文本翻译完成消息	result; text;file_id	{"file_id":"","text":"","result":0}

语音消息及转文本

最近更新时间：2024-12-17 17:39:22

为方便 Cocos2d 开发者调试和接入腾讯云游戏多媒体引擎客户端 API，本文为您介绍适用于 Cocos2d 语音消息服务及转文本服务的接入技术文档。

使用 GME 重要事项

GME 提供实时语音服务、语音消息服务及转文本服务，使用 GME 服务都依赖 Init 和 Poll 等核心接口。

> **注意**
> 语音转文本相关接口有默认频率限制，限额范围内计费方式请参见 计费文档；若需提升接口频率限额或了解超额计费方式，请联系商务或 提交工单咨询 。
> - 语音消息非流式转文本接口 **SpeechToText()**：默认单账号限制并发数为10路
> - 语音消息流式转文本接口 **StartRecordingWithStreamingRecognition()**：默认单账号限制并发数为50路
> - 实时语音流式转文本接口 **StartRealTimeASR()**：默认单账号限制并发数为50路

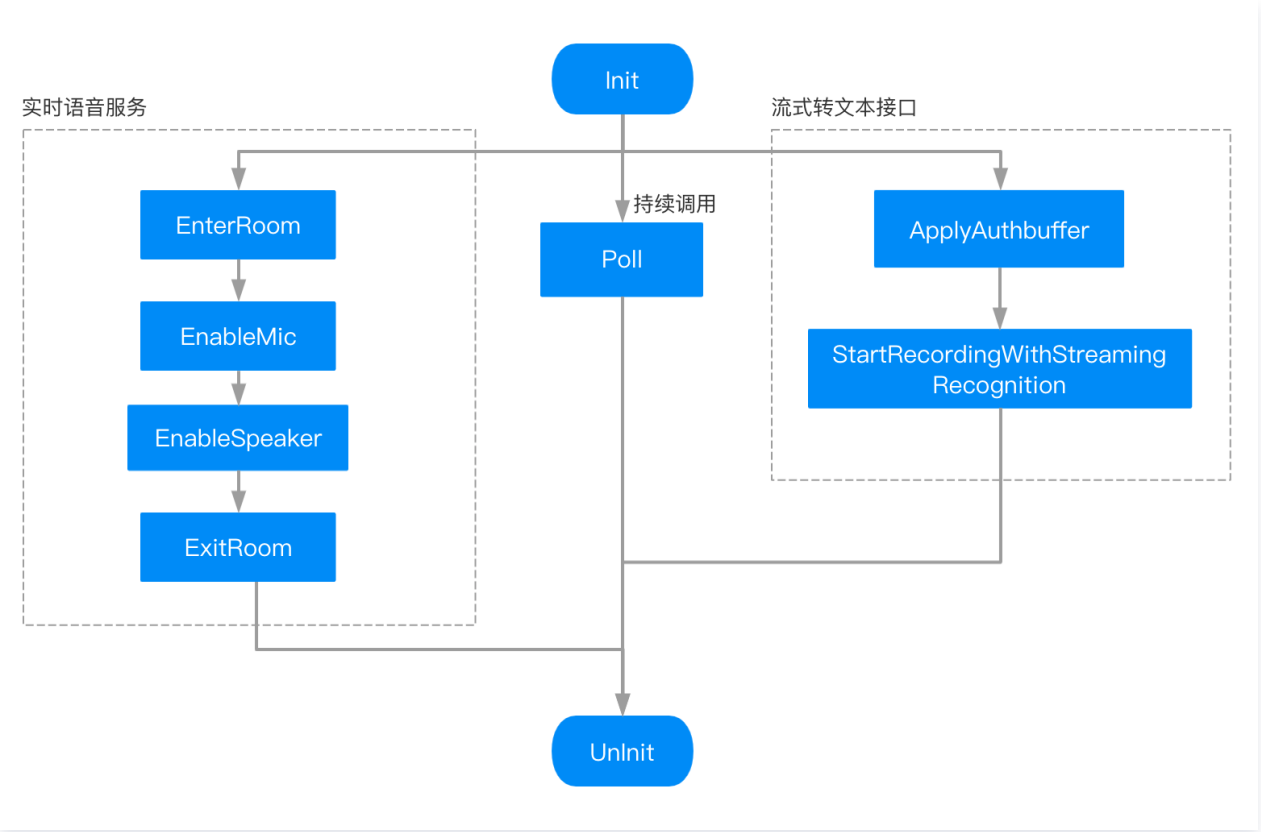
重点提示

- 已完成 GME 应用创建，并获取 SDK AppID 和 Key。请参见 服务开通指引 。
- 已开通 **GME 实时语音服务、语音消息服务以及转文本服务**。请参见 服务开通指引 。
- GME 使用前请对工程进行配置，否则 SDK 不生效。
- GME 的接口调用成功后返回值为 AV_OK，数值为0。
- GME 的接口调用要在同一个线程下。
- GME 需要周期性的调用 Poll 接口触发事件回调。
- 错误码详情可参考 错误码 。

接入 SDK

重要步骤

接入 SDK 重要流程如下：



1. 初始化 GME
2. 周期性调用 Poll 触发回调
3. 鉴权初始化
4. 启动流式语音识别
5. 停止录制
6. 反初始化 GME

C++ 类

类	含义
ITMGContext	核心接口
ITMGPTT	语音消息及转文本相关接口

核心接口

接口	接口含义
Init	初始化 GME
Poll	触发事件回调
Pause	系统暂停
Resume	系统恢复
Uninit	反初始化 GME

引入头文件

```
#include "auth_buffer.h"
#include "tmg_sdk.h"
#include "AdvanceHeaders/tmg_sdk_adv.h"
#include <vector>
```

回调

设置回调示例代码

```
//在初始化 SDK 时候
m_pTmgContext = ITMGContextGetInstance();
m_pTmgContext->SetTMGDelegate(this);

//在析构函数中
CTMGSDK_For_AudioDlg::~CTMGSDK_For_AudioDlg()
{
            ITMGContextGetInstance()->SetTMGDelegate(NULL);
}
```

消息传递

接口类采用 Delegate 方法用于向应用程序发送回调通知，消息类型参考 ITMG_MAIN_EVENT_TYPE，data 在 Windows 平台下是 json 字符串格式， 具体 key-value 参见说明文档。

```
//头文件中声明
virtual void OnEvent(ITMG_MAIN_EVENT_TYPE eventType,const char* data);
//示例代码
void CTMGSDK_For_AudioDlg::OnEvent(ITMG_MAIN_EVENT_TYPE eventType, const char* data)
{
            switch(eventType)
            {
            case ITMG_MAIN_EVENT_TYPE_XXXX_XXXX:
                {
                    //对回调进行处理
                }
                break;
            }
}
```

获取单例

GME SDK 以单例的形式提供，所有调用都从 ITMGContext 开始，通过 ITMGDelegate 回调回传给应用，必须首先设置。

示例代码

```
ITMGContext* m_pTmgContext;
m_pTmgContext->Init(AppID, OpenID);
```

初始化 SDK

未初始化前，SDK 处于未初始化阶段，**需要通过接口 Init 初始化 SDK**，才可以使用实时语音服务、语音消息服务及转文本服务。调用 Init 接口的线程必须于其他接口在同一线程,建议都在主线程调用接口。

接口原型

```
ITMGContext virtual int Init(const char* sdkAppId, const char* openId)
```

参数	类型	含义
sdkAppId	const char*	来自 腾讯云控制台 的 GME 服务提供的 AppID，获取请参见 语音服务开通指引 。
openID	const char*	openID 只支持 Int64 类型（转为 const char* 传入）。规则由 App 开发者自行制定，App 内不重复即可。如需使用字符串作为 Openid 传入，可 提交工单 联系开发者。

返回值

返回值	处理
AV_OK = 0	初始化 SDK 成功
AV_ERR_SDK_NOT_FULL_UPDATE=7015	检查 SDK 文件是否完整，建议删除后重新导入 SDK

⚠ 关于7015错误提示

- 7015错误码是通过 md5 进行判断，在接入过程中若出现此错误，请根据提示检查 SDK 文件是否完整、SDK 文件版本是否一致。
- 出现返回值 AV_ERR_SDK_NOT_FULL_UPDATE 时，此返回值**只有提示作用**，并不会造成初始化失败。
- 由于第三方加固、Unity 打包机制等因素会影响库文件 md5，造成误判，所以**正式发布请在逻辑中忽略此错误**，并尽量不在 UI 中提示。

示例代码

```
#define SDKAPPID3RD "14000xxxxxx"
cosnt char* openId="10001";
ITMGContext* context = ITMGContextGetInstance();
context->Init(SDKAPPID3RD, openId);
```

触发事件回调

通过在 update 里面周期的调用 Poll 可以触发事件回调。Poll 是 GME 的消息泵，GME 需要周期性的调用 Poll 接口触发事件回调。如果没有调用 Poll ，将会导致整个 SDK 服务运行异常。
可参考 Demo 中的 EnginePollHelper.cpp 文件。

⚠ 注意

务必周期性调用 Poll 接口且在主线程调用，以免接口回调异常。

接口原型

```
class ITMGContext {
protected:
    virtual ~ITMGContext() {}

public:
    virtual void Poll()= 0;
}
```

示例代码

```
void TMGTestScene::update(float delta)
{
    ITMGContextGetInstance()->Poll();
}
```

系统暂停

当系统发生 Pause 事件时，需要同时通知引擎进行 Pause。如果不需要后台播放房间内声音，请调用 Pause 接口暂停整个 GME 服务。

接口原型

```
ITMGContext int Pause()
```

系统恢复

当系统发生 Resume 事件时，需要同时通知引擎进行 Resume。Resume 接口只恢复实时语音。

接口原型

```
ITMGContext int Resume()
```

反初始化 SDK

反初始化 SDK，进入未初始化状态。**如果游戏业务侧账号与 openid 是绑定的，那切换游戏账号需要反初始化 GME，再用新的 openid 初始化。**

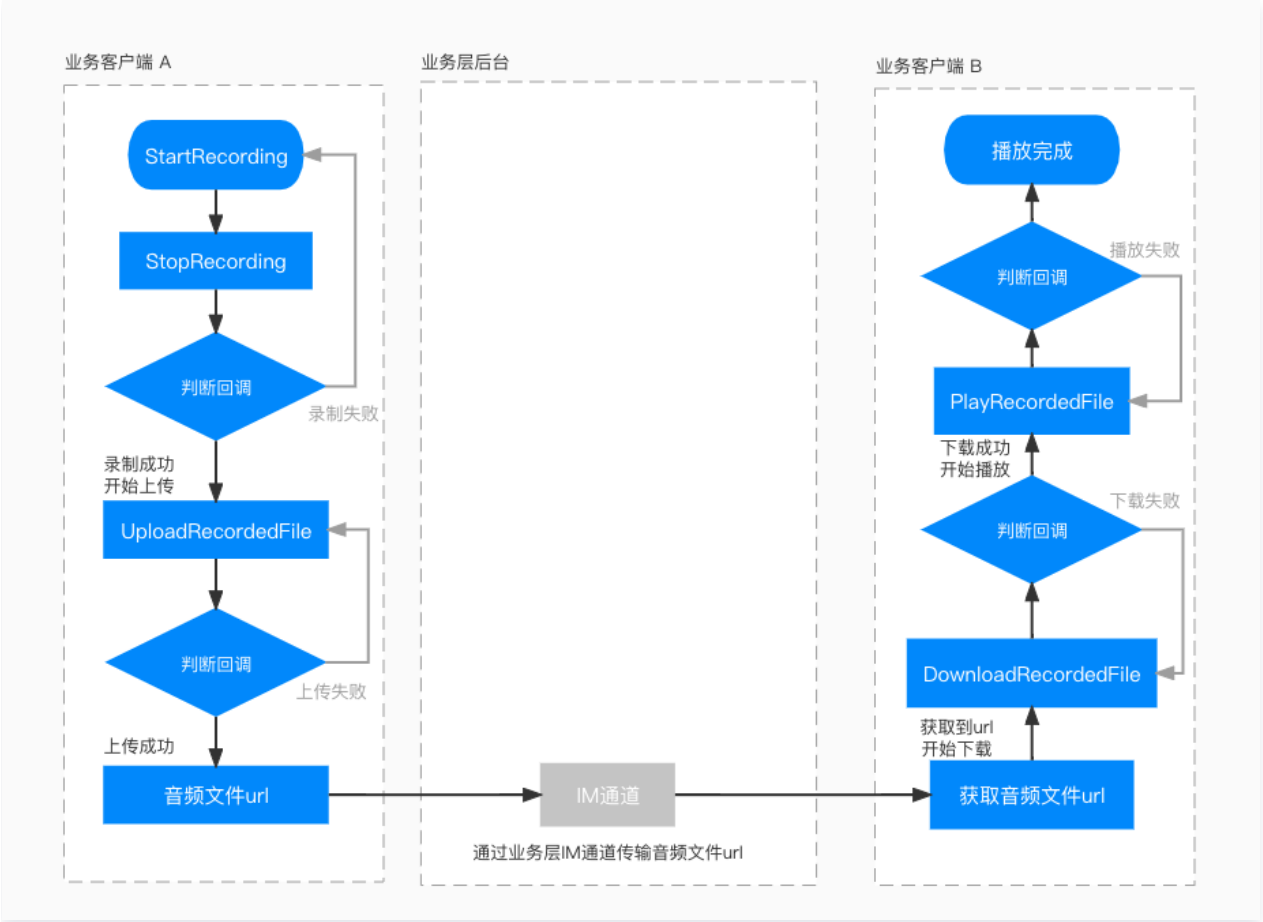
接口原型

```
ITMGContext int Uninit()
```

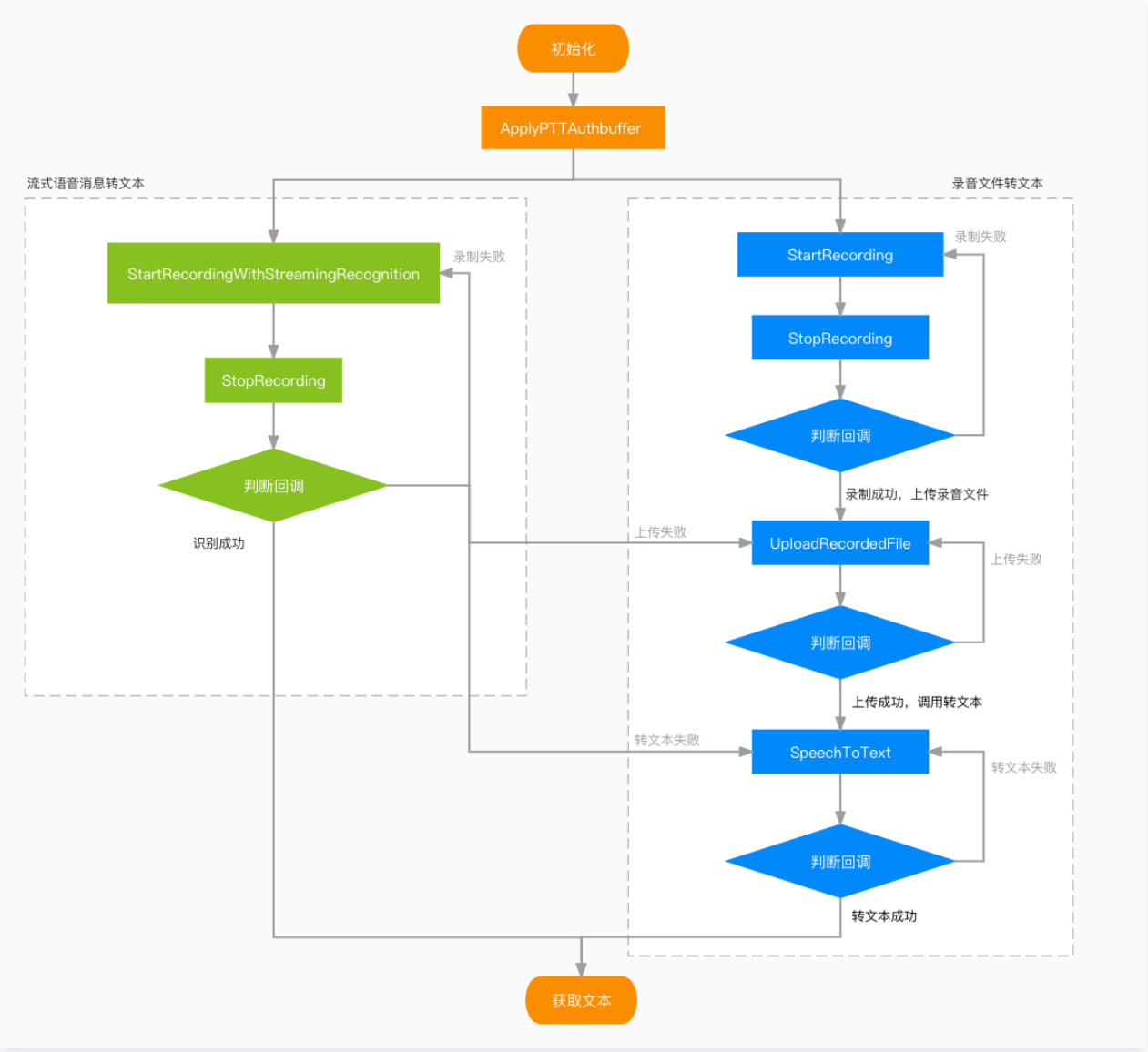
语音消息服务及转文本服务

> **说明**
> - 转文本服务分录音文件极速转文本以及语音消息流式转文本。
> - 使用语音消息服务不需要进入实时语音房间。
> - 语音消息最大录制时长默认为58秒，最短不能小于1秒。如果需要再加以限制，例如限制为最大录制时长为10秒，请在初始化之后调用 SetMaxMessageLength 接口进行设置。

语音消息服务使用流程


业务客户端 A
业务层后台
业务客户端 B
StartRecording
StopRecording
判断回调
录制失败
录制成功
开始上传
UploadRecordedFile
判断回调
上传失败
上传成功
音频文件url
IM通道
通过业务层IM通道传输音频文件url
播放完成
判断回调
播放失败
PlayRecordedFile
下载成功
开始播放
判断回调
下载失败
DownloadRecordedFile
获取到url
开始下载
获取音频文件url

转文本服务使用流程



接口	接口含义
GenAuthBuffer	生成本地鉴权
ApplyPTTAuthbuffer	鉴权初始化
SetMaxMessageLength	限制最大语音信息时长

生成本地鉴权

生成 AuthBuffer，用于相关功能的加密和鉴权，如正式发布请使用后台部署密钥，后台部署请参见 鉴权密钥 。

接口原型

```
int  QAVSDK_AuthBuffer_GenAuthBuffer(unsigned int dwSdkAppID, const char* strRoomID, const char*
strOpenID,
    const char* strKey, unsigned char* strAuthBuffer, unsigned int bufferLength);
```

参数	类型	含义

dwSdkAppID	int	来自腾讯云控制台的 AppId 号码。
strRoomID	const char*	填 null 或者空字符串。
strOpenID	const char*	用户标识。与 Init 时候的 openID相同。
strKey	const char*	来自腾讯云 控制台 的权限密钥。
strAuthBuffer	const char*	返回的 authbuff。
bufferLength	int	传入的 authbuff 长度，建议为500。

应用鉴权

生成鉴权信息后，将鉴权赋值到 SDK 中。

接口原型

```
ITMGPTT virtual int ApplyPTTAuthbuffer(const char* authBuffer, int authBufferLen)
```

参数	类型	含义
authBuffer	const char*	鉴权
authBufferLen	int	鉴权长度

示例代码

```
ITMGContextGetInstance()->GetPTT()->ApplyPTTAuthbuffer(authBuffer,authBufferLen);
```

限制最大语音信息时长

限制最大语音消息的长度，最大支持58秒。

接口原型

```
ITMGPTT virtual int SetMaxMessageLength(int msTime)
```

参数	类型	含义
msTime	int	语音时长，单位 ms，区间为 1000 < msTime <= 58000

示例代码

```
int msTime = 10000;
ITMGContextGetInstance()->GetPTT()->SetMaxMessageLength(msTime);
```

流式语音识别

语音消息及转文字相关接口

接口	接口含义
StartRecordingWithStreamingRecognition	启动流式录音

StopRecording	停止录音

启动流式语音识别

此接口用于启动流式语音识别，同时在回调中会有实时的语音转文字返回，可以指定语言进行识别，也可以将语音中识别到的信息翻译成指定的语言返回。**停止录音调用** 停止录制 。

接口原型

```
ITMGPTT virtual int StartRecordingWithStreamingRecognition(const char* filePath)
ITMGPTT virtual int StartRecordingWithStreamingRecognition(const char* filePath,const char*
translateLanguage,const char* translateLanguage)
```

参数	类型	含义
filePath	const char*	存放的语音路径
speechLanguage	const char*	识别成指定文字的语言参数，参数请参见 语言参数参考列表
translateLanguage	const char*	翻译成指定文字的语言参数，参数请参见 语言参数参考列表

示例代码

```
ITMGContextGetInstance()->GetPTT()->StartRecordingWithStreamingRecognition(filePath,"cmn-Hans-
CN","cmn-Hans-CN");
```

> ⚠ **注意**
> 翻译会收取额外费用，请参见 购买指南 。

流式语音识别的回调

启动流式语音识别后，需要在 OnEvent 通知中监听回调消息，事件消息分为以下两个：

- `ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_COMPLETE` 是在停止录制并完成识别后才返回文字，相当于一段话说完才会返回识别的文字。
- `ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_IS_RUNNING` 是在录音过程中就会实时返回识别到的文字，相当于边说话边返回识别到的文字。

根据需求在 OnEvent 通知中对相应事件消息进行判断。传递的参数包含以下四个信息。

消息名称	含义
result	用于判断流式语音识别是否成功的返回码
text	语音转文字识别的文本
file_path	录音存放的本地地址
file_id	录音在后台的 url 地址，录音在服务器存放90天

> ⚠ **注意**
> 监听 `ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_IS_RUNNING` 消息时，file_id 为空。

错误码

错误码	含义	处理方式
32775	流式语音转文本失败，但是录音成功	调用 UploadRecordedFile 接口上传录音，再调用 SpeechToText 接口进行语音转文字操作
32777	流式语音转文本失败，但是录音成功，上传成功	返回的信息中有上传成功的后台 url 地址，调用 SpeechToText 接口进行语音转文字操作
32786	流式语音转文本失败	在流式录制状态当中，请等待流式录制接口执行结果返回
32787	转文本成功，文本翻译服务未开通	需要在控制台开通文本翻译服务
32788	转文本成功，文本翻译语言参数不支持	重新检查传入参数

如果出现 4098 错误码，请参见 常见问题文档 进行解决。

示例代码

```
void TMGTestScene::OnEvent(ITMG_MAIN_EVENT_TYPE eventType,const char* data){
                switch (eventType) {
                    case ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_COMPLETE:
                    {
                        HandleSTREAM2TEXTComplete(data,true);
                        break;
                        }
                    ...
                            case ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_IS_RUNNING:
                    {
                        HandleSTREAM2TEXTComplete(data, false);
                        break;
                    }
                }
}

void CTMGSDK_For_AudioDlg::HandleSTREAM2TEXTComplete(const char* data, bool isComplete)
{
                    std::string strText = "STREAM2TEXT: ret=";
                    strText += data;
                    m_EditMonitor.SetWindowText(MByteToWChar(strText).c_str());
                    Json::Reader reader;
                    Json::Value root;
                    bool parseRet = reader.parse(data, root);
                    if (!parseRet) {
                        ::SetWindowText(m_EditInfo.GetSafeHwnd(),MByteToWChar(std::string("parse
result Json error")).c_str());
                    }
                    else
                    {
                        if (isComplete) {
                            ::SetWindowText(m_EditUpload.GetSafeHwnd(),
MByteToWChar(root["file_id"].asString()).c_str());
                        }
```

```
                        else {
                            std::string isruning = "STREAMINGRECOGNITION_IS_RUNNING";
                            ::SetWindowText(m_EditUpload.GetSafeHwnd(),
MByteToWChar(isruning).c_str());
                        }
                    }
}
```

语音消息录制

录制的流程为：录音 > 停止录音 > 录音回调返回 > 启动下一次录音。

语音消息及转文字相关接口

接口	接口含义
StartRecording	启动录音
PauseRecording	暂停录音
ResumeRecording	恢复录音
StopRecording	停止录音
CancelRecording	取消录音

启动录音

此接口用于启动录音。

接口原型

```
ITMGPTT virtual int StartRecording(const char* fileDir)
```

参数	类型	含义
fileDir	const char*	存放的语音路径

示例代码

```
char buffer[256]={0};
snprintf(buffer, sizeof(buffer), "%sunreal_ptt_local.file", getFilePath().c_str());
ITMGContextGetInstance()->GetPTT()->StartRecording(buffer);
```

停止录音

此接口用于停止录音。此接口为异步接口，停止录音后会有录音完成回调，成功之后录音文件才可用。

接口原型

```
ITMGPTT virtual int StopRecording()
```

示例代码

```
ITMGContextGetInstance()->GetPTT()->StopRecording();
```

启动录音的回调

启动录音的结果会通过回调返回。
停止录音调用StopRecording。停止录音后才有启动录音的回调。

参数	类型	含义
result	int32	当 code 为 0 时，录制完成
filepath	FString	录制的存放地址，必须是可以访问到的路径，不可将 fileid 作为路径

错误码

错误码值	原因	建议方案
4097	参数为空	检查代码中接口参数是否正确
4098	初始化错误	检查设备是否被占用，或者权限是否正常，是否初始化正常
4099	正在录制中	确保在正确的时机使用 SDK 录制功能
4100	没有采集到音频数据	检查麦克风设备是否正常
4101	录音时，录制文件访问错误	确保文件存在，文件路径的合法性
4102	麦克风未授权错误	使用 SDK 需要麦克风权限，添加权限请参考对应引擎或平台的 SDK 工程配置文档
4103	录音时间太短错误	首先，限制录音时长的单位为毫秒，检查参数是否正确；其次，录音时长要1000毫秒以上才能成功录制
4104	没有启动录音操作	检查是否已经调用启动录音接口

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_EVNET_TYPE_PTT_RECORD_COMPLETE:
                    {
                    //进行处理
                    break;
                    }
                }
}
```

暂停录音

此接口用于暂停录音。如需恢复录音请调用接口 ResumeRecording。

接口原型

```
ITMGPTT virtual int PauseRecording()
```

示例代码

```
ITMGContextGetInstance()->GetPTT()->PauseRecording();
```

恢复录音

此接口用于恢复录音。

接口原型

```
ITMGPTT virtual int ResumeRecording()
```

示例代码

```
ITMGContextGetInstance()->GetPTT()->ResumeRecording();
```

取消录音

调用此接口取消录音。**取消之后没有回调。**

接口原型

```
ITMGPTT virtual int CancelRecording()
```

示例代码

```
ITMGContextGetInstance()->GetPTT()->CancelRecording();
```

语音消息上传、下载及播放

接口	接口含义
UploadRecordedFile	上传语音文件
DownloadRecordedFile	下载语音文件
PlayRecordedFile	播放语音
StopPlayFile	停止播放语音
GetFileSize	语音文件的大小
GetVoiceFileDuration	语音文件的时长

上传语音文件

此接口用于上传语音文件。

接口原型

```
ITMGPTT virtual int UploadRecordedFile(const char* filePath)
```

参数	类型	含义
filePath	const char*	上传的语音路径，此路径为本地路径

示例代码

```
ITMGContextGetInstance()->GetPTT()->UploadRecordedFile(filePath);
```

上传语音完成的回调

上传语音完成后，事件消息为 ITMG_MAIN_EVNET_TYPE_PTT_UPLOAD_COMPLETE，在 OnEvent 函数中对事件消息进行判断。传递的参数包含三个信息，result，file_path 和 file_id。

参数	类型	含义
result	int32	当 code 为0时，录制完成
filepath	FString	录制的存放地址
fileid	FString	文件的 url 路径

错误码

错误码值	原因	建议方案
8193	上传文件时，文件访问错误	确保文件存在，文件路径的合法性
8194	签名校验失败错误	检查鉴权密钥是否正确，检查是否有初始化离线语音
8195	网络错误	检查设备网络是否可以正常访问外网环境
8196	获取上传参数过程中网络失败	检查鉴权是否正确，检查设备网络是否可以正常访问外网环境
8197	获取上传参数过程中回包数据为空	检查鉴权是否正确，检查设备网络是否可以正常访问外网环境
8198	获取上传参数过程中回包解包失败	检查鉴权是否正确，检查设备网络是否可以正常访问外网环境
8200	没有设置 appinfo	检查 apply 接口是否有调用，或者入参是否为空

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_EVNET_TYPE_PTT_UPLOAD_COMPLETE:
            {
            //进行处理
            break;
            }
```

```
        }
}
```

下载语音文件

此接口用于下载语音文件。

接口原型

```
ITMGPTT virtual int DownloadRecordedFile(const char* fileId, const char* filePath)
```

参数	类型	含义
fileId	const char*	文件的 url 路径
filePath	const char*	文件的本地保存路径

示例代码

```
ITMGContextGetInstance()->GetPTT()->DownloadRecordedFile(fileID,filePath);
```

下载语音文件完成回调

下载语音完成后，事件消息为 ITMG_MAIN_EVNET_TYPE_PTT_DOWNLOAD_COMPLETE， 在 OnEvent 函数中对事件消息进行判断。
传递的参数包含三个信息，result、file_path 和 file_id。

参数	类型	含义
result	int32	当 code 为0时，录制完成
filepath	FString	录制的存放地址
fileid	FString	文件的 url 路径，录音在服务器存放 90 天

错误码

错误码值	原因	建议方案
12289	下载文件时，文件访问错误	检查文件路径是否合法
12290	签名校验失败	检查鉴权密钥是否正确，检查是否有初始化离线语音
12291	网络存储系统异常	服务器获取语音文件失败，检查接口参数 fileid 是否正确，检查网络是否正常，检查 COS 文件存不存在
12292	服务器文件系统错误	检查设备网络是否可以正常访问外网环境，检查服务器上是否有此文件
12293	获取下载参数过程中，HTTP 网络失败	检查设备网络是否可以正常访问外网环境
12294	获取下载参数过程中，回包数据为空	检查设备网络是否可以正常访问外网环境
12295	获取下载参数过程中，回包解包失败	检查设备网络是否可以正常访问外网环境

12297	没有设置 appinfo	检查鉴权密钥是否正确，检查是否有初始化离线语音

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;
                }
```

播放语音

此接口用于播放语音。

接口原型

```
ITMGPTT virtual int PlayRecordedFile(const char* filePath)
ITMGPTT virtual int PlayRecordedFile(const char* filePath, nt voiceType)
```

参数	类型	含义
filePath	const char*	本地语音文件的路径
voicetype	int	变声类型，请参见 变声接入文档

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

示例代码

```
ITMGContextGetInstance()->GetPTT()->PlayRecordedFile(filePath);
```

播放语音的回调

播放语音的回调，事件消息为 ITMG_MAIN_EVNET_TYPE_PTT_PLAY_COMPLETE， 在 OnEvent 函数中对事件消息进行判断。传递的参数包含两个信息，一个是 result，另一个是 file_path。

参数	类型	含义
code	int	当 code 为0时，播放完成
filepath	FString	录制的存放地址

错误码

错误码值	原因	建议方案
20481	初始化错误	检查设备是否被占用，或者权限是否正常，是否初始化正常

20482	正在播放中，试图打断并播放下一个失败了（正常是可以打断的）	检查代码逻辑是否正确
20483	参数为空	检查代码中接口参数是否正确
20484	内部错误	初始化播放器错误，解码失败等问题产生此错误码，需要结合日志定位问题

示例代码

```
void TMGTestScene::OnEvent(ITMG_MAIN_EVENT_TYPE eventType,const char* data){
    switch (eventType) {
    ...
        else if (eventType == ITMG_MAIN_EVNET_TYPE_PTT_PLAY_COMPLETE) {
            int32 result = JsonObject->GetIntegerField(TEXT("result"));
            FString filepath = JsonObject->GetStringField(TEXT("file_path"));
            onPttPlayFileCompleted(result, filepath);
        }
    }
}
```

停止播放语音

此接口用于停止播放语音。停止播放语音也会有播放完成的回调。

接口原型

```
ITMGPTT virtual int StopPlayFile()
```

示例代码

```
ITMGContextGetInstance()->GetPTT()->StopPlayFile();
```

获取语音文件的大小

通过此接口，获取语音文件的大小。

接口原型

```
ITMGPTT virtual int GetFileSize(const char* filePath)
```

参数	类型	含义
filePath	const char*	语音文件的路径，此路径为本地路径

示例代码

```
ITMGContextGetInstance()->GetPTT()->GetFileSize(filePath);
```

获取语音文件的时长

此接口用于获取语音文件的时长，单位：毫秒。

接口原型

```
ITMGPTT virtual int GetVoiceFileDuration(const char* filePath)
```

参数	类型	含义
filePath	const char*	语音文件的路径，此路径为本地路径

示例代码

```
ITMGContextGetInstance()->GetPTT()->GetVoiceFileDuration(filePath);
```

录音文件极速转文本

接口	接口含义
SpeechToText	语音识别成文字

将指定的语音文件识别成文字

此接口用于将指定的语音文件识别成文字。

接口原型

```
ITMGPTT virtual void SpeechToText(const char* fileID)
```

参数	类型	含义
fileID	const char*	语音文件 url

示例代码

```
ITMGContextGetInstance()->GetPTT()->SpeechToText(fileID);
```

将指定的语音文件翻译成文字（指定语言）

此接口可以指定语言进行识别，也可以将语音中识别到的信息翻译成指定的语言返回。

> **注意**
> 翻译会收取额外费用，请参见 购买指南 。

接口原型

```
ITMGPTT virtual int SpeechToText(const char* fileID,const char* speechLanguage)
ITMGPTT virtual int SpeechToText(const char* fileID,const char* speechLanguage,const char*
translateLanguage)
```

参数	类型	含义
fileID	const char*	语音文件 url，录音在服务器存放90天
speechLanguage	const char*	识别出指定文字的语言参数，参数参见 语言参数参考列表

translatelanguage	const char*	翻译成指定文字的语言参数，参数参见 言参数参考列表

示例代码

```
ITMGContextGetInstance()->GetPTT()->SpeechToText(filePath,"cmn-Hans-CN","cmn-Hans-CN");
```

识别回调

将指定的语音文件识别成文字的回调，事件消息为 ITMG_MAIN_EVNET_TYPE_PTT_SPEECH2TEXT_COMPLETE，在 OnEvent 函数中对事件消息进行判断。
传递的参数包含三个信息，result、file_path 和 text，其中 text 为识别的文本。

参数	类型	含义
result	int32	当 code 为0时，录制完成
fileid	FString	语音文件 url，录音在服务器存放 90 天
text	FString	转换的文本结果

错误码

错误码值	原因	建议方案
32769	内部错误	分析日志，获取后台返回给客户端的真正错误码，并联系后台同事协助解决
32770	网络失败	检查设备网络是否可以正常访问外网环境
32772	回包解包失败	分析日志，获取后台返回给客户端的真正错误码，并联系后台同事协助解决
32774	没有设置 appinfo	检查鉴权密钥是否正确，检查是否有初始化离线语音
32776	authbuffer 校验失败	检查 authbuffer 是否正确
32784	语音转文本参数错误	检查代码中接口参数 fileid 是否为空
32785	语音转文本翻译返回错误	离线语音后台错误，请分析日志，获取后台返回给客户端的真正错误码，并联系后台同事协助解决
32787	转文本成功，文本翻译服务未开通	需要在控制台开通文本翻译服务
32788	转文本成功，文本翻译语言参数不支持	重新检查传入参数

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_EVNET_TYPE_PTT_SPEECH2TEXT_COMPLETE:
            {
```

```
            //进行处理
            break;
            }
        }
}
```

语音消息音量相关接口

接口	接口含义
GetMicLevel	获取实时麦克风音量
SetMicVolume	设置录制音量
GetMicVolume	获取录制音量
GetSpeakerLevel	获取实时扬声器音量
SetSpeakerVolume	设置播放音量
GetSpeakerVolume	获取播放音量

获取语音消息麦克风实时音量

此接口用于获取麦克风实时音量，返回值为 int 类型，值域为0 – 200。

接口原型

```
ITMGPTT virtual int GetMicLevel()
```

示例代码

```
ITMGContext.GetInstance(this).GetPTT().GetMicLevel();
```

设置语音消息录制音量

此接口用于设置离线语音录制音量，值域为0 – 200。

接口原型

```
ITMGPTT virtual int SetMicVolume(int vol)
```

示例代码

```
ITMGContextGetInstance()->GetPTT()->SetMicVolume(100);
```

获取语音消息录制音量

此接口用于获取离线语音录制音量。返回值为 int 类型，值域为0 – 200。

接口原型

```
ITMGPTT virtual int GetMicVolume()
```

示例代码

```
ITMGContextGetInstance()->GetPTT()->GetMicVolume();
```

获取语音消息扬声器实时音量

此接口用于获取扬声器实时音量。返回值为 int 类型，值域为0 – 200。

接口原型

```
ITMGPTT virtual int GetSpeakerLevel()
```

示例代码

```
ITMGContextGetInstance()->GetPTT()->GetSpeakerLevel();
```

设置语音消息播放音量

此接口用于设置离线语音播放音量，值域为0 – 200。

接口原型

```
ITMGPTT virtual int SetSpeakerVolume(int vol)
```

示例代码

```
ITMGContextGetInstance()->GetPTT()->SetSpeakerVolume(100);
```

获取语音消息播放音量

此接口用于获取离线语音播放音量。返回值为 int 类型，值域为0 – 200。

接口原型

```
ITMGPTT virtual int GetSpeakerVolume()
```

示例代码

```
ITMGContextGetInstance()->GetPTT()->GetSpeakerVolume();
```

高级 API

获取版本号

获取 SDK 版本号，用于分析 SDK 使用情况。

接口原型

```
ITMGContext virtual const char* GetSDKVersion()
```

示例代码

```
ITMGContextGetInstance()->GetSDKVersion();
```

设置打印日志等级

用于设置打印日志等级。建议保持默认等级。需要在 Init 之前调用。

接口原型

```
ITMGContext int SetLogLevel(ITMG_LOG_LEVEL levelWrite, ITMG_LOG_LEVEL levelPrint)
```

参数含义

参数	类型	含义
levelWrite	ITMG_LOG_LEVEL	设置写入日志的等级，TMG_LOG_LEVEL_NONE 表示不写入，默认为 TMG_LOG_LEVEL_INFO
levelPrint	ITMG_LOG_LEVEL	设置打印日志的等级，TMG_LOG_LEVEL_NONE 表示不打印，默认为 TMG_LOG_LEVEL_ERROR

ITMG_LOG_LEVEL 说明如下：

ITMG_LOG_LEVEL	含义
TMG_LOG_LEVEL_NONE	不打印日志
TMG_LOG_LEVEL_ERROR	打印错误日志（默认）
TMG_LOG_LEVEL_INFO	打印提示日志
TMG_LOG_LEVEL_DEBUG	打印开发调试日志
TMG_LOG_LEVEL_VERBOSE	打印高频日志

示例代码

```
ITMGContextGetInstance()->SetLogLevel(TMG_LOG_LEVEL_INFO,TMG_LOG_LEVEL_INFO);
```

设置打印日志路径

用于设置打印日志路径。默认路径如下。需要在 Init 之前调用。

平台	路径
Windows	%appdata%\Tencent\GME\ProcessName
iOS	Application/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/Documents
Android	/sdcard/Android/data/xxx.xxx.xxx/files
Mac	/Users/username/Library/Containers/xxx.xxx.xxx/Data/Documents

接口原型

```
ITMGContext virtual int SetLogPath(const char* logDir)
```

参数	类型	含义
logDir	const char*	路径

示例代码

```
cosnt char* logDir = ""//自行设置路径
ITMGContext* context = ITMGContextGetInstance();
context->SetLogPath(logDir);
```

回调消息

消息	含义	Data	例子
ITMG_MAIN_EVENT_TYPE_ENTER_ROOM	进入音频房间消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_EXIT_ROOM	退出音频房间消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_ROOM_DISCONNECT	房间因为网络等原因断开消息	result; error_info	{"error_info":"waiting timeout, please check your network","result":0}
ITMG_MAIN_EVNET_TYPE_USER_UPDATE	房间成员更新消息	user_list; event_id	{"event_id":1,"user_list":["0"]}
ITMG_MAIN_EVENT_TYPE_RECONNECT_START	房间重连开始消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_RECONNECT_SUCCESS	房间重连成功消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_SWITCH_ROOM	快速切换房间消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE	房间状态切换消息	result; error_info; sub_event_type; new_room_type	{"error_info":"","new_room_type":0,"result":0}
ITMG_MAIN_EVENT_TYPE_ROOM_SHARING_START	开始跨房连麦消息	result;	{"result":0}
ITMG_MAIN_EVENT_TYPE_ROOM_SHARING_STOP	跨房连麦停止消息	result;	{"result":0}
ITMG_MAIN_EVENT_TYPE_SPEAKER_DEF	默认扬声器设备修	result; error_info	{"deviceID":"{0.0.0.00000000}.{a4f1e8be-49fa-43e2-b8cf-dd00542b47ae}","deviceName":"扬声器 (Realtek

AULT_DEVICE_CHANGED	改消息		High Definition Audio)","error_info":"","isNewDevice":true,"isUsedDevice":false,"result":0}
ITMG_MAIN_EVENT_TYPE_SPEAKER_NEW_DEVICE	新增扬声器设备消息	result; error_info	{"deviceID":"{0.0.0.00000000}.{a4f1e8be-49fa-43e2-b8cf-dd00542b47ae}","deviceName":"扬声器 (Realtek High Definition Audio)","error_info":"","isNewDevice":true,"isUsedDevice":false,"result":0}
ITMG_MAIN_EVENT_TYPE_SPEAKER_LOST_DEVICE	丢失扬声器设备消息	result; error_info	{"deviceID":"{0.0.0.00000000}.{a4f1e8be-49fa-43e2-b8cf-dd00542b47ae}","deviceName":"扬声器 (Realtek High Definition Audio)","error_info":"","isNewDevice":false,"isUsedDevice":false,"result":0}
ITMG_MAIN_EVENT_TYPE_MIC_NEW_DEVICE	新增麦克风设备消息	result; error_info	{"deviceID":"{0.0.1.00000000}.{5fdf1a5b-f42d-4ab2-890a-7e454093f229}","deviceName":"麦克风 (Realtek High Definition Audio)","error_info":"","isNewDevice":true,"isUsedDevice":true,"result":0}
ITMG_MAIN_EVENT_TYPE_MIC_LOST_DEVICE	丢失麦克风设备消息	result; error_info	{"deviceID":"{0.0.1.00000000}.{5fdf1a5b-f42d-4ab2-890a-7e454093f229}","deviceName":"麦克风 (Realtek High Definition Audio)","error_info":"","isNewDevice":false,"isUsedDevice":true,"result":0}
ITMG_MAIN_EVENT_TYPE_MIC_DEFAULT_DEVICE_CHANGED	默认麦克风设备修改消息	result; error_info	{"deviceID":"{0.0.1.00000000}.{5fdf1a5b-f42d-4ab2-890a-7e454093f229}","deviceName":"麦克风 (Realtek High Definition Audio)","error_info":"","isNewDevice":false,"isUsedDevice":true,"result":0}
ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_QUALITY	房间质量消息	weight; loss; delay	{"weight":5,"loss":0.1,"delay":1}
ITMG_MAIN_EVNET_TYPE_PTT_RECORD_COMPLETE	语音消息录制完成消息	result; file_path	{"file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_UPLOAD_COMPLETE	语音消息上传完成消息	result; file_path;file_id	{"file_id":"","file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_DOWNLOAD_COMPLETE	语音消息下载完成消息	result; file_path;file_id	{"file_id":"","file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_PLAY_COMPLETE	语音消息播放完成消息	result; file_path	{"file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_SPEECH2TEXT_COMPLETE	语音消息极速转文本完成消息	result; text;file_id	{"file_id":"","text":"","result":0}

ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_COMPLETE	语音消息流式转文本完成消息	result;file_path;text;file_id	{"file_id":"","file_path":","text":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_IS_RUNNING	语音消息正在流式转文本中	result;file_path;text;file_id	{"file_id":"","file_path":","text":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_TEXT2SPEECH_COMPLETE	文本转语音完成消息	result;text;file_id	{"file_id":"","text":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_TRANSLATE_TEXT_COMPLETE	文本翻译完成消息	result;text;file_id	{"file_id":"","text":"","result":0}

版权所有：腾讯云计算（北京）有限责任公司

Windows SDK 工程配置

最近更新时间：2023-03-20 15:36:29

为方便 Windows 开发者调试和接入腾讯云游戏多媒体引擎产品 API，本文档主要为您介绍适用于 Windows 开发的工程配置指引。

SDK 准备

1. 请下载相关 Demo 及 SDK，详情请参见 下载指引 。
2. 解压获取到的 SDK 资源。
3. 文件夹内容如下：
   - include：游戏音视频 SDK 头文件。
   - libs：游戏音视频 SDK dll 文件。

配置指引

1. 将 include 文件夹及 libs 文件夹复制到工程文件夹中。
2. 在工程的属性页中，将 SDK 文件地址添加到附加包含目录中。

版权所有：腾讯云计算（北京）有限责任公司

实时语音

最近更新时间：2024-07-25 15:11:31

为方便 Windows 开发者调试和接入腾讯云游戏多媒体引擎客户端 API，本文为您介绍适用于 Windows 实时语音功能的开发接入技术文档。

使用 GME 重要事项

GME 提供实时语音服务、语音消息服务及转文本服务，使用 GME 服务都依赖 Init 和 Poll 等核心接口。

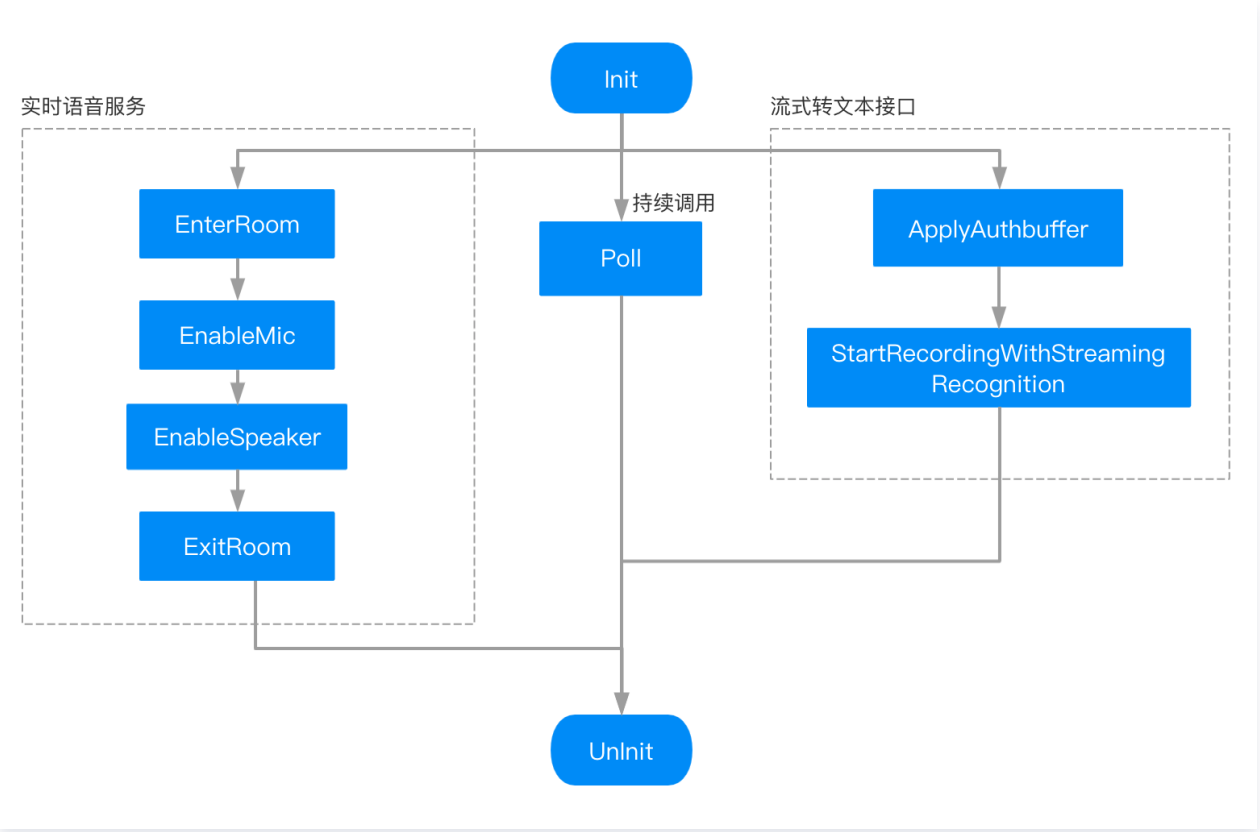
重点提示

- 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、语音消息服务以及转文本服务**。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。
- GME 的接口调用成功后返回值为 QAVError.OK，数值为 0。
- GME 的接口调用要在同一个线程下。
- GME 需要周期性的调用 Poll 接口触发事件回调。
- 错误码详情可参考 错误码 。

接入 SDK

重要步骤

接入 SDK 重要流程如下：



1. 初始化 GME
2. 周期性调用 Poll 触发回调
3. 进入实时语音房间
4. 打开麦克风

5. 打开扬声器
6. 退出语音房间
7. 反初始化 GME

C++ 类

类	含义
ITMGContext	核心接口
ITMGDelegate	回调相关
ITMGRoom	房间相关接口
ITMGRoomManager	房间管理接口
ITMGAudioCtrl	音频相关接口
ITMGAudioEffectCtrl	音效及伴奏相关接口
ITMGPTT	语音消息、转文本相关接口

核心接口

接口	接口含义
Init	初始化 GME
Poll	触发事件回调
Pause	系统暂停
Resume	系统恢复
Uninit	反初始化 GME

引用头文件

接入 GME 首先需要引入头文件 tmg_sdk.h，头文件类继承 ITMGDelegate 以进行消息的传递及回调。

示例代码

```
#include "auth_buffer.h"
#include "tmg_sdk.h"
#include "AdvanceHeaders/tmg_sdk_adv.h"
#include <vector>
```

回调

设置回调示例代码

```
//在初始化 SDK 时候
m_pTmgContext = ITMGContextGetInstance();
m_pTmgContext->SetTMGDelegate(this);

//在析构函数中
CTMGSDK_For_AudioDlg::~CTMGSDK_For_AudioDlg()
{
```

```
            ITMGContextGetInstance()->SetTMGDelegate(NULL);
}
```

消息传递

接口类采用 Delegate 方法用于向应用程序发送回调通知，消息类型参考 ITMG_MAIN_EVENT_TYPE，data 在 Windows 平台下是 json 字符串格式， 具体 key-value 参见说明文档。

```
//头文件中声明
virtual void OnEvent(ITMG_MAIN_EVENT_TYPE eventType,const char* data);
//示例代码
void CTMGSDK_For_AudioDlg::OnEvent(ITMG_MAIN_EVENT_TYPE eventType, const char* data)
{
            switch(eventType)
            {
            case ITMG_MAIN_EVENT_TYPE_XXXX_XXXX:
                {
                    //对回调进行处理
                }
                break;
            }
}
```

获取单例

GME SDK 以单例的形式提供，所有调用都从 ITMGContext 开始，通过 ITMGDelegate 回调回传给应用，必须首先设置。

示例代码

```
ITMGContext* m_pTmgContext;
m_pTmgContext->Init(AppID, OpenID);
```

初始化 SDK

未初始化前，SDK 处于未初始化阶段，**需要通过接口 Init 初始化 SDK**，才可以使用实时语音服务、语音消息服务及转文本服务。调用 Init 接口的线程必须于其他接口在同一线程,建议都在主线程调用接口。

接口原型

```
ITMGContext virtual int Init(const char* sdkAppId, const char* openId)
```

参数	类型	含义
sdkAppId	const char*	来自 腾讯云控制台 的 GME 服务提供的 AppID，获取请参考 语音服务开通指引
openID	const char*	openID 只支持 Int64 类型（转为 const char* 传入）。规则由 App 开发者自行制定，App 内不重复即可。如需使用字符串作为 Openid 传入，可 通过工单 联系开发者。

返回值

返回值	处理
AV_OK = 0	初始化 SDK 成功
AV_ERR_SDK_NOT_FULL_UPDATE=7015	检查 SDK 文件是否完整，建议删除后重新导入 SDK

关于7015错误提示

- 7015错误码是通过 md5 进行判断，在接入过程中若出现此错误，请根据提示检查 SDK 文件是否完整、SDK 文件版本是否一致。
- 出现返回值 AV_ERR_SDK_NOT_FULL_UPDATE 时，此返回值**只有提示作用**，并不会造成初始化失败。
- 由于第三方加固、Unity 打包机制等因素会影响库文件 md5，造成误判，所以**正式发布请在逻辑中忽略此错误**，并尽量不在 UI 中提示。

示例代码

```
#define SDKAPPID3RD "14000xxxxx"
cosnt char* openId="10001";
ITMGContext* context = ITMGContextGetInstance();
context->Init(SDKAPPID3RD, openId);
```

触发事件回调

通过在 update 里面周期的调用 Poll 可以触发事件回调。Poll 是 GME 的消息泵，GME 需要周期性的调用 Poll 接口触发事件回调。如果没有调用 Poll ，将会导致整个 SDK 服务运行异常。
可参考 Demo 中的 EnginePollHelper.cpp 文件。

务必周期性调用 Poll 接口
务必周期性调用 Poll 接口且在主线程调用，以免接口回调异常。

接口原型

```
class ITMGContext {
protected:
    virtual ~ITMGContext() {}

public:
    virtual void Poll()= 0;
}
```

示例代码

```
//头文件中的声明

//代码实现
void TMGTestScene::update(float delta)
{
    ITMGContextGetInstance()->Poll();
}
```

系统暂停

当系统发生 Pause 事件时，需要同时通知引擎进行 Pause。如果不需要后台播放房间内声音，请调用 Pause 接口暂停整个 GME 服务。

接口原型

```
ITMGContext int Pause()
```

系统恢复

当系统发生 Resume 事件时，需要同时通知引擎进行 Resume。Resume 接口只恢复实时语音。

接口原型

```
ITMGContext int Resume()
```

反初始化 SDK

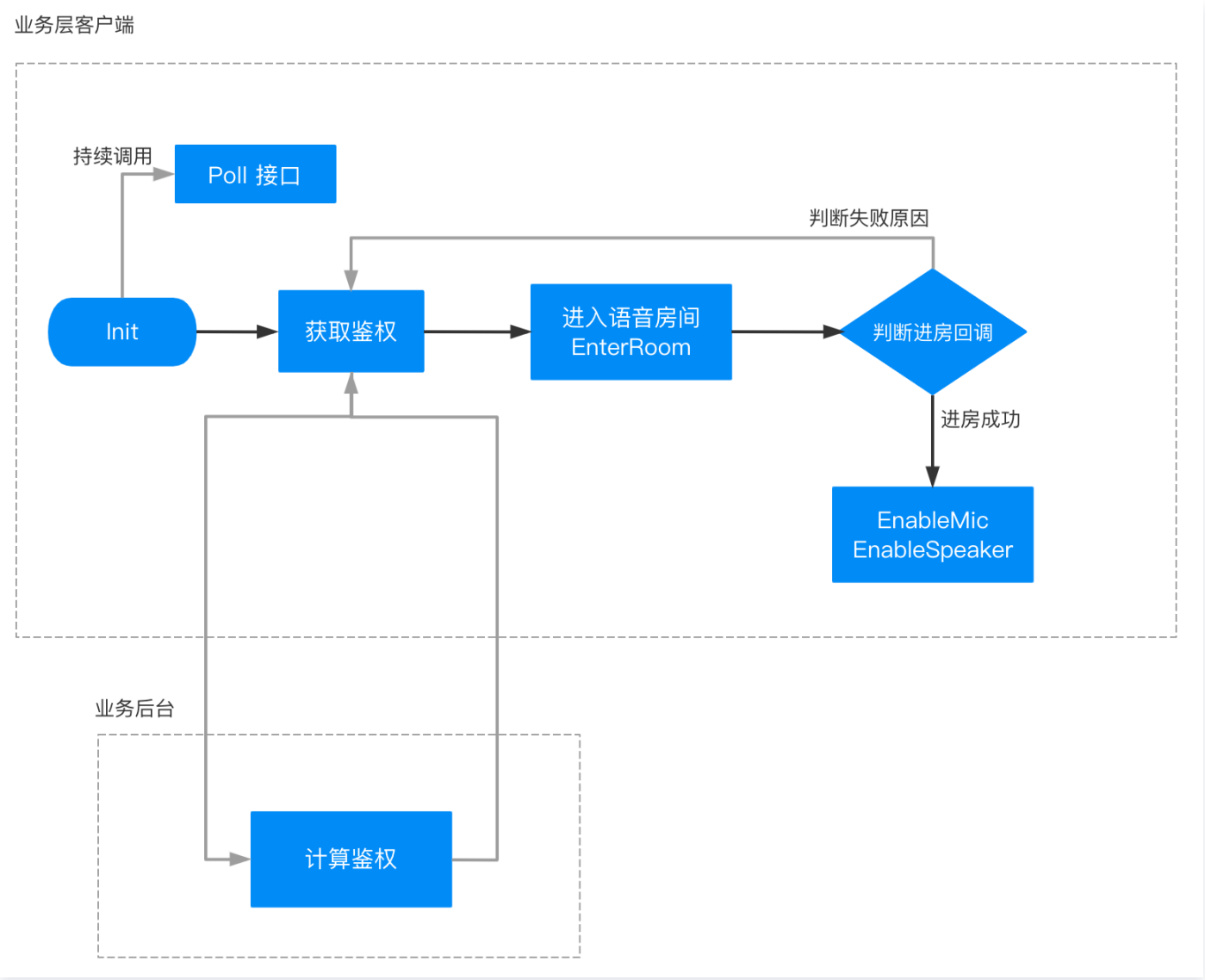
反初始化 SDK，进入未初始化状态。**如果游戏业务侧账号与 openid 是绑定的，那切换游戏账号需要反初始化 GME，再用新的 openid 初始化。**

接口原型

```
ITMGContext int Uninit()
```

实时语音房间相关接口

初始化之后，SDK 调用进房后进去了房间，才可以进行实时语音通话。
使用问题可参考 实时语音相关问题 。



接口	接口含义
GenAuthBuffer	本地鉴权计算
EnterRoom	加入房间
ExitRoom	退出房间
IsRoomEntered	判断是否已经进入房间
SwitchRoom	快速切换房间

本地鉴权计算

生成 AuthBuffer，用于相关功能的加密和鉴权，如正式发布请使用后台部署密钥，后台部署请参考 鉴权密钥 。

接口原型

```
int  QAVSDK_AuthBuffer_GenAuthBuffer(unsigned int dwSdkAppID, const char* strRoomID, const char*
strOpenID,
    const char* strKey, unsigned char* strAuthBuffer, unsigned int bufferLength);
```

参数	类型	含义
dwSdkAppID	unsigned int	来自腾讯云控制台的 AppId 号码
strRoomID	const char*	房间号，最大支持127字符
strOpenID	const char*	用户标识。与 Init 时候的 openID相同。
strKey	const char*	来自腾讯云 控制台 的权限密钥
strAuthBuffer	const char*	返回的 authbuff
bufferLength	int	传入的 authbuff 长度，建议为 500

示例代码

```
unsigned int bufferLen = 512;
unsigned char retAuthBuff[512] = {0};
QAVSDK_AuthBuffer_GenAuthBuffer(atoi(SDKAPPID3RD), roomId, "10001",
AUTHKEY,retAuthBuff,bufferLen);
```

加入房间

用生成的鉴权信息进房，加入房间默认不打开麦克风及扬声器。

> **注意**
> - 加入房间事件回调结果 result 为 0 代表进房成功，进房接口 EnterRoom 返回值为 0 不代表进房成功。
> - 房间的音频类型由第一个进房的人确定，此后房间里有成员修改房间类型，将对此房间所有成员生效。例如第一个进入房间的人使用的房间音频类型是流畅音质，第二个进房的是即使进房时候调用接口的音频类型参数是高清音质，进入房间之后也会变成流畅音质。需要有成员调用 ChangeRoomType 才会修改房间的音频类型。

接口原型

```
ITMGContext virtual int EnterRoom(const char*  roomID, ITMG_ROOM_TYPE roomType, const char*
authBuff, int buffLen)
```

参数	类型	含义
roomID	const char*	房间号，最大支持127字符
roomType	ITMG_ROOM_TYPE	房间类型，游戏建议使用 ITMG_ROOM_TYPE_FLUENCY。房间音频类型请参考 音质选择 。
authBuffer	const char*	鉴权码
buffLen	int	鉴权码长度

示例代码

```
ITMGContext* context = ITMGContextGetInstance();
context->EnterRoom(roomID, ITMG_ROOM_TYPE_FLUENCY, (char*)retAuthBuff,bufferLen);
```

加入房间事件回调

加入房间完成后会发送信息 ITMG_MAIN_EVENT_TYPE_ENTER_ROOM，在 OnEvent 函数中进行判断回调后处理。如果回调为成功，即此时进房成功，开始进行计费。

> **计费问题参考：**
> - 购买指南。
> - 计费相关问题。
> - 使用实时语音后，如果客户端掉线了，是否还会继续计费？

示例代码

```
void TMGTestScene::OnEvent(ITMG_MAIN_EVENT_TYPE eventType,const char* data){
                    switch (eventType) {
                        case ITMG_MAIN_EVENT_TYPE_ENTER_ROOM:
                        {
                            ListMicDevices();
                            ListSpeakerDevices();

                            std::string strText = "EnterRoom complete: ret=";
                            strText += data;
                            m_EditMonitor.SetWindowText(MByteToWChar(strText).c_str());
                            }
                    }
}
```

Data 详情

消息	Data	例子
ITMG_MAIN_EVENT_TYPE_ENTER_ROOM	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_ROOM_DISCONNECT	result; error_info	{"error_info":"waiting timeout, please check your network","result":0}

如果断网，将会有断网的回调提示 `ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVENT_TYPE_ROOM_DISCONNECT`，此时 SDK 会自动进行重连，回调是 `ITMG_MAIN_EVENT_TYPE_RECONNECT_START`，当重连成功时，会有 `ITMG_MAIN_EVENT_TYPE_RECONNECT_SUCCESS` 回调。

错误码

错误码值	原因及建议方案
7006	鉴权失败原因。 • AppID 不存在或者错误 • authbuff 鉴权错误 • 鉴权过期 • OpenId 不符合规范
7007	已经在其它房间
1001	已经在进房过程中，然后又重复了此操作。建议在进房回调返回之前不要再调用进房接口
1003	已经进房了在房间中，又调用一次进房接口
1101	确保已经初始化 SDK，确保 OpenId 是否符合规则，或者确保在同一线程调用接口，以及确保 Poll 接口正常调用

版权所有：腾讯云计算（北京）有限责任公司

退出房间

通过调用此接口可以退出所在房间。这是一个异步接口，返回值为 AV_OK 的时候代表异步投递成功。如果应用中有退房后立即进房的场景，在接口调用流程上，开发者无需要等待 ExitRoom 的回调 RoomExitComplete 通知，只需直接调用接口。

接口原型

```
ITMGContext virtual int ExitRoom()
```

示例代码

```
ITMGContext* context = ITMGContextGetInstance();
context->ExitRoom();
```

退出房间事件回调

退出房间完成后会有回调，消息为 ITMG_MAIN_EVENT_TYPE_EXIT_ROOM。

示例代码

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

Data详情

消息	Data	例子
ITMG_MAIN_EVENT_TYPE_EXIT_ROOM	result; error_info	{"error_info":"","result":0}

判断是否已经进入房间

通过调用此接口可以判断是否已经进入房间，返回值为 bool 类型。请勿在进房过程中调用。

接口原型

```
ITMGContext virtual bool IsRoomEntered()
```

示例代码

```
ITMGContext* context = ITMGContextGetInstance();
context->IsRoomEntered();
```

快速切换房间

调用此接口快速切换实时语音房间。此接口在进房后调用。切换房间后，不重置设备，即如果在此房间已经是打开麦克风状态，在切换房间后也会是打开麦克风状态。

快速切换房间的回调是 ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVENT_TYPE_SWITCH_ROOM，字段是 error_info 以及 result。

接口原型

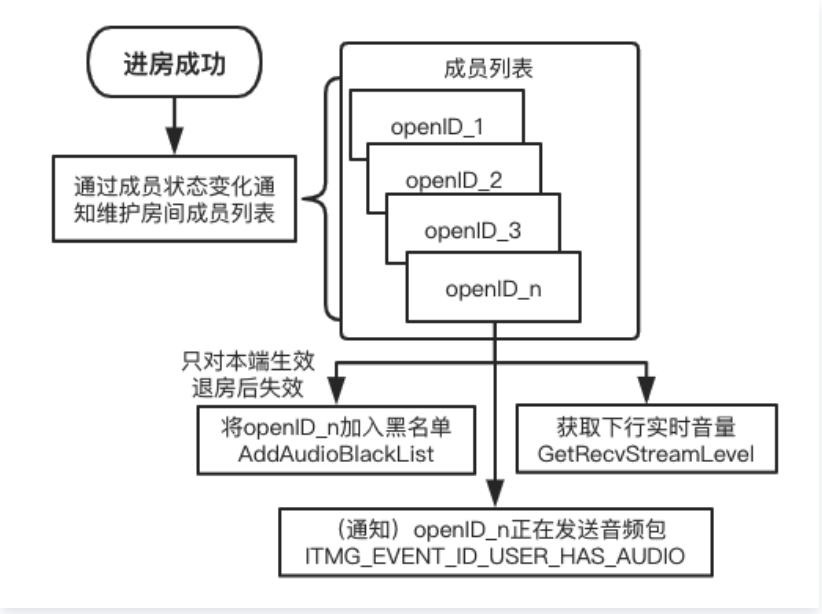
```
ITMGContext virtual int SwitchRoom(const char* targetRoomID, const char* authBuff, int buffLen);
```

类型说明

参数	类型	含义
targetRoomID	const char*	将要进入的房间号
authBuffer	const char*	用将要进入的房间号生成的新鉴权
buffLen	int	鉴权码长度

房间内状态维护

此部分接口用于业务层显示说话成员、进退房成员，以及将房间内某成员禁言等功能。



接口/通知	含义
ITMG_MAIN_EVNET_TYPE_USER_UPDATE	成员状态变化通知
AddAudioBlackList	房间中禁言某成员
RemoveAudioBlackList	移除禁言

成员进房、开关麦克风、说话状态通知事件

此接口适用于获取房间中说话的人并在 UI 中展示，以及有人进入、退出语音房间的一个通知。
该事件在状态变化才通知，状态不变化的情况下不通知。如需实时获取成员状态，请在上层收到通知时缓存，事件消息为 ITMG_MAIN_EVNET_TYPE_USER_UPDATE，其中 data 包含两个信息，event_id 及 user_list，在 OnEvent 函数中对事件消息进行判断。
音频事件的通知有一个阈值，超过这个阈值才会发送通知。超过两秒没有收到音频包才通知“有成员停止发送音频包”消息。

event_id	含义	应用侧维护内容
ITMG_EVENT_ID_USER_ENTER	有成员进入房间，返回此时进房的 openid	应用侧维护成员列表
ITMG_EVENT_ID_USER_EXIT	有成员退出房间，返回此时退房的 openid	应用侧维护成员列表
ITMG_EVENT_ID_USER_HAS_AUDIO	有成员发送音频包，返回此时房间内说话的 openid，通过此事件可以判断用户是否说话，并展示声纹效果	应用侧维护通话成员列表
ITMG_EVENT_ID_USER_NO_AUDIO	有成员停止发送音频包，返回此时房间内停止说话的 openid	应用侧维护通话成员列表
ITMG_EVENT_ID_USER_MIC_OPENED	有成员打开麦克风设备（2.9.10及以后版本支持）	应用侧维护通话成员列表
ITME_EVENT_ID_USER_MIC_CLOSED	有成员关闭麦克风设备（2.9.10及以后版本支持）	应用侧维护通话成员列表

示例代码

```
void TMGTestScene::OnEvent(ITMG_MAIN_EVENT_TYPE eventType,const char* data){
    switch (eventType) {
            case ITMG_MAIN_EVNET_TYPE_USER_UPDATE:
        {
        //进行处理
        //开发者对参数进行解析，得到信息 eventID 及 user_list
            switch (eventID)
             {
             case ITMG_EVENT_ID_USER_ENTER:
                  //有成员进入房间
                 break;
             case ITMG_EVENT_ID_USER_EXIT:
                  //有成员退出房间
                break;
            case ITMG_EVENT_ID_USER_HAS_AUDIO:
                //有成员发送音频包
                break;
            case ITMG_EVENT_ID_USER_NO_AUDIO:
                //有成员停止发送音频包
                break;
             default:
                break;
            }
        break;
        }
    }
}
```

房间中禁言某成员

将某个 ID 加入音频数据黑名单，即不接受某人的语音， 只对本端生效，不会影响其他端。返回值为 0 表示调用成功。例如： A、B、C 都在同一个房间开麦说话：

- 如果 A 设置了 C 的黑名单， 则 A 只能听见 B 的声音。
- B 因为没有设置黑名单， 仍旧可以听见 A 和 C 的声音。

- C 同样因为没有设置黑名单，可以听见 A 和 B 的声音。

此接口适用于在语音房间中将某用户禁言的场景。

接口原型

```
ITMGContext ITMGAudioCtrl int AddAudioBlackList(const char* openId)
```

参数	类型	含义
openId	char*	需添加黑名单的用户 openid

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->AddAudioBlackList(openId);
```

移除禁言

将某个 Id 移除音频数据黑名单。返回值为0表示调用成功。

接口原型

```
ITMGContext ITMGAudioCtrl int RemoveAudioBlackList(const char* openId)
```

参数	类型	含义
openId	char*	需移除黑名单的 ID

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->RemoveAudioBlackList(openId);
```

实时语音采集相关接口

- 初始化 SDK 之后进房，在房间中，才可以调用实时音频语音相关接口。
- 当用户界面单击打开/关闭麦克风/扬声器按钮时，建议采用 EnableMic 以及 EnableSpeaker 接口进行调用。
- 当用户界面按住麦克风按钮时发言，放开按钮不发言，建议采用进房时候调用 EnableAudioCaptureDevice 一次，后续按住发言调用 EnableAudioSend 来实现。

接口	接口含义
EnableMic	开关麦克风
GetMicState	获取麦克风状态
EnableAudioCaptureDevice	开关采集设备
IsAudioCaptureDeviceEnabled	获取采集设备状态
EnableAudioSend	打开关闭音频上行
IsAudioSendEnabled	获取音频上行状态
GetMicLevel	获取实时麦克风音量

GetSendStreamLevel	获取音频上行实时音量
SetMicVolume	设置麦克风音量
GetMicVolume	获取麦克风音量

开启或关闭麦克风

此接口用来开启关闭麦克风。加入房间默认不打开麦克风及扬声器。EnableMic = EnableAudioCaptureDevice + EnableAudioSend

接口原型

```
ITMGAudioCtrl virtual int EnableMic(bool bEnabled)
```

参数	类型	含义
bEnabled	bool	如果需要打开麦克风，则传入的参数为 true，如果关闭麦克风，则参数为 false

示例代码

```
//打开麦克风
ITMGContextGetInstance()->GetAudioCtrl()->EnableMic(true);
```

麦克风状态获取

此接口用于获取麦克风状态，返回值0为关闭麦克风状态，返回值1为打开麦克风状态。

接口原型

```
ITMGAudioCtrl virtual int GetMicState()
```

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->GetMicState();
```

开启或关闭采集设备

此接口用来开启/关闭采集设备。加入房间默认不打开设备。

- 只能在进房后调用此接口，退房会自动关闭设备。
- 在移动端，打开采集设备通常会伴随权限申请，音量类型调整等操作。

接口原型

```
ITMGAudioCtrl virtual int EnableAudioCaptureDevice(bool enable)
```

参数	类型	含义
enable	bool	如果需要打开采集设备，则传入的参数为 true，如果关闭采集设备，则参数为 false

示例代码

```
//打开采集设备
ITMGContextGetInstance()->GetAudioCtrl()->EnableAudioCaptureDevice(true);
```

采集设备状态获取

此接口用于采集设备状态获取。

接口原型

```
ITMGContext virtual bool IsAudioCaptureDeviceEnabled()
```

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->IsAudioCaptureDeviceEnabled();
```

打开或关闭音频上行

此接口用于打开/关闭音频上行。如果采集设备已经打开，那么会发送采集到的音频数据。如果采集设备没有打开，那么仍旧无声。采集设备的打开关闭请参见接口 EnableAudioCaptureDevice。

接口原型

```
ITMGContext  virtual int EnableAudioSend(bool bEnable)
```

参数	类型	含义
bEnable	bool	如果需要打开音频上行，则传入的参数为 true，如果关闭音频上行，则参数为 false

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->EnableAudioSend(true);
```

音频上行状态获取

此接口用于音频上行状态获取。

接口原型

```
ITMGContext virtual bool IsAudioSendEnabled()
```

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->IsAudioSendEnabled();
```

获取麦克风实时音量

此接口用于获取麦克风实时音量，返回值为 int 类型。建议20ms获取一次。值域为0 – 100，通过此接口可以获取到麦克风采集到的实时音量情况。

接口原型

```
ITMGAudioCtrl virtual int GetMicLevel()
```

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->GetMicLevel();
```

获取音频上行实时音量

此接口用于获取自己音频上行实时音量，返回值为 int 类型，取值范围为0 – 100。

接口原型

```
ITMGAudioCtrl virtual int GetSendStreamLevel()
```

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->GetSendStreamLevel();
```

设置麦克风软件音量

此接口用于设置麦克风的音量。参数 volume 用于设置麦克风的音量，相当于对采集的声音做衰减或增益。

接口原型

```
ITMGAudioCtrl virtual int SetMicVolume(int vol)
```

参数	类型	含义
vol	int	取值范围为 0–200，数值为0的时候表示静音，当数值为100的时候表示音量不增不减，默认数值为100。

示例代码

```
int micVol = (int)(value * 100);
ITMGContextGetInstance()->GetAudioCtrl()->SetMicVolume(vol);
```

获取麦克风软件音量

此接口用于获取麦克风的音量。返回值为一个int类型数值，返回值为101代表没调用过接口 SetMicVolume。

接口原型

```
ITMGAudioCtrl virtual int GetMicVolume()
```

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->GetMicVolume();
```

实时语音播放相关接口

接口	接口含义

EnableSpeaker	开关扬声器
GetSpeakerState	获取扬声器状态
EnableAudioPlayDevice	开关播放设备
IsAudioPlayDeviceEnabled	获取播放设备状态
EnableAudioRecv	打开关闭音频下行
IsAudioRecvEnabled	获取音频下行状态
GetSpeakerLevel	获取实时扬声器音量
GetRecvStreamLevel	获取房间内其他成员下行实时音量
SetSpeakerVolume	设置扬声器音量
GetSpeakerVolume	获取扬声器音量
TrackingVolume	开始监听房间内用户的音量变化
StopTrackingVolume	停止监听房间内用户的音量变化

开启或关闭扬声器

此接口用于开启关闭扬声器。**EnableSpeaker = EnableAudioPlayDevice + EnableAudioRecv**

接口原型

```
ITMGAudioCtrl virtual int EnableSpeaker(bool enable)
```

参数	类型	含义
enable	bool	如果需要关闭扬声器，则传入的参数为 false，如果打开扬声器，则参数为 true

示例代码

```
//打开扬声器
ITMGContextGetInstance()->GetAudioCtrl()->EnableSpeaker(true);
```

扬声器状态获取

此接口用于扬声器状态获取。返回值0为关闭扬声器状态，返回值1为打开扬声器状态。

接口原型

```
ITMGAudioCtrl virtual int GetSpeakerState()
```

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->GetSpeakerState();
```

开启或关闭播放设备

此接口用于开启关闭播放设备。

接口原型

```
ITMGAudioCtrl virtual int EnableAudioPlayDevice(bool enable)
```

参数	类型	含义
enable	bool	如果需要关闭播放设备，则传入的参数为 false，如果打开播放设备，则参数为 true

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->EnableAudioPlayDevice(true);
```

播放设备状态获取

此接口用于播放设备状态获取。

接口原型

```
ITMGAudioCtrl virtual bool IsAudioPlayDeviceEnabled()
```

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->IsAudioPlayDeviceEnabled();
```

打开或关闭音频下行

此接口用于打开/关闭音频下行。如果播放设备已经打开，那么会播放房间里其他人的音频数据。如果播放设备没有打开，那么仍旧无声。播放设备的打开关闭参见接口请参见 EnableAudioPlayDevice。

接口原型

```
ITMGAudioCtrl virtual int EnableAudioRecv(bool enable)
```

参数	类型	含义
enable	bool	如果需要打开音频下行，则传入的参数为 true，如果关闭音频下行，则参数为 false

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->EnableAudioRecv(true);
```

音频下行状态获取

此接口用于音频下行状态获取。

接口原型

```
ITMGAudioCtrl virtual bool IsAudioRecvEnabled()
```

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->IsAudioRecvEnabled();
```

获取扬声器实时音量

此接口用于获取扬声器实时音量。返回值为 int 类型数值，表示扬声器实时音量。建议20ms获取一次。

接口原型

```
ITMGAudioCtrl virtual int GetSpeakerLevel()
```

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->GetSpeakerLevel();
```

获取房间内其他成员下行实时音量

此接口用于获取房间内其他成员下行实时音量，返回值为 int 类型，取值范围为0 – 200。

接口原型

```
ITMGAudioCtrl virtual int GetRecvStreamLevel(const char* openId)
```

参数	类型	含义
openId	char*	房间其他成员的openId

示例代码

```
iter->second.level = ITMGContextGetInstance()->GetAudioCtrl()->GetRecvStreamLevel(iter-
>second.openid.c_str());
```

动态设置房间内某成员音量

此接口用于设置房间内某成员的说话音量大小，此设置只在本端生效。

接口原型

```
ITMGAudioCtrl virtual int SetSpeakerVolumeByOpenID(const char* openId, int vol) = 0;
```

参数	类型	含义
openId	const char*	需要调节音量大小的 OpenID
vol	Int	百分比，建议[0-200]，其中100为默认值

获取设置的声音百分比

调用此接口获取 SetSpeakerVolumeByOpenID 设置的能量值

接口原型

```
ITMGAudioCtrl virtual int GetSpeakerVolumeByOpenID(const char* openId) = 0;
```

参数	类型	含义
openId	const char*	需要调节音量大小的OpenID

返回值

接口返回 OpenID 设置的能量百分比， 默认返回100。

设置扬声器的音量

此接口用于设置扬声器的音量。

接口原型

```
ITMGAudioCtrl virtual int SetSpeakerVolume(int vol)
```

参数	类型	含义
vol	Int	设置音量，范围0 – 200，当数值为0时，表示静音，当数值为100时，表示音量不增不减，默认数值为100。

示例代码

```
int vol = 100;
ITMGContextGetInstance()->GetAudioCtrl()->SetSpeakerVolume(vol);
```

获取扬声器的音量

此接口用于获取扬声器的音量。返回值为 int 类型数值，代表扬声器的音量，返回值为101代表没调用过接口 SetSpeakerVolume。
Level 是实时音量，Volume 是扬声器的音量，最终声音音量 = Level × Volume %。例如实时音量是数值是100，此时 Volume 的数值是60，那么最终发出来的声音数值也是60。

接口原型

```
ITMGAudioCtrl virtual int GetSpeakerVolume()
```

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->GetSpeakerVolume();
```

开始监听房间内用户的音量变化

开始监听房间类用户（包括自己）的音量变化，当音量有变化时会触发ITMG_MAIN_EVENT_TYPE_USER_VOLUMES时间回调。

接口原型

```
ITMGAudioCtrl virtual int TrackingVolume(float trackingTimeS);
```

参数	类型	含义
trackingTimeS	Float	音量变化事件回调的间隔，单位为秒

消息	含义	data

ITMG_MAIN_EVENT_TYPE_USER_VOLUMES	房间内用户音量发生变化	[{"open_id":"abc", "volume":2}, {"open_id":"def", "volume":2}]

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->TrackingVolume(1.0);
```

停止监听房间内用户的音量变化

停止监听房间内用户（包括自己）的音量变化。

接口原型

```
ITMGAudioCtrl virtual int StopTrackingVolume()
```

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->StopTrackingVolume();
```

设备选择相关接口

接口	接口含义
GetMicListCount	获取麦克风设备数量
GetMicList	枚举麦克风设备
GetSpeakerListCount	获取扬声器设备数量
GetSpeakerList	枚举扬声器设备
SelectMic	选定麦克风设备
SelectSpeaker	选定扬声器设备
GetCurrentMic	获取当前使用的麦克风设备
GetCurrentSpeaker	获取当前使用的扬声器设备

获取麦克风设备数量

此接口用来获取麦克风设备数量。

函数原型

```
ITMGAudioCtrl virtual int GetMicListCount()
```

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->GetMicListCount();
```

枚举麦克风设备

此接口用来枚举麦克风设备。配合 GetMicListCount 接口使用。

函数原型

```
ITMGAudioCtrl virtual int GetMicList(TMGAudioDeviceInfo* ppDeviceInfoList, int nCount)

class TMGAudioDeviceInfo
{
public:
    const char* pDeviceID;
    const char* pDeviceName;
};
```

参数	类型	含义
ppDeviceInfoList	TMGAudioDeviceInfo	设备列表
nCount	Int	获取的麦克风设备数量

TMGAudioDeviceInfo 参数	类型	含义
pDeviceID	const char*	设备名称
pDeviceName	const char*	设备 ID

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->GetMicList(ppDeviceInfoList,nCount);
```

选定麦克风设备

此接口用来选中麦克风设备。如果不调用或者传入 "DEVICEID_DEFAULT"，则选中系统默认设备。GetMicList接口中返回的第0个设备 id 为默认设备，未选中设备时通话设备为默认设备，选中后业务层维护通话设备。如果该通话设备被拔出，则此时通话设备为默认设备，拔出的通话设备插入后，此时通话设备恢复为插入的通话设备。

函数原型

```
ITMGAudioCtrl virtual int SelectMic(const char* pMicID)
```

参数	类型	含义
pMicID	const char*	麦克风设备 ID，设备 ID 来自于 GetMicList 返回列表。

示例代码

```
const char* pMicID ="{0.0.1.00000000}.{7b0b712d-3b46-4f7a-bb83-bf9be4047f0d}";
ITMGContextGetInstance()->GetAudioCtrl()->SelectMic(pMicID);
```

获取当前使用的麦克风设备

获取当前选中的麦克风设备信息。

函数原型

```
ITMGAudioCtrl virtual int GetCurrentMic(TMGAudioDeviceInfo &DeviceInfo);
```

参数	类型	含义
DeviceInfo	TMGAudioDeviceInfo	设备信息

TMGAudioDeviceInfo 数据结构参看 GetMicList 接口介绍。

示例代码

```
TMGAudioDeviceInfo micInfo;
ITMGContextGetInstance()->GetAudioCtrl()-
>TMGAudioDeviceInfo(micInfoITMG_EVENT_ID_USER_MIC_OPENED );
```

获取扬声器设备数量

此接口用来获取扬声器设备数量。

接口原型

```
ITMGAudioCtrl virtual int GetSpeakerListCount()
```

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->GetSpeakerListCount();
```

枚举扬声器设备

此接口用来枚举扬声器设备。配合 GetSpeakerListCount 接口使用。

函数原型

```
ITMGAudioCtrl virtual int GetSpeakerList(TMGAudioDeviceInfo* ppDeviceInfoList, int nCount)
```

参数	类型	含义
ppDeviceInfoList	TMGAudioDeviceInfo	设备列表
nCount	Int	获取的扬声器设备数量

TMGAudioDeviceInfo 数据结构参看 GetMicList 接口介绍。

选中扬声器设备

此接口用来选中播放设备。如果不调用或者传入 "DEVICEID_DEFAULT"，则选中系统默认播放设备。

函数原型

```
ITMGAudioCtrl virtual int SelectSpeaker(const char* pSpeakerID)
```

参数	类型	含义
pSpeakerID	const char*	扬声器设备 ID，设备 ID 来自于 GetSpeakerList 返回列表。

示例代码

```
const char* pSpeakerID ="{0.0.1.00000000}.{7b0b712d-3b46-4f7a-bb83-bf9be4047f0d}";
ITMGContextGetInstance()->GetAudioCtrl()->SelectSpeaker(pSpeakerID);
```

获取当前使用的扬声器设备

获取当前选中的扬声器设备信息

函数原型

```
ITMGAudioCtrl virtual int GetCurrentSpeaker(TMGAudioDeviceInfo &DeviceInfo);
```

参数	类型	含义
DeviceInfo	TMGAudioDeviceInfo	扬声器设备信息

示例代码

```
TMGAudioDeviceInfo speakerInfo;
ITMGContextGetInstance()->GetAudioCtrl()->GetCurrentSpeaker(speakerInfo);
```

高级 API

启动耳返

此接口用于启动耳返，需要 `EnableLoopBack + EnableSpeaker` 才可以听到自己声音。

接口原型

```
ITMGAudioCtrl virtual int EnableLoopBack(bool enable)
```

参数	类型	含义
enable	bool	设置是否启动

示例代码

```
ITMGContextGetInstance()->GetAudioCtrl()->EnableLoopBack(true);
```

获取用户房间音频类型

此接口用于获取用户房间音频类型，返回值为房间音频类型，返回值为0时代表获取用户房间音频类型发生错误，房间音频类型参考 EnterRoom 接口。

接口原型

```
class ITMGRoom {
public:
    virtual ~ITMGRoom() {} ;
    virtual int GetRoomType() = 0;

};
```

示例代码

```
ITMGContext* context = ITMGContextGetInstance();
ITMGContextGetInstance()->GetRoom()->GetRoomType();
```

获取房间号

此接口用于获取实时语音房间号，只能在进房成功之后使用。

接口原型

```
ITMGRoom virtual int GetRoomID(char* pBuffer, int nLength) = 0;
```

参数	类型	含义
pBuffer	char*	用于接收返回的 roomid
nLength	int	pBuffer 长度，在 128 – 256 之间

修改用户房间音频类型

此接口用于修改用户房间音频类型，结果参见回调事件，事件类型为 ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE。房间的音频类型由第一个进房的人确定，此后房间里有成员修改房间类型，将对此房间所有成员生效。

接口原型

```
IITMGContext TMGRoom public int ChangeRoomType((ITMG_ROOM_TYPE roomType)
```

参数	类型	含义
roomType	ITMG_ROOM_TYPE	房间切换成的目标类型，房间音频类型参考 EnterRoom 接口

示例代码

```
ITMGContext* context = ITMGContextGetInstance();
ITMGContextGetInstance()->GetRoom()->ChangeRoomType(ITMG_ROOM_TYPE_FLUENCY);
```

Data 详情

消息	Data	例子
ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE	result; error_info; new_room_type	{"error_info":"","new_room_type":0,"result":0}

房间类型修改回调

房间类型设置完成后，回调的事件消息为 ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE，返回的参数为 result、error_info 及 new_room_type，new_room_type 代表的信息如下，在 OnEvent 函数中对事件消息进行判断。

事件子类型	代表参数	含义
ITMG_ROOM_CHANGE_EVENT_ENTERROOM	1	表示在进房的过程中，自带的音频类型与房间不符合，被修改为所进入房间的音频类型。
ITMG_ROOM_CHANGE_EVENT_START	2	表示已经在房间内，音频类型开始切换（例如调用 ChangeRoomType 接口后切换音频类型）。
ITMG_ROOM_CHANGE_EVENT_COMPLETE	3	表示已经在房间，音频类型切换完成。
ITMG_ROOM_CHANGE_EVENT_REQUEST	4	表示房间成员调用 ChangeRoomType 接口，请求切换房间音频类型。

示例代码

```
void TMGTestScene::OnEvent(ITMG_MAIN_EVENT_TYPE eventType,const char* data) {
    if (ITMGContext.ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE == type)
        {
        //对房间类型事件进行处理
     }
}
```

房间通话质量监控事件

质量监控事件，在进房后触发，2秒回调一次，事件消息为 ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_QUALITY，返回的参数为 weight、loss 及 delay，代表的信息如下：

参数	类型	含义
weight	int	范围是1 – 50，数值为50是音质评分极好，数值为1是音质评分很差，几乎不能使用，数值为0代表初始值，无含义。一般数值在30以下就可以提醒用户网络较差，建议切换网络。
loss	double	上行丢包率。
delay	int	音频触达延迟时间（ms）。

获取版本号

获取 SDK 版本号，用于分析。

接口原型

```
ITMGContext virtual const char* GetSDKVersion()
```

示例代码

```
ITMGContextGetInstance()->GetSDKVersion();
```

检查麦克风设备状态

函数原型

```
ITMGContext virtual ITMG_CHECK_MIC_STATUS CheckMic() = 0;
```

返回值处理

返回值	含义	处理
ITMG_CHECK_MIC_STATUS_AVAILABLE = 0	正常可用	无需处理
ITMG_CHECK_MIC_STATUS_NO_GRANTED = 2	未获得/拒绝授权权限	需要在打开麦克风之前获取下权限
ITMG_CHECK_MIC_STATUS_INVALID_MIC = 3	没有可用的设备	一般是 PC 设备上，没有可用的麦克风设备会报此错误，请提示插入耳机或麦克风
ITMG_CHECK_MIC_STATUS_NOT_INIT = 5	没有初始化	在Init之后调用 EnableMic 接口

设置打印日志等级

用于设置打印日志等级。建议保持默认等级。需要在 Init 之前调用。

接口原型

```
ITMGContext int SetLogLevel(ITMG_LOG_LEVEL levelWrite, ITMG_LOG_LEVEL levelPrint)
```

参数含义

参数	类型	含义
levelWrite	ITMG_LOG_LEVEL	设置写入日志的等级，TMG_LOG_LEVEL_NONE 表示不写入，默认为 TMG_LOG_LEVEL_INFO
levelPrint	ITMG_LOG_LEVEL	设置打印日志的等级，TMG_LOG_LEVEL_NONE 表示不打印，默认为 TMG_LOG_LEVEL_ERROR

ITMG_LOG_LEVEL 说明如下：

ITMG_LOG_LEVEL	含义
TMG_LOG_LEVEL_NONE	不打印日志
TMG_LOG_LEVEL_ERROR	打印错误日志（默认）
TMG_LOG_LEVEL_INFO	打印提示日志
TMG_LOG_LEVEL_DEBUG	打印开发调试日志
TMG_LOG_LEVEL_VERBOSE	打印高频日志

示例代码

```
ITMGContextGetInstance()->SetLogLevel(TMG_LOG_LEVEL_INFO,TMG_LOG_LEVEL_INFO);
```

设置打印日志路径

用于设置打印日志路径。默认路径如下。需要在 Init 之前调用。

平台	路径
Windows	%appdata%\GMEGLOBAL\GME\ProcessName
iOS	Application/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/Documents
Android	/sdcard/Android/data/xxx.xxx.xxx/files
Mac	/Users/username/Library/Containers/xxx.xxx.xxx/Data/Documents

接口原型

```
ITMGContext virtual int SetLogPath(const char* logDir)
```

参数	类型	含义
logDir	const char*	路径

示例代码

```
cosnt char* logDir = ""//自行设置路径

ITMGContext* context = ITMGContextGetInstance();
context->SetLogPath(logDir);
```

获取打印日志路径

通过接口获取日志路径，返回值为 const char* 类型字符串。

接口原型

```
ITMGContext virtual const char* GetLogPath() = 0;
```

获取诊断信息

获取音视频通话的实时通话质量的相关信息。该接口主要用来查看实时通话质量、排查问题等，业务侧可以忽略。

接口原型

```
ITMGRoom virtual const char* GetQualityTips()
```

示例代码

```
ITMGContextGetInstance()->GetRoom()->GetQualityTips();
```

回调消息

消息	含义	Data	例子
ITMG_MAIN_EVENT_TYPE_ENTER_ROOM	进入音频房间消息	result; error_info	{"error_info":"","result":0}

ITMG_MAIN_EVENT_TYPE_EXIT_ROOM	退出音频房间消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_ROOM_DISCONNECT	房间因为网络等原因断开消息	result; error_info	{"error_info":"waiting timeout, please check your network","result":0}
ITMG_MAIN_EVNET_TYPE_USER_UPDATE	房间成员更新消息	user_list; event_id	{"event_id":1,"user_list":["0"]}
ITMG_MAIN_EVENT_TYPE_RECONNECT_START	房间重连开始消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_RECONNECT_SUCCESS	房间重连成功消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_SWITCH_ROOM	快速切换房间消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE	房间状态切换消息	result; error_info; sub_event_type; new_room_type	{"error_info":"","new_room_type":0,"result":0}
ITMG_MAIN_EVENT_TYPE_ROOM_SHARING_START	开始跨房连麦消息	result;	{"result":0}
ITMG_MAIN_EVENT_TYPE_ROOM_SHARING_STOP	跨房连麦停止消息	result;	{"result":0}
ITMG_MAIN_EVENT_TYPE_SPEAKER_DEFAULT_DEVICE_CHANGED	默认扬声器设备修改消息	result; error_info	{"deviceID":"{0.0.0.00000000}.{a4f1e8be-49fa-43e2-b8cf-dd00542b47ae}","deviceName":"扬声器 (Realtek High Definition Audio)","error_info":"","isNewDevice":true,"isUsedDevice":false,"result":0}
ITMG_MAIN_EVENT_TYPE_SPEAKER_NEW_DEVICE	新增扬声器设备消息	result; error_info	{"deviceID":"{0.0.0.00000000}.{a4f1e8be-49fa-43e2-b8cf-dd00542b47ae}","deviceName":"扬声器 (Realtek High Definition Audio)","error_info":"","isNewDevice":true,"isUsedDevice":false,"result":0}
ITMG_MAIN_EVENT_TYPE_SPEAKER_LOST_DEVICE	丢失扬声器设备消息	result; error_info	{"deviceID":"{0.0.0.00000000}.{a4f1e8be-49fa-43e2-b8cf-dd00542b47ae}","deviceName":"扬声器 (Realtek High Definition Audio)","error_info":"","isNewDevice":false,"isUsedDevice":false,"result":0}
ITMG_MAIN_EVENT_TYPE_MIC_NEW_DEVICE	新增麦克风设备消息	result; error_info	{"deviceID":"{0.0.1.00000000}.{5fdf1a5b-f42d-4ab2-890a-7e454093f229}","deviceName":"麦克风 (Realtek High Definition

			Audio)","error_info":"","isNewDevice":true,"isUsedDevice":true,"result":0}
ITMG_MAIN_EVENT_TYPE_MIC_LOST_DEVICE	丢失麦克风设备消息	result; error_info	{"deviceID":"{0.0.1.00000000}.{5fdf1a5b-f42d-4ab2-890a-7e454093f229}","deviceName":"麦克风 (Realtek High Definition Audio)","error_info":"","isNewDevice":false,"isUsedDevice":true,"result":0}
ITMG_MAIN_EVENT_TYPE_MIC_DEFAULT_DEVICE_CHANGED	默认麦克风设备修改消息	result; error_info	{"deviceID":"{0.0.1.00000000}.{5fdf1a5b-f42d-4ab2-890a-7e454093f229}","deviceName":"麦克风 (Realtek High Definition Audio)","error_info":"","isNewDevice":false,"isUsedDevice":true,"result":0}
ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_QUALITY	房间质量消息	weight; loss; delay	{"weight":5,"loss":0.1,"delay":1}
ITMG_MAIN_EVNET_TYPE_PTT_RECORD_COMPLETE	语音消息录制完成消息	result; file_path	{"file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_UPLOAD_COMPLETE	语音消息上传完成消息	result; file_path;file_id	{"file_id":"","file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_DOWNLOAD_COMPLETE	语音消息下载完成消息	result; file_path;file_id	{"file_id":"","file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_PLAY_COMPLETE	语音消息播放完成消息	result; file_path	{"file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_SPEECH2TEXT_COMPLETE	语音消息极速转文本完成消息	result; text;file_id	{"file_id":"","text":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_COMPLETE	语音消息流式转文本完成消息	result; file_path; text;file_id	{"file_id":"","file_path":","text":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_IS_RUNNING	语音消息正在流式转文本中	result; file_path; text;file_id	{"file_id":"","file_path":","text":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_TEXT2SPEECH_COMPLETE	文本转语音完成消息	result; text;file_id	{"file_id":"","text":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_TRANSLATE_TEXT_COMPLETE	文本翻译完成消息	result; text;file_id	{"file_id":"","text":"","result":0}

语音消息及转文本

最近更新时间：2024-12-17 17:39:22

为方便 Windows 开发者调试和接入腾讯云游戏多媒体引擎客户端 API，本文为您介绍适用于 Windows 语音消息服务及转文本服务的接入技术。

使用 GME 重要事项

GME 提供实时语音服务、语音消息服务及转文本服务，使用 GME 服务都依赖 Init 和 Poll 等核心接口。

重点提示

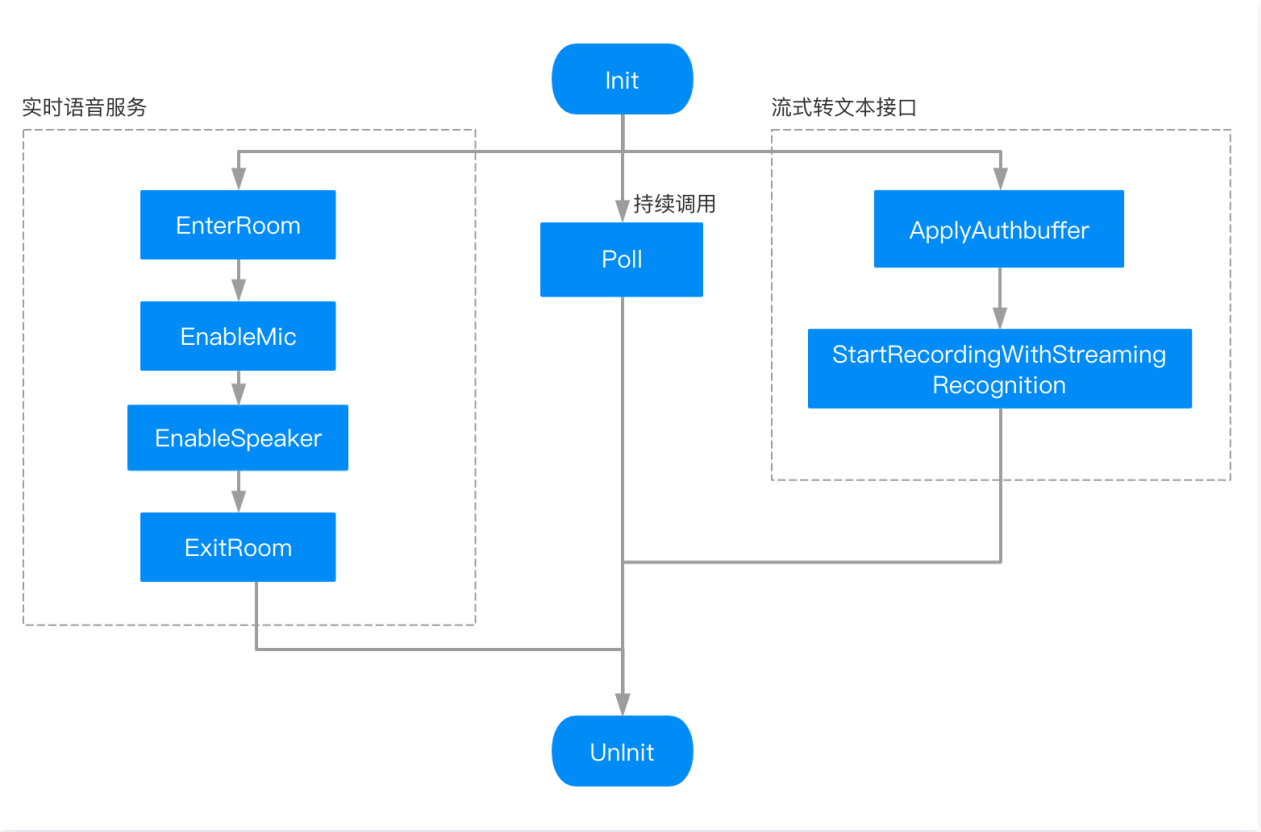
- 已完成 GME 应用创建，并获取 SDK AppID 和 Key。请参见 服务开通指引 。
- 已开通 **GME 实时语音服务、语音消息服务以及转文本服务**。请参见 服务开通指引 。
- GME 使用前请对工程进行配置，否则 SDK 不生效。
- GME 的接口调用成功后返回值为 QAVError.OK，数值为0。
- GME 的接口调用要在同一个线程下。
- GME 需要周期性的调用 Poll 接口触发事件回调。
- 错误码详情可参考 错误码 。

> ⚠ **注意**
> 语音转文本相关接口有默认频率限制，限额范围内计费方式请参见 计费文档；若需提升接口频率限额或了解超额计费方式，请联系商务或 提交工单咨询 。
> - 语音消息非流式转文本接口 ***SpeechToText()***：默认单账号限制并发数为10路
> - 语音消息流式转文本接口 ***StartRecordingWithStreamingRecognition()***：默认单账号限制并发数为50路
> - 实时语音流式转文本接口 ***StartRealTimeASR()***：默认单账号限制并发数为50路

接入 SDK

重要步骤

接入 SDK 重要流程如下：



1. 初始化 GME
2. 周期性调用 Poll 触发回调
3. 鉴权初始化
4. 启动流式语音识别
5. 停止录制
6. 反初始化 GME

C++ 类

类	含义
ITMGContext	核心接口
ITMGPTT	语音消息及转文本相关接口

核心接口

接口	接口含义
Init	初始化 GME
Poll	触发事件回调
Pause	系统暂停
Resume	系统恢复
Uninit	反初始化 GME

引入头文件

```
#include "auth_buffer.h"
#include "tmg_sdk.h"
#include "AdvanceHeaders/tmg_sdk_adv.h"
#include <vector>
```

回调

设置回调示例代码

```
//在初始化 SDK 时候
m_pTmgContext = ITMGContextGetInstance();
m_pTmgContext->SetTMGDelegate(this);

//在析构函数中
CTMGSDK_For_AudioDlg::~CTMGSDK_For_AudioDlg()
{
            ITMGContextGetInstance()->SetTMGDelegate(NULL);
}
```

消息传递

接口类采用 Delegate 方法用于向应用程序发送回调通知，消息类型参考 ITMG_MAIN_EVENT_TYPE，data 在 Windows 平台下是 json 字符串格式， 具体 key-value 参见说明文档。

```
//头文件中声明
virtual void OnEvent(ITMG_MAIN_EVENT_TYPE eventType,const char* data);
//示例代码
void CTMGSDK_For_AudioDlg::OnEvent(ITMG_MAIN_EVENT_TYPE eventType, const char* data)
{
            switch(eventType)
            {
            case ITMG_MAIN_EVENT_TYPE_XXXX_XXXX:
                {
                    //对回调进行处理
                }
                break;
            }
}
```

获取单例

GME SDK 以单例的形式提供，所有调用都从 ITMGContext 开始，通过 ITMGDelegate 回调回传给应用，必须首先设置。

示例代码

```
ITMGContext* m_pTmgContext;
m_pTmgContext->Init(AppID, OpenID);
```

初始化 SDK

未初始化前，SDK 处于未初始化阶段，**需要通过接口 Init 初始化 SDK**，才可以使用实时语音服务、语音消息服务及转文本服务。调用 Init 接口的线程必须于其他接口在同一线程,建议都在主线程调用接口。

接口原型

```
ITMGContext virtual int Init(const char* sdkAppId, const char* openId)
```

参数	类型	含义
sdkAppId	const char*	来自 腾讯云控制台 的 GME 服务提供的 AppID，获取请参见 语音服务开通指引 。
openID	const char*	openID 只支持 Int64 类型（转为 const char* 传入）。规则由 App 开发者自行制定，App 内不重复即可。如需使用字符串作为 Openid 传入，可 提交工单 联系开发者。

返回值

返回值	处理
AV_OK = 0	初始化 SDK 成功
AV_ERR_SDK_NOT_FULL_UPDATE=7015	检查 SDK 文件是否完整，建议删除后重新导入 SDK

⚠ 关于7015错误提示

- 7015错误码是通过 md5 进行判断，在接入过程中若出现此错误，请根据提示检查 SDK 文件是否完整、SDK 文件版本是否一致。
- 出现返回值 AV_ERR_SDK_NOT_FULL_UPDATE 时，此返回值**只有提示作用**，并不会造成初始化失败。
- 由于第三方加固、Unity 打包机制等因素会影响库文件 md5，造成误判，所以**正式发布请在逻辑中忽略此错误**，并尽量不在 UI 中提示。

示例代码

```
#define SDKAPPID3RD "14000xxxxxx"
cosnt char* openId="10001";
ITMGContext* context = ITMGContextGetInstance();
context->Init(SDKAPPID3RD, openId);
```

触发事件回调

通过在 update 里面周期的调用 Poll 可以触发事件回调。Poll 是 GME 的消息泵，GME 需要周期性的调用 Poll 接口触发事件回调。如果没有调用 Poll ，将会导致整个 SDK 服务运行异常。
可参考 Demo 中的 EnginePollHelper.cpp 文件。

⚠ 注意

务必周期性调用 Poll 接口且在主线程调用，以免接口回调异常。

接口原型

```
class ITMGContext {
protected:
    virtual ~ITMGContext() {}

public:
    virtual void Poll()= 0;
}
```

示例代码

```
void TMGTestScene::update(float delta)
{
    ITMGContextGetInstance()->Poll();
}
```

系统暂停

当系统发生 Pause 事件时，需要同时通知引擎进行 Pause。如果不需要后台播放房间内声音，请调用 Pause 接口暂停整个 GME 服务。

接口原型

```
ITMGContext int Pause()
```

系统恢复

当系统发生 Resume 事件时，需要同时通知引擎进行 Resume。Resume 接口只恢复实时语音。

接口原型

```
ITMGContext int Resume()
```

反初始化 SDK

反初始化 SDK，进入未初始化状态。**如果游戏业务侧账号与 openid 是绑定的，那切换游戏账号需要反初始化 GME，再用新的 openid 初始化。**

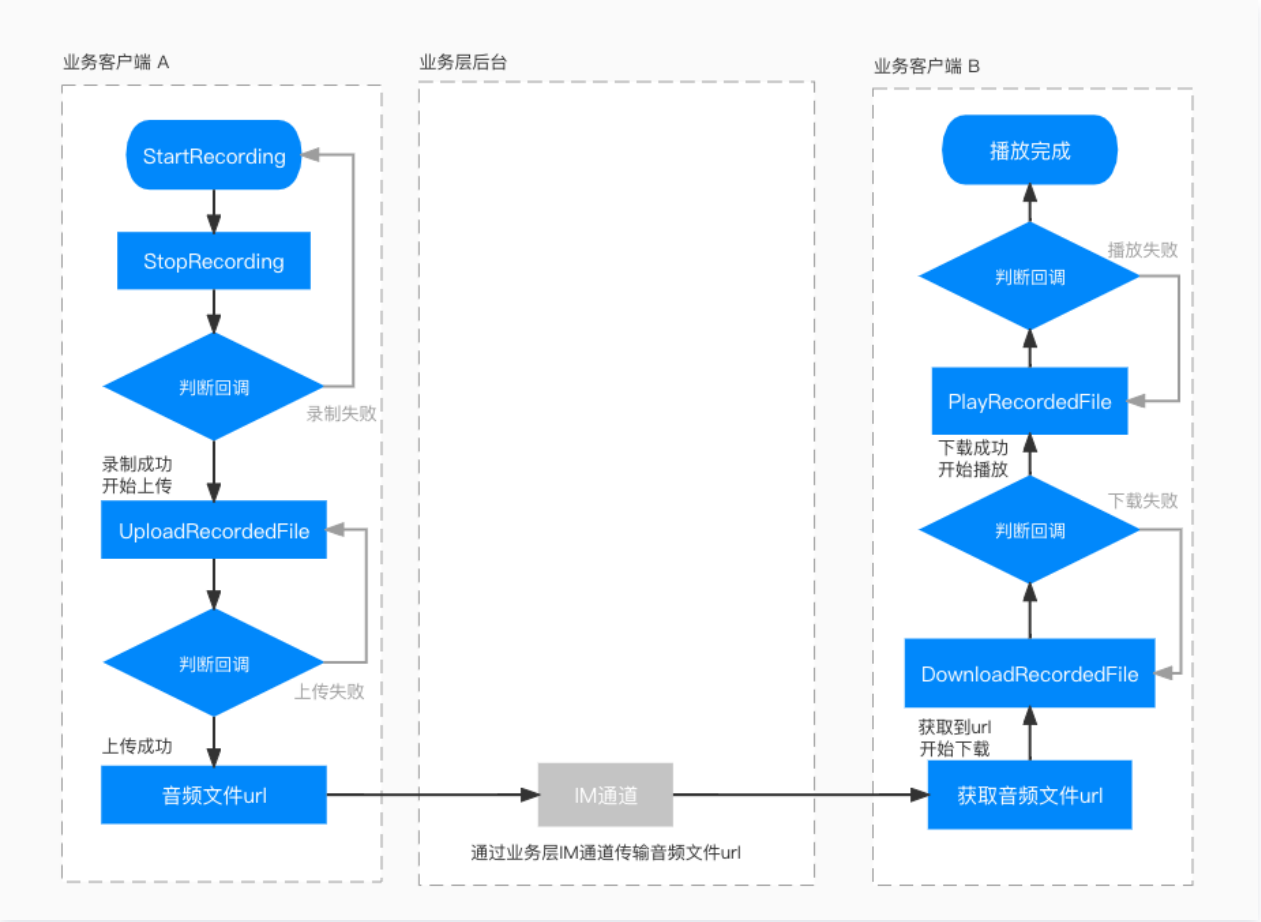
接口原型

```
ITMGContext int Uninit()
```

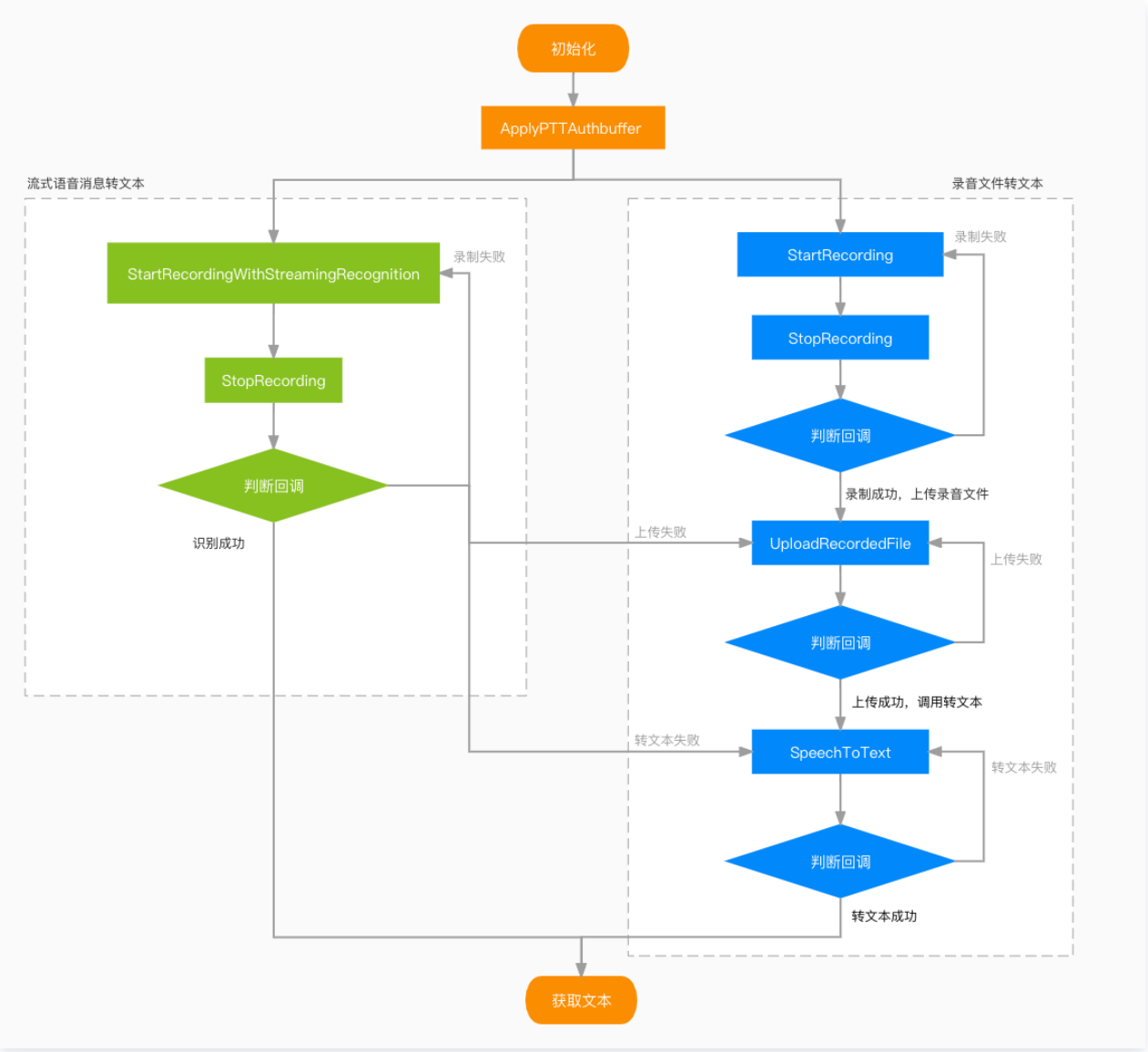
语音消息服务及转文本服务

> **说明**
> - 转文本服务分录音文件极速转文本以及语音消息流式转文本。
> - 使用语音消息服务不需要进入实时语音房间。
> - 语音消息最大录制时长默认为58秒，最短不能小于1秒。如果需要再加以限制，例如限制为最大录制时长为10秒，请在初始化之后调用 SetMaxMessageLength 接口进行设置。

语音消息服务使用流程


业务客户端 A
业务层后台
业务客户端 B
StartRecording
StopRecording
判断回调
录制失败
录制成功
开始上传
UploadRecordedFile
判断回调
上传失败
上传成功
音频文件url
IM通道
通过业务层IM通道传输音频文件url
播放完成
判断回调
播放失败
PlayRecordedFile
下载成功
开始播放
判断回调
下载失败
DownloadRecordedFile
获取到url
开始下载
获取音频文件url

转文本服务使用流程



接口	接口含义
GenAuthBuffer	生成本地鉴权
ApplyPTTAuthbuffer	鉴权初始化
SetMaxMessageLength	限制最大语音信息时长

生成本地鉴权

生成 AuthBuffer，用于相关功能的加密和鉴权，如正式发布请使用后台部署密钥，后台部署请参见 鉴权密钥 。

接口原型

```
int  QAVSDK_AuthBuffer_GenAuthBuffer(unsigned int dwSdkAppID, const char* strRoomID, const char*
strOpenID,
    const char* strKey, unsigned char* strAuthBuffer, unsigned int bufferLength);
```

参数	类型	含义

dwSdkAppID	int	来自腾讯云控制台的 AppId 号码。
strRoomID	const char*	填 null 或者空字符串。
strOpenID	const char*	用户标识。与 Init 时候的 openID 相同。
strKey	const char*	来自腾讯云 控制台 的权限密钥。
strAuthBuffer	const char*	返回的 authbuff。
bufferLength	int	传入的 authbuff 长度，建议为500。

应用鉴权

生成鉴权信息后，将鉴权赋值到 SDK 中。

接口原型

```
ITMGPTT virtual int ApplyPTTAuthbuffer(const char* authBuffer, int authBufferLen)
```

参数	类型	含义
authBuffer	const char*	鉴权
authBufferLen	int	鉴权长度

示例代码

```
ITMGContextGetInstance()->GetPTT()->ApplyPTTAuthbuffer(authBuffer,authBufferLen);
```

限制最大语音信息时长

限制最大语音消息的长度，最大支持58秒。

接口原型

```
ITMGPTT virtual int SetMaxMessageLength(int msTime)
```

参数	类型	含义
msTime	int	语音时长，单位 ms，区间为 1000 < msTime < = 58000

示例代码

```
int msTime = 10000;
ITMGContextGetInstance()->GetPTT()->SetMaxMessageLength(msTime);
```

流式语音识别

语音消息及转文字相关接口

接口	接口含义
StartRecordingWithStreamingRecognition	启动流式录音

StopRecording	停止录音

启动流式语音识别

此接口用于启动流式语音识别，同时在回调中会有实时的语音转文字返回，可以指定语言进行识别，也可以将语音中识别到的信息翻译成指定的语言返回。**停止录音调用** 停止录制 。

接口原型

```
ITMGPTT virtual int StartRecordingWithStreamingRecognition(const char* filePath)
ITMGPTT virtual int StartRecordingWithStreamingRecognition(const char* filePath,const char*
translateLanguage,const char* translateLanguage)
```

参数	类型	含义
filePath	const char*	存放的语音路径
speechLanguage	const char*	识别成指定文字的语言参数，参数请参见 语言参数参考列表
translateLanguage	const char*	翻译成指定文字的语言参数，参数请参见 语言参数参考列表

示例代码

```
ITMGContextGetInstance()->GetPTT()->StartRecordingWithStreamingRecognition(filePath,"cmn-Hans-
CN","cmn-Hans-CN");
```

> **注意**
> 翻译会收取额外费用，请参见 购买指南 。

流式语音识别的回调

启动流式语音识别后，需要在 OnEvent 通知中监听回调消息，事件消息分为以下两个：

- `ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_COMPLETE` 是在停止录制并完成识别后才返回文字，相当于一段话说完才会返回识别的文字。
- `ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_IS_RUNNING` 是在录音过程中就会实时返回识别到的文字，相当于边说话边返回识别到的文字。

根据需求在 OnEvent 通知中对相应事件消息进行判断。传递的参数包含以下四个信息。

消息名称	含义
result	用于判断流式语音识别是否成功的返回码
text	语音转文字识别的文本
file_path	录音存放的本地地址
file_id	录音在后台的 url 地址，录音在服务器存放90天

> **注意**
> 监听 `ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_IS_RUNNING` 消息时，file_id 为空。

错误码

错误码	含义	处理方式
32775	流式语音转文本失败，但是录音成功	调用 UploadRecordedFile 接口上传录音，再调用 SpeechToText 接口进行语音转文字操作
32777	流式语音转文本失败，但是录音成功，上传成功	返回的信息中有上传成功的后台 url 地址，调用 SpeechToText 接口进行语音转文字操作
32786	流式语音转文本失败	在流式录制状态当中，请等待流式录制接口执行结果返回
32787	转文本成功，文本翻译服务未开通	需要在控制台开通文本翻译服务
32788	转文本成功，文本翻译语言参数不支持	重新检查传入参数

如果出现 4098 错误码，请参见 常见问题文档 进行解决。

示例代码

```
void TMGTestScene::OnEvent(ITMG_MAIN_EVENT_TYPE eventType,const char* data){
                switch (eventType) {
                    case ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_COMPLETE:
                    {
                        HandleSTREAM2TEXTComplete(data,true);
                        break;
                        }
                    ...
                            case ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_IS_RUNNING:
                    {
                        HandleSTREAM2TEXTComplete(data, false);
                        break;
                    }
                }
}

void CTMGSDK_For_AudioDlg::HandleSTREAM2TEXTComplete(const char* data, bool isComplete)
{
                    std::string strText = "STREAM2TEXT: ret=";
                    strText += data;
                    m_EditMonitor.SetWindowText(MByteToWChar(strText).c_str());
                    Json::Reader reader;
                    Json::Value root;
                    bool parseRet = reader.parse(data, root);
                    if (!parseRet) {
                        ::SetWindowText(m_EditInfo.GetSafeHwnd(),MByteToWChar(std::string("parse
result Json error")).c_str());
                    }
                    else
                    {
                        if (isComplete) {
                            ::SetWindowText(m_EditUpload.GetSafeHwnd(),
MByteToWChar(root["file_id"].asString()).c_str());
                        }
```

```
                        else {
                            std::string isruning = "STREAMINGRECOGNITION_IS_RUNNING";
                            ::SetWindowText(m_EditUpload.GetSafeHwnd(),
MByteToWChar(isruning).c_str());
                        }
                    }
}
```

语音消息录制

录制的流程为：录音 > 停止录音 > 录音回调返回 > 启动下一次录音。

语音消息及转文字相关接口

接口	接口含义
StartRecording	启动录音
PauseRecording	暂停录音
ResumeRecording	恢复录音
StopRecording	停止录音
CancelRecording	取消录音

启动录音

此接口用于启动录音。

接口原型

```
ITMGPTT virtual int StartRecording(const char* fileDir)
```

参数	类型	含义
fileDir	const char*	存放的语音路径

示例代码

```
char buffer[256]={0};
snprintf(buffer, sizeof(buffer), "%sunreal_ptt_local.file", getFilePath().c_str());
ITMGContextGetInstance()->GetPTT()->StartRecording(buffer);
```

停止录音

此接口用于停止录音。此接口为异步接口，停止录音后会有录音完成回调，成功之后录音文件才可用。

接口原型

```
ITMGPTT virtual int StopRecording()
```

示例代码

```
ITMGContextGetInstance()->GetPTT()->StopRecording();
```

启动录音的回调

启动录音的结果会通过回调返回。
停止录音调用StopRecording。停止录音后才有启动录音的回调。

参数	类型	含义
result	int32	当 code 为 0 时，录制完成
filepath	FString	录制的存放地址，必须是可以访问到的路径，不可将 fileid 作为路径

错误码

错误码值	原因	建议方案
4097	参数为空	检查代码中接口参数是否正确
4098	初始化错误	检查设备是否被占用，或者权限是否正常，是否初始化正常
4099	正在录制中	确保在正确的时机使用 SDK 录制功能
4100	没有采集到音频数据	检查麦克风设备是否正常
4101	录音时，录制文件访问错误	确保文件存在，文件路径的合法性
4102	麦克风未授权错误	使用 SDK 需要麦克风权限，添加权限请参考对应引擎或平台的 SDK 工程配置文档
4103	录音时间太短错误	首先，限制录音时长的单位为毫秒，检查参数是否正确；其次，录音时长要1000毫秒以上才能成功录制
4104	没有启动录音操作	检查是否已经调用启动录音接口

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_EVNET_TYPE_PTT_RECORD_COMPLETE:
                    {
                    //进行处理
                    break;
                    }
                }
}
```

暂停录音

此接口用于暂停录音。如需恢复录音请调用接口 ResumeRecording。

接口原型

```
ITMGPTT virtual int PauseRecording()
```

示例代码

```
ITMGContextGetInstance()->GetPTT()->PauseRecording();
```

恢复录音

此接口用于恢复录音。

接口原型

```
ITMGPTT virtual int ResumeRecording()
```

示例代码

```
ITMGContextGetInstance()->GetPTT()->ResumeRecording();
```

取消录音

调用此接口取消录音。**取消之后没有回调。**

接口原型

```
ITMGPTT virtual int CancelRecording()
```

示例代码

```
ITMGContextGetInstance()->GetPTT()->CancelRecording();
```

语音消息上传、下载及播放

接口	接口含义
UploadRecordedFile	上传语音文件
DownloadRecordedFile	下载语音文件
PlayRecordedFile	播放语音
StopPlayFile	停止播放语音
GetFileSize	语音文件的大小
GetVoiceFileDuration	语音文件的时长

上传语音文件

此接口用于上传语音文件。

接口原型

```
ITMGPTT virtual int UploadRecordedFile(const char* filePath)
```

参数	类型	含义
filePath	const char*	上传的语音路径，此路径为本地路径

示例代码

```
ITMGContextGetInstance()->GetPTT()->UploadRecordedFile(filePath);
```

上传语音完成的回调

上传语音完成后，事件消息为 ITMG_MAIN_EVNET_TYPE_PTT_UPLOAD_COMPLETE，在 OnEvent 函数中对事件消息进行判断。传递的参数包含三个信息，result，file_path 和 file_id。

参数	类型	含义
result	int32	当 code 为0时，录制完成
filepath	FString	录制的存放地址
fileid	FString	文件的 url 路径

错误码

错误码值	原因	建议方案
8193	上传文件时，文件访问错误	确保文件存在，文件路径的合法性
8194	签名校验失败错误	检查鉴权密钥是否正确，检查是否有初始化离线语音
8195	网络错误	检查设备网络是否可以正常访问外网环境
8196	获取上传参数过程中网络失败	检查鉴权是否正确，检查设备网络是否可以正常访问外网环境
8197	获取上传参数过程中回包数据为空	检查鉴权是否正确，检查设备网络是否可以正常访问外网环境
8198	获取上传参数过程中回包解包失败	检查鉴权是否正确，检查设备网络是否可以正常访问外网环境
8200	没有设置 appinfo	检查 apply 接口是否有调用，或者入参是否为空

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_EVNET_TYPE_PTT_UPLOAD_COMPLETE:
            {
            //进行处理
            break;
            }
```

```
        }
}
```

下载语音文件

此接口用于下载语音文件。

接口原型

```
ITMGPTT virtual int DownloadRecordedFile(const char* fileId, const char* filePath)
```

参数	类型	含义
fileId	const char*	文件的 url 路径
filePath	const char*	文件的本地保存路径

示例代码

```
ITMGContextGetInstance()->GetPTT()->DownloadRecordedFile(fileID,filePath);
```

下载语音文件完成回调

下载语音完成后，事件消息为 ITMG_MAIN_EVNET_TYPE_PTT_DOWNLOAD_COMPLETE， 在 OnEvent 函数中对事件消息进行判断。
传递的参数包含三个信息，result、file_path 和 file_id。

参数	类型	含义
result	int32	当 code 为0时，录制完成
filepath	FString	录制的存放地址
fileid	FString	文件的 url 路径，录音在服务器存放 90 天

错误码

错误码值	原因	建议方案
12289	下载文件时，文件访问错误	检查文件路径是否合法
12290	签名校验失败	检查鉴权密钥是否正确，检查是否有初始化离线语音
12291	网络存储系统异常	服务器获取语音文件失败，检查接口参数 fileid 是否正确，检查网络是否正常，检查 COS 文件存不存在
12292	服务器文件系统错误	检查设备网络是否可以正常访问外网环境，检查服务器上是否有此文件
12293	获取下载参数过程中，HTTP 网络失败	检查设备网络是否可以正常访问外网环境
12294	获取下载参数过程中，回包数据为空	检查设备网络是否可以正常访问外网环境
12295	获取下载参数过程中，回包解包失败	检查设备网络是否可以正常访问外网环境

12297	没有设置 appinfo	检查鉴权密钥是否正确，检查是否有初始化离线语音

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;
                }
```

播放语音

此接口用于播放语音。

接口原型

```
ITMGPTT virtual int PlayRecordedFile(const char* filePath)
ITMGPTT virtual int PlayRecordedFile(const char* filePath, nt voiceType)
```

参数	类型	含义
filePath	const char*	本地语音文件的路径
voicetype	int	变声类型，请参见 变声接入文档

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

示例代码

```
ITMGContextGetInstance()->GetPTT()->PlayRecordedFile(filePath);
```

播放语音的回调

播放语音的回调，事件消息为 ITMG_MAIN_EVNET_TYPE_PTT_PLAY_COMPLETE， 在 OnEvent 函数中对事件消息进行判断。传递的参数包含两个信息，一个是 result，另一个是 file_path。

参数	类型	含义
code	int	当 code 为0时，播放完成
filepath	FString	录制的存放地址

错误码

错误码值	原因	建议方案
20481	初始化错误	检查设备是否被占用，或者权限是否正常，是否初始化正常

20482	正在播放中，试图打断并播放下一个失败了（正常是可以打断的）	检查代码逻辑是否正确
20483	参数为空	检查代码中接口参数是否正确
20484	内部错误	初始化播放器错误，解码失败等问题产生此错误码，需要结合日志定位问题

示例代码

```
void TMGTestScene::OnEvent(ITMG_MAIN_EVENT_TYPE eventType,const char* data){
    switch (eventType) {
    ...
        else if (eventType == ITMG_MAIN_EVNET_TYPE_PTT_PLAY_COMPLETE) {
            int32 result = JsonObject->GetIntegerField(TEXT("result"));
            FString filepath = JsonObject->GetStringField(TEXT("file_path"));
            onPttPlayFileCompleted(result, filepath);
        }
    }
}
```

停止播放语音

此接口用于停止播放语音。停止播放语音也会有播放完成的回调。

接口原型

```
ITMGPTT virtual int StopPlayFile()
```

示例代码

```
ITMGContextGetInstance()->GetPTT()->StopPlayFile();
```

获取语音文件的大小

通过此接口，获取语音文件的大小。

接口原型

```
ITMGPTT virtual int GetFileSize(const char* filePath)
```

参数	类型	含义
filePath	const char*	语音文件的路径，此路径为本地路径

示例代码

```
ITMGContextGetInstance()->GetPTT()->GetFileSize(filePath);
```

获取语音文件的时长

此接口用于获取语音文件的时长，单位毫秒。

接口原型

```
ITMGPTT virtual int GetVoiceFileDuration(const char* filePath)
```

参数	类型	含义
filePath	const char*	语音文件的路径，此路径为本地路径

示例代码

```
ITMGContextGetInstance()->GetPTT()->GetVoiceFileDuration(filePath);
```

录音文件极速转文本

接口	接口含义
SpeechToText	语音识别成文字

将指定的语音文件识别成文字

此接口用于将指定的语音文件识别成文字。

接口原型

```
ITMGPTT virtual void SpeechToText(const char* fileID)
```

参数	类型	含义
fileID	const char*	语音文件 url

示例代码

```
ITMGContextGetInstance()->GetPTT()->SpeechToText(fileID);
```

将指定的语音文件翻译成文字（指定语言）

此接口可以指定语言进行识别，也可以将语音中识别到的信息翻译成指定的语言返回。

> **注意**
> 翻译会收取额外费用，请参见 购买指南 。

接口原型

```
ITMGPTT virtual int SpeechToText(const char* fileID,const char* speechLanguage)
ITMGPTT virtual int SpeechToText(const char* fileID,const char* speechLanguage,const char*
translateLanguage)
```

参数	类型	含义
fileID	const char*	语音文件 url，录音在服务器存放90天
speechLanguage	const char*	识别出指定文字的语言参数，参数参见 语言参数参考列表

translatelanguage	const char*	翻译成指定文字的语言参数，参数参见 言参数参考列表

示例代码

```
ITMGContextGetInstance()->GetPTT()->SpeechToText(filePath,"cmn-Hans-CN","cmn-Hans-CN");
```

识别回调

将指定的语音文件识别成文字的回调，事件消息为 ITMG_MAIN_EVNET_TYPE_PTT_SPEECH2TEXT_COMPLETE，在 OnEvent 函数中对事件消息进行判断。
传递的参数包含三个信息，result、file_path 和 text，其中 text 为识别的文本。

参数	类型	含义
result	int32	当 code 为0时，录制完成
fileid	FString	语音文件 url，录音在服务器存放90天
text	FString	转换的文本结果

错误码

错误码值	原因	建议方案
32769	内部错误	分析日志，获取后台返回给客户端的真正错误码，并联系后台同事协助解决
32770	网络失败	检查设备网络是否可以正常访问外网环境
32772	回包解包失败	分析日志，获取后台返回给客户端的真正错误码，并联系后台同事协助解决
32774	没有设置 appinfo	检查鉴权密钥是否正确，检查是否有初始化离线语音
32776	authbuffer 校验失败	检查 authbuffer 是否正确
32784	语音转文本参数错误	检查代码中接口参数 fileid 是否为空
32785	语音转文本翻译返回错误	离线语音后台错误，请分析日志，获取后台返回给客户端的真正错误码，并联系后台同事协助解决
32787	转文本成功，文本翻译服务未开通	需要在控制台开通文本翻译服务
32788	转文本成功，文本翻译语言参数不支持	重新检查传入参数

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_EVNET_TYPE_PTT_SPEECH2TEXT_COMPLETE:
            {
```

```
            //进行处理
            break;
            }
        }
}
```

语音消息音量相关接口

接口	接口含义
GetMicLevel	获取实时麦克风音量
SetMicVolume	设置录制音量
GetMicVolume	获取录制音量
GetSpeakerLevel	获取实时扬声器音量
SetSpeakerVolume	设置播放音量
GetSpeakerVolume	获取播放音量

获取语音消息麦克风实时音量

此接口用于获取麦克风实时音量，返回值为 int 类型，值域为0 – 200。

接口原型

```
ITMGPTT virtual int GetMicLevel()
```

示例代码

```
ITMGContext.GetInstance(this).GetPTT().GetMicLevel();
```

设置语音消息录制音量

此接口用于设置离线语音录制音量，值域为0 – 200。

接口原型

```
ITMGPTT virtual int SetMicVolume(int vol)
```

示例代码

```
ITMGContextGetInstance()->GetPTT()->SetMicVolume(100);
```

获取语音消息录制音量

此接口用于获取离线语音录制音量。返回值为 int 类型，值域为0 – 200。

接口原型

```
ITMGPTT virtual int GetMicVolume()
```

示例代码

```
ITMGContextGetInstance()->GetPTT()->GetMicVolume();
```

获取语音消息扬声器实时音量

此接口用于获取扬声器实时音量。返回值为 int 类型，值域为0 – 200。

接口原型

```
ITMGPTT virtual int GetSpeakerLevel()
```

示例代码

```
ITMGContextGetInstance()->GetPTT()->GetSpeakerLevel();
```

设置语音消息播放音量

此接口用于设置离线语音播放音量，值域为0 – 200。

接口原型

```
ITMGPTT virtual int SetSpeakerVolume(int vol)
```

示例代码

```
ITMGContextGetInstance()->GetPTT()->SetSpeakerVolume(100);
```

获取语音消息播放音量

此接口用于获取离线语音播放音量。返回值为 int 类型，值域为0 – 200。

接口原型

```
ITMGPTT virtual int GetSpeakerVolume()
```

示例代码

```
ITMGContextGetInstance()->GetPTT()->GetSpeakerVolume();
```

高级 API

获取版本号

获取 SDK 版本号，用于分析 SDK 使用情况。

接口原型

```
ITMGContext virtual const char* GetSDKVersion()
```

示例代码

```
ITMGContextGetInstance()->GetSDKVersion();
```

设置打印日志等级

用于设置打印日志等级。建议保持默认等级。需要在 Init 之前调用。

接口原型

```
ITMGContext int SetLogLevel(ITMG_LOG_LEVEL levelWrite, ITMG_LOG_LEVEL levelPrint)
```

参数含义

参数	类型	含义
levelWrite	ITMG_LOG_LEVEL	设置写入日志的等级，TMG_LOG_LEVEL_NONE 表示不写入，默认为 TMG_LOG_LEVEL_INFO
levelPrint	ITMG_LOG_LEVEL	设置打印日志的等级，TMG_LOG_LEVEL_NONE 表示不打印，默认为 TMG_LOG_LEVEL_ERROR

ITMG_LOG_LEVEL 说明如下：

ITMG_LOG_LEVEL	含义
TMG_LOG_LEVEL_NONE	不打印日志
TMG_LOG_LEVEL_ERROR	打印错误日志（默认）
TMG_LOG_LEVEL_INFO	打印提示日志
TMG_LOG_LEVEL_DEBUG	打印开发调试日志
TMG_LOG_LEVEL_VERBOSE	打印高频日志

示例代码

```
ITMGContextGetInstance()->SetLogLevel(TMG_LOG_LEVEL_INFO,TMG_LOG_LEVEL_INFO);
```

设置打印日志路径

用于设置打印日志路径。默认路径如下。需要在 Init 之前调用。

平台	路径
Windows	%appdata%\Tencent\GME\ProcessName
iOS	Application/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/Documents
Android	/sdcard/Android/data/xxx.xxx.xxx/files
Mac	/Users/username/Library/Containers/xxx.xxx.xxx/Data/Documents

接口原型

```
ITMGContext virtual int SetLogPath(const char* logDir)
```

参数	类型	含义
logDir	const char*	路径

示例代码

```
cosnt char* logDir = ""//自行设置路径
ITMGContext* context = ITMGContextGetInstance();
context->SetLogPath(logDir);
```

回调消息

消息	含义	Data	例子
ITMG_MAIN_EVENT_TYPE_ENTER_ROOM	进入音频房间消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_EXIT_ROOM	退出音频房间消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_ROOM_DISCONNECT	房间因为网络等原因断开消息	result; error_info	{"error_info":"waiting timeout, please check your network","result":0}
ITMG_MAIN_EVNET_TYPE_USER_UPDATE	房间成员更新消息	user_list; event_id	{"event_id":1,"user_list":["0"]}
ITMG_MAIN_EVENT_TYPE_RECONNECT_START	房间重连开始消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_RECONNECT_SUCCESS	房间重连成功消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_SWITCH_ROOM	快速切换房间消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE	房间状态切换消息	result; error_info; sub_event_type; new_room_type	{"error_info":"","new_room_type":0,"result":0}
ITMG_MAIN_EVENT_TYPE_ROOM_SHARING_START	开始跨房连麦消息	result;	{"result":0}
ITMG_MAIN_EVENT_TYPE_ROOM_SHARING_STOP	跨房连麦停止消息	result;	{"result":0}
ITMG_MAIN_EVENT_TYPE_SPEAKER_DEF	默认扬声器设备修	result; error_info	{"deviceID":"{0.0.0.00000000}.{a4f1e8be-49fa-43e2-b8cf-dd00542b47ae}","deviceName":"扬声器 (Realtek

AULT_DEVICE_CHANGED	改消息		High Definition Audio)","error_info":"","isNewDevice":true,"isUsedDevice":false,"result":0}
ITMG_MAIN_EVENT_TYPE_SPEAKER_NEW_DEVICE	新增扬声器设备消息	result; error_info	{"deviceID":"{0.0.0.00000000}.{a4f1e8be-49fa-43e2-b8cf-dd00542b47ae}","deviceName":"扬声器 (Realtek High Definition Audio)","error_info":"","isNewDevice":true,"isUsedDevice":false,"result":0}
ITMG_MAIN_EVENT_TYPE_SPEAKER_LOST_DEVICE	丢失扬声器设备消息	result; error_info	{"deviceID":"{0.0.0.00000000}.{a4f1e8be-49fa-43e2-b8cf-dd00542b47ae}","deviceName":"扬声器 (Realtek High Definition Audio)","error_info":"","isNewDevice":false,"isUsedDevice":false,"result":0}
ITMG_MAIN_EVENT_TYPE_MIC_NEW_DEVICE	新增麦克风设备消息	result; error_info	{"deviceID":"{0.0.1.00000000}.{5fdf1a5b-f42d-4ab2-890a-7e454093f229}","deviceName":"麦克风 (Realtek High Definition Audio)","error_info":"","isNewDevice":true,"isUsedDevice":true,"result":0}
ITMG_MAIN_EVENT_TYPE_MIC_LOST_DEVICE	丢失麦克风设备消息	result; error_info	{"deviceID":"{0.0.1.00000000}.{5fdf1a5b-f42d-4ab2-890a-7e454093f229}","deviceName":"麦克风 (Realtek High Definition Audio)","error_info":"","isNewDevice":false,"isUsedDevice":true,"result":0}
ITMG_MAIN_EVENT_TYPE_MIC_DEFAULT_DEVICE_CHANGED	默认麦克风设备修改消息	result; error_info	{"deviceID":"{0.0.1.00000000}.{5fdf1a5b-f42d-4ab2-890a-7e454093f229}","deviceName":"麦克风 (Realtek High Definition Audio)","error_info":"","isNewDevice":false,"isUsedDevice":true,"result":0}
ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_QUALITY	房间质量消息	weight; loss; delay	{"weight":5,"loss":0.1,"delay":1}
ITMG_MAIN_EVNET_TYPE_PTT_RECORD_COMPLETE	语音消息录制完成消息	result; file_path	{"file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_UPLOAD_COMPLETE	语音消息上传完成消息	result; file_path;file_id	{"file_id":"","file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_DOWNLOAD_COMPLETE	语音消息下载完成消息	result; file_path;file_id	{"file_id":"","file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_PLAY_COMPLETE	语音消息播放完成消息	result; file_path	{"file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_SPEECH2TEXT_COMPLETE	语音消息极速转文本完成消息	result; text;file_id	{"file_id":"","text":"","result":0}

ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_COMPLETE	语音消息流式转文本完成消息	result;file_path;text;file_id	{"file_id":"","file_path":","text":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_IS_RUNNING	语音消息正在流式转文本中	result;file_path;text;file_id	{"file_id":"","file_path":","text":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_TEXT2SPEECH_COMPLETE	文本转语音完成消息	result;text;file_id	{"file_id":"","text":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_TRANSLATE_TEXT_COMPLETE	文本翻译完成消息	result;text;file_id	{"file_id":"","text":"","result":0}

版权所有：腾讯云计算（北京）有限责任公司

iOS SDK
集成 SDK

最近更新时间：2023-02-21 15:29:46

为方便 iOS 开发者调试和接入腾讯云游戏多媒体引擎产品 API，本文档主要为您介绍如何在 iOS 项目工程中集成 GME SDK。

SDK 准备

1. 请下载相关 Demo 及 SDK，详情请参见 下载指引 。
2. 解压获取到的 SDK 资源。
3. 文件夹内容如下：
   - GMESDK.framework：原生 iOS 开发相关资源。
   - libGMESDK.a：Unity iOS 开发相关资源（Unity 引擎开发者才需关注此文件）。

> **说明**
> SDK 支持在 iOS9.0 及以上系统上运行。

实时语音

最近更新时间：2023-10-13 17:09:53

为方便 iOS 开发者调试和接入腾讯云游戏多媒体引擎客户端 API，本文为您介绍适用于 iOS 实时语音功能的开发接入技术文档。

使用 GME 重要事项

GME 提供实时语音服务、语音消息服务及转文本服务，使用 GME 服务都依赖 Init 和 Poll 等核心接口。

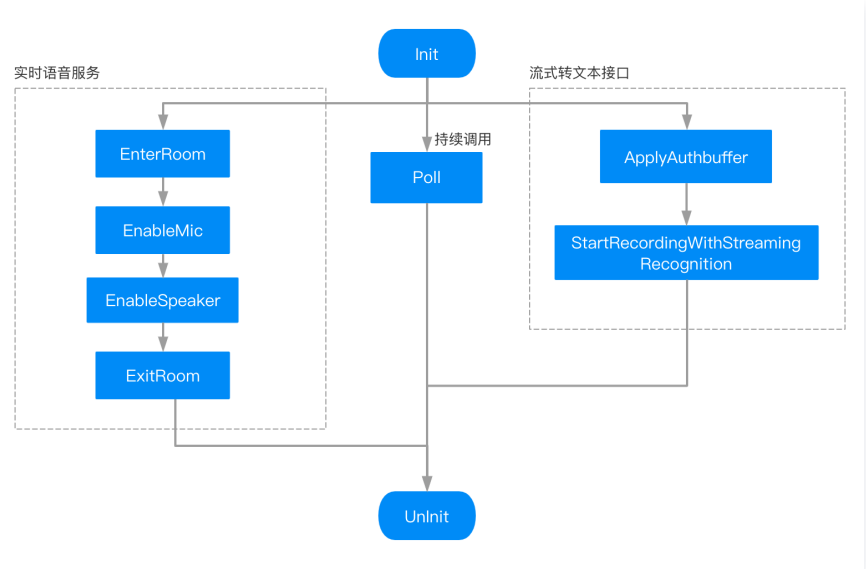
重点提示

- 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、语音消息服务以及转文本服务**。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。
- GME 的接口调用成功后返回值为 QAVError.OK，数值为 0。
- GME 的接口调用要在同一个线程下。
- GME 需要周期性的调用 Poll 接口触发事件回调。
- 错误码详情可参考 错误码 。

接入 SDK

重要步骤

接入 SDK 重要流程如下：



1. 初始化 GME
2. 周期性调用 Poll 触发回调
3. 进入实时语音房间
4. 打开麦克风
5. 打开扬声器
6. 退出语音房间
7. 反初始化 GME

接口类

```
@class ITMGRoom;//房间相关
```

```
@class ITMGAudioCtrl;//音频相关
@class ITMGAudioEffectCtrl;//音效、伴奏相关
```

核心接口

接口	接口含义
InitEngine	初始化 GME
Poll	触发事件回调
Pause	系统暂停
Resume	系统恢复
Uninit	反初始化 GME
SetDefaultAudienceAudioCategory	iOS 设备音频设置

> **注意**
> 如果切换账号，请调用 UnInit 反初始化 SDK。InitEngine 接口调用不会产生计费。

引用头文件

```
#import "GMESDK/TMGEngine.h"
#import "GMESDK/QAVAuthBuffer.h"
```

获取单例

在使用语音功能时，需要首先获取 ITMGContext 对象。

```
+ (ITMGContext*) GetInstance;
```

示例代码

```
//TMGSampleViewController.m
ITMGContext* _context = [ITMGContext GetInstance];
```

设置回调

接口类采用 Delegate 方法用于向应用程序发送回调通知。将回调函数注册给 SDK，用于接收回调的信息。

示例代码

声明采用 ITMGDelegate。

```
@interface TMGDemoViewController ()<ITMGDelegate>{}
ITMGDelegate < NSObject >

//TMGSampleViewController.m
ITMGContext* _context = [ITMGContext GetInstance];
_context.TMGDelegate = [DispatchCenter getInstance];
```

在 OnEvent 中处理接口的回调信息，消息类型参考 ITMG_MAIN_EVENT_TYPE，消息内容为一个字典，解析出接口回调的内容。

接口原型

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

示例代码

```
//TMGRealTimeViewController.m
TMGRealTimeViewController ()< ITMGDelegate >


- (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) {
        // Step 6/11 : Perform the enter room event
        case ITMG_MAIN_EVENT_TYPE_ENTER_ROOM: {
            int result = ((NSNumber *)[data objectForKey:@"result"]).intValue;
            NSString *error_info = [data objectForKey:@"error_info"];

            [self showLog:[NSString stringWithFormat:@"OnEnterRoomComplete:%d msg:(%@)", result,
error_info]];

            if (result == 0) {
                [self updateStatusEnterRoom:YES];
            }
        }
        break;
    }
}

//需要参考 DispatchCenter.h、DispatchCenter.m
```

初始化 SDK

未初始化前，SDK 处于未初始化阶段，需要通过接口 InitEngine 初始化 SDK，才可以使用实时语音服务、语音消息服务及转文本服务，启动后会收集个人信息。调用 InitEngine 接口的线程必须与其他接口在同一线程,建议都在主线程调用接口。

> **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 InitEngine 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

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

参数	类型	含义
sdkAppId	String	来自 腾讯云控制台 的 GME 服务提供的 AppID，获取请参考 语音服务开通指引
OpenId	String	openID 只支持 Int64 类型（转为 const char* 传入）。规则由 App 开发者自行制定，App 内不重复即可。如需使用字符串作为 Openid 传入，可 通过工单 联系开发者。

返回值

返回值	处理
QAV_OK= 0	初始化 SDK 成功
QAV_ERR_SDK_NOT_FULL_UPDATE= 7015	检查 SDK 文件是否完整，建议删除后重新导入 SDK

关于7015错误提示

- 7015错误码是通过 md5 进行判断，在接入过程中若出现此错误，请根据提示检查 SDK 文件是否完整、SDK 文件版本是否一致。
- 出现返回值 AV_ERR_SDK_NOT_FULL_UPDATE 时，此返回值**只有提示作用**，并不会造成初始化失败。
- 由于第三方加固、Unity 打包机制等因素会影响库文件 md5，造成误判，所以**正式发布请在逻辑中忽略此错误**，并尽量不在 UI 中提示。

示例代码

```
_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"];
}
```

触发事件回调

通过在 update 里面周期的调用 Poll 可以触发事件回调。Poll 是 GME 的消息泵，GME 需要周期性的调用 Poll 接口触发事件回调。如果没有调用 Poll ，将会导致整个 SDK 服务运行异常。
可参考 Demo 中的 EnginePollHelper.m 文件。

务必周期性调用 Poll 接口

务必周期性调用 Poll 接口且在主线程调用，以免接口回调异常。

接口原型

```
-(void)Poll;
```

示例代码

```
[[ITMGContext GetInstance] Poll];
```

系统暂停

当系统发生 Pause 事件时，需要同时通知引擎进行 Pause。如果不需要后台播放房间内声音，请调用 Pause 接口暂停整个 GME 服务。

接口原型

```
-(QAVResult)Pause;
```

系统恢复

当系统发生 Resume 事件时，需要同时通知引擎进行 Resume。Resume 接口只恢复实时语音。

接口原型

```
-(QAVResult)Resume;
```

反初始化 SDK

反初始化 SDK，进入未初始化状态。**如果游戏业务侧账号与 openid 是绑定的，那切换游戏账号需要反初始化 GME，再用新的 openid 初始化。**

> **注意**
> 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

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

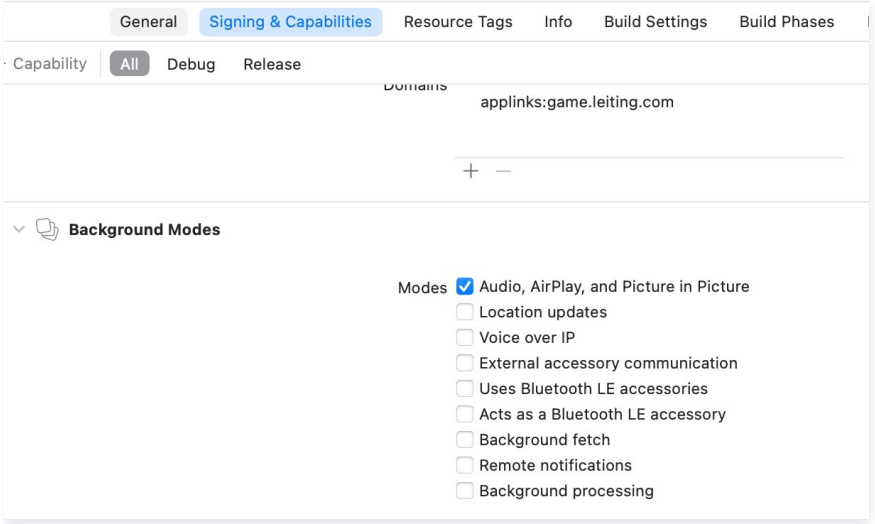
示例代码

```
[[ITMGContext GetInstance] Uninit];
```

iOS 设备音频设置

此接口用于设置后台播放声音，以及 GME 音频不受物理静音按键、锁屏的影响。例如拉起通知中心或者控制中心时候依然可以接收播放 GME 音频。此接口需要在进房前调用。
同时，应用侧有如下两点需要注意：

- 退后台时没有暂停音频引擎的采集和播放（即 PauseAudio）。
- App 的 Info.plist 中，需要至少增加 key:Required background modes，string:App plays audio or streams audio/video using AirPlay。



> **注意**
> 建议开发者调用此接口设置音频。

函数原型

```
-(QAVResult)SetDefaultAudienceAudioCategory:(ITMG_AUDIO_CATEGORY)audioCategory;
```

类型	参数代表	含义
ITMG_CATEGORY_AMBIENT	0	退后台没有声音（默认）
ITMG_CATEGORY_PLAYBACK	1	退后台有声音

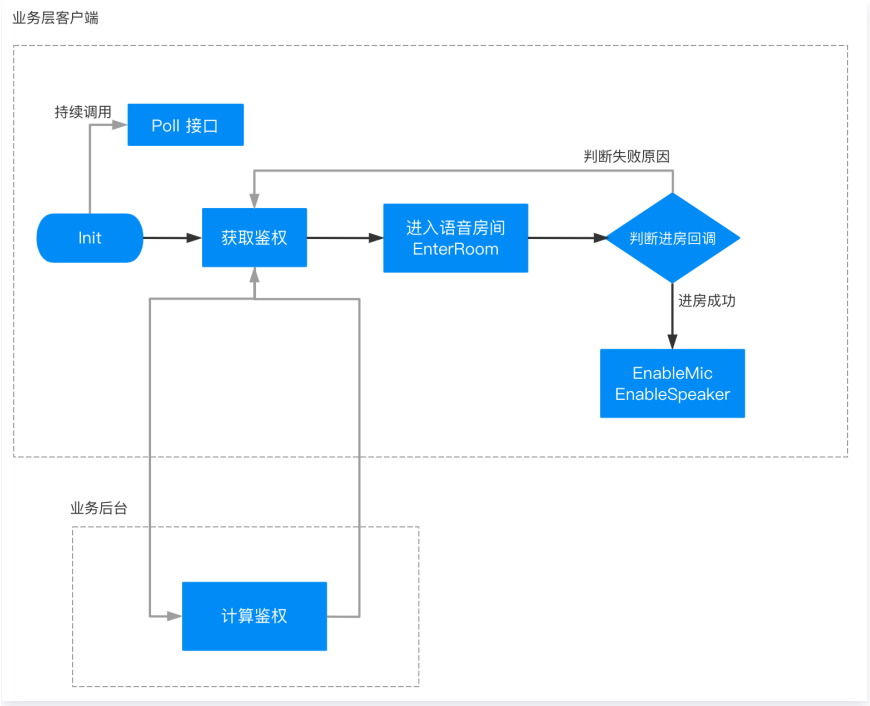
具体实现为修改 kAudioSessionProperty_AudioCategory，相关资料请参见 Apple 官方文档。

示例代码

```
[[ITMGContext GetInstance]SetDefaultAudienceAudioCategory:ITMG_CATEGORY_AMBIENT];
```

实时语音房间相关接口

初始化之后，SDK 调用进房后进去了房间，才可以进行实时语音通话。
使用问题可参考 实时语音相关问题 。



接口	接口含义
GenAuthBuffer	本地鉴权计算
EnterRoom	加入房间
ExitRoom	退出房间
IsRoomEntered	判断是否已经进入房间

SwitchRoom	快速切换房间
StartRoomSharing	跨房连麦

本地鉴权计算

生成 AuthBuffer，用于相关功能的加密和鉴权，如正式发布请使用后台部署密钥，后台部署请参考 鉴权密钥 。

接口原型

```
@interface QAVAuthBuffer : NSObject
+ (NSData*) GenAuthBuffer:(unsigned int)appId roomId:(NSString*)roomId openID:(NSString*)openID
key:(NSString*)key;
+ @end
```

参数	类型	含义
appId	unsigned int	来自腾讯云控制台的 AppId 号码。
roomId	NSString *	房间号，最大支持127字符。
openID	NSString *	用户标识。与 Init 时候的 openID相同。
key	NSString *	来自腾讯云 控制台 的权限密钥。

示例代码

```
#import "GMESDK/QAVAuthBuffer.h"
NSData* authBuffer = [QAVAuthBuffer GenAuthBuffer:SDKAPPID3RD.intValue roomId:_roomId
openID:_openId key:AUTHKEY];
```

加入房间

用生成的鉴权信息进房，加入房间默认不打开麦克风及扬声器。

> ⚠ 注意
> - 加入房间事件回调结果 result 为 0 代表进房成功，进房接口 EnterRoom 返回值为 0 不代表进房成功。
> - 房间的音频类型由第一个进房的人确定，此后房间里有成员修改房间类型，将对此房间所有成员生效。例如第一个进入房间的人使用的房间音频类型是流畅音质，第二个进房的是即使进房时候调用接口的音频类型参数是高清音质，进入房间之后也会变成流畅音质。需要有成员调用 ChangeRoomType 才会修改房间的音频类型。

函数原型

```
-(int)EnterRoom:(NSString*) roomId roomType:(int)roomType authBuffer:(NSData*)authBuffer;
```

参数	类型	含义
roomId	NSString *	房间号，最大支持127字符
roomType	int	房间类型，建议填 ITMG_ROOM_TYPE_FLUENCY。房间音频类型请参考 音质选择 。
authBuffer	NSData *	鉴权码

示例代码

```
[[ITMGContext GetInstance] EnterRoom:_roomId roomType:_roomType authBuffer:authBuffer];
```

加入房间的事件回调

加入房间完成后会发送信息 ITMG_MAIN_EVENT_TYPE_ENTER_ROOM，在 OnEvent 函数中进行判断回调后处理。如果回调为成功，即此时进房成功，开始进行计费。

> **计费问题参考：**
> - 购买指南。
> - 计费相关问题。
> - 使用实时语音后，如果客户端掉线了，是否还会继续计费？

示例代码

回调处理相关参考代码，包括加入房间事件以及断网事件。

```
-(void)OnEvent:(ITMG_MAIN_EVENT_TYPE)eventType data:(NSDictionary *)data{
    NSLog(@"OnEvent:%lu,data:%@",(unsigned long)eventType,data);
    switch (eventType) {
        case ITMG_MAIN_EVENT_TYPE_ENTER_ROOM:
        {
            int result = ((NSNumber*)[data objectForKey:@"result"]).intValue;
            NSString* error_info = [data objectForKey:@"error_info"];
                //收到进房成功事件
        }
            break;
    }
}
```

Data 详情

消息	Data	例子
ITMG_MAIN_EVENT_TYPE_ENTER_ROOM	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_ROOM_DISCONNECT	result; error_info	{"error_info":"waiting timeout, please check your network","result":0}

如果断网，将会有断网的回调提示 `ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVENT_TYPE_ROOM_DISCONNECT`，此时 SDK 会自动进行重连，回调是 `ITMG_MAIN_EVENT_TYPE_RECONNECT_START`，当重连成功时，会有 `ITMG_MAIN_EVENT_TYPE_RECONNECT_SUCCESS` 回调。

错误码

错误码值	原因及建议方案
7006	鉴权失败原因。 • AppID 不存在或者错误 • authbuff 鉴权错误 • 鉴权过期 • OpenId 不符合规范
7007	已经在其它房间

1001	已经在进房过程中，然后又重复了此操作。建议在进房回调返回之前不要再调用进房接口
1003	已经进房了在房间中，又调用一次进房接口
1101	确保已经初始化 SDK，确保 OpenId 是否符合规则，或者确保在同一线程调用接口，以及确保 Poll 接口正常调用

退出房间

通过调用此接口可以退出所在房间。这是一个异步接口，返回值为 AV_OK 的时候代表异步投递成功。如果应用中有退房后立即进房的场景，在接口调用流程上，开发者无需要等待 ExitRoom 的回调 RoomExitComplete 通知，只需直接调用接口。

接口原型

```
-(int)ExitRoom
```

示例代码

```
[[ITMGContext GetInstance] ExitRoom];
```

退出房间事件回调

退出房间完成后会有回调，消息为 ITMG_MAIN_EVENT_TYPE_EXIT_ROOM。

示例代码

```
-(void)OnEvent:(ITMG_MAIN_EVENT_TYPE)eventType data:(NSDictionary *)data{
    NSLog(@"OnEvent:%lu,data:%@",(unsigned long)eventType,data);
    switch (eventType) {
        case ITMG_MAIN_EVENT_TYPE_EXIT_ROOM:
        {
            //收到退房成功事件
        }
            break;
    }
}
```

Data详情

消息	Data	例子
ITMG_MAIN_EVENT_TYPE_EXIT_ROOM	result; error_info	{"error_info":"","result":0}

判断是否已经进入房间

通过调用此接口可以判断是否已经进入房间，返回值为 bool 类型。请勿在进房过程中调用。

接口原型

```
-(BOOL)IsRoomEntered;
```

示例代码

```
[[ITMGContext GetInstance] IsRoomEntered];
```

快速切换房间

调用此接口快速切换实时语音房间。此接口在进房后调用。切换房间后，不重置设备，即如果在此房间已经是打开麦克风状态，在切换房间后也会是打开麦克风状态。
快速切换房间的回调是 ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVENT_TYPE_SWITCH_ROOM，字段是 error_info 以及 result。

接口原型

```
-(int) SwitchRoom:(NSString *)roomID  authBuffer:(NSData*)authBuffer;
```

类型说明

参数	类型	含义
targetRoomID	NSString *	将要进入的房间号
authBuffer	NSData*	用将要进入的房间号生成的新鉴权

回调示例代码

```
- (IBAction)swichRoom:(id)sender {
    NSData* authBuffer = [QAVAuthBuffer GenAuthBuffer:_appId.intValue roomID:_roomIdText.text
openID:_openId key:_key];
    [[[ITMGContext GetInstance]GetRoom]SwitchRoom:_roomIdText.text authBuffer:authBuffer];
}

-(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_SWITCH_ROOM:
        {
            int result = ((NSNumber*)[data objectForKey:@"result"]).intValue;
            NSString* log = nil;
            if (result == QAV_OK) {
                log = [NSString stringWithFormat:@"switch room success."];
            } else {
                log = [NSString stringWithFormat:@"switch room failed."];
            }
            [self showLog:log];
            break;
        }
    }
}
```

跨房连麦

调用此接口进行跨房连麦，此接口在进房后调用。调用接口后，本端可以与目标房间的目标 OpenID 用户进行连麦交流。目标房间与本端房间类型相同才能成功。

场景示例

a 用户在 A 房间中，b 用户在 B 房间中，a 用户可以通过跨房接口与 b 进行通话，A 房间中的用户 c 说话，B 房间的 b 与 d 无法听到；A 房间中的用户 c 只能听到 A 房间的声音以及 B 房间中 b 的声音，B 房间其他人说话 c 无法听到。

接口原型

```
-(int) StartRoomSharing:(NSString *)targetRoomID targetOpenID:(NSString *)targetOpenID
authBuffer:(NSData*)authBuffer;

-(int) StopRoomSharing;
```

类型说明

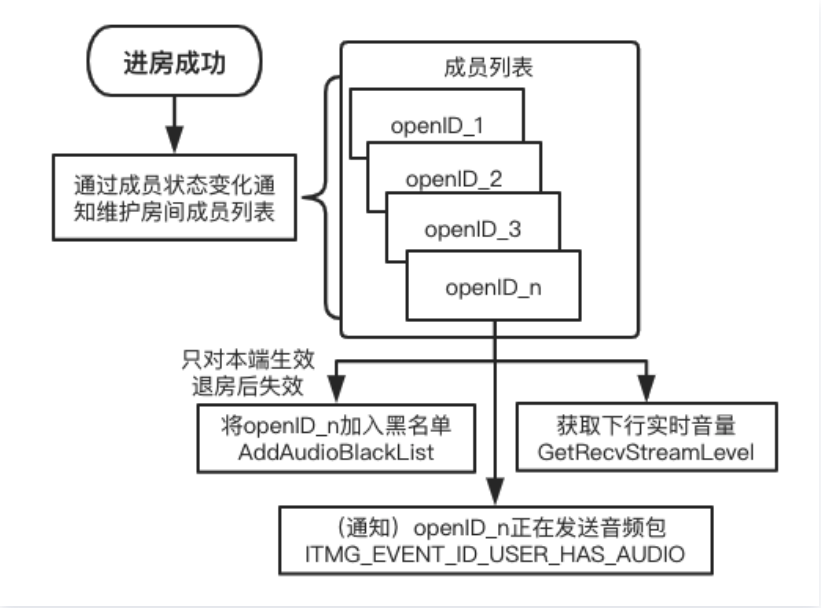
参数	类型	含义
targetRoomID	NSString *	将要连麦的房间号
targetOpenID	NSString *	将要连麦的目标 OpenID
authBuffer	NSData*	保留标志位，只需填 NULL

示例代码

```
- (IBAction)shareRoom:(id)sender {
    if(_shareRoomSwitch.isOn){
        [[[ITMGContext GetInstance]GetRoom]StartRoomSharing:_shareRoomID.text
targetOpenID:_shareOpenID.text authBuffer:NULL];
    }else{
        [[[ITMGContext GetInstance]GetRoom]StopRoomSharing];
    }
}
```

房间内状态维护

此部分接口用于业务层显示说话成员、进退房成员，以及将房间内某成员禁言等功能。



接口/通知	含义

ITMG_MAIN_EVNET_TYPE_USER_UPDATE	成员状态变化通知
AddAudioBlackList	房间中禁言某成员
RemoveAudioBlackList	移除禁言

成员进房、开关麦克风、说话状态通知事件

此接口适用于获取房间中说话的人并在 UI 中展示，以及有人进入、退出语音房间的一个通知。
该事件在状态变化才通知，状态不变化的情况下不通知。如需实时获取成员状态，请在上层收到通知时缓存，事件消息为 ITMG_MAIN_EVNET_TYPE_USER_UPDATE，其中 data 包含两个信息，event_id 及 user_list，在 OnEvent 函数中对事件消息进行判断。
音频事件的通知有一个阈值，超过这个阈值才会发送通知。超过两秒没有收到音频包才通知“有成员停止发送音频包”消息。

event_id	含义	应用侧维护内容
ITMG_EVENT_ID_USER_ENTER	有成员进入房间，返回此时进房的 openid	应用侧维护成员列表
ITMG_EVENT_ID_USER_EXIT	有成员退出房间，返回此时退房的 openid	应用侧维护成员列表
ITMG_EVENT_ID_USER_HAS_AUDIO	有成员发送音频包，返回此时房间内说话的 openid，通过此事件可以判断用户是否说话，并展示声纹效果	应用侧维护通话成员列表
ITMG_EVENT_ID_USER_NO_AUDIO	有成员停止发送音频包，返回此时房间内停止说话的 openid	应用侧维护通话成员列表
ITMG_EVENT_ID_USER_MIC_OPENED	有成员打开麦克风设备（2.9.10及以后版本支持）	应用侧维护通话成员列表
ITME_EVENT_ID_USER_MIC_CLOSED	有成员关闭麦克风设备（2.9.10及以后版本支持）	应用侧维护通话成员列表

示例代码

```
-(void)OnEvent:(ITMG_MAIN_EVENT_TYPE)eventType data:(NSDictionary *)data{
    ITMG_EVENT_ID_USER_UPDATE event_id=((NSNumber*)[data objectForKey:@"event_id"]).intValue;
    NSMutableArray* uses = [NSMutableArray arrayWithArray: [data objectForKey:@"user_list"]];
    NSLog(@"OnEvent:%lu,data:%@",(unsigned long)eventType,data);
    switch (eventType) {
        case ITMG_MAIN_EVNET_TYPE_USER_UPDATE:
        {
        //进行处理
        //开发者对参数进行解析，得到信息 event_id及 user_list
            switch (eventID)
             {
             case ITMG_EVENT_ID_USER_ENTER:
                  //有成员进入房间
                  break;
             case ITMG_EVENT_ID_USER_EXIT:
                  //有成员退出房间
               break;
            case ITMG_EVENT_ID_USER_HAS_AUDIO:
               //有成员发送音频包
               break;
            case ITMG_EVENT_ID_USER_NO_AUDIO:
               //有成员停止发送音频包
               break;
```

```
            case ITMG_EVENT_ID_USER_MIC_OPENED:
                //有成员开启了麦克风
                break;
            case ITME_EVENT_ID_USER_MIC_CLOSED:
                //有成员关闭了麦克风
                break;
             }
        break;
        }
    }
}
```

Data 详情

消息	Data	例子
ITMG_MAIN_EVNET_TYPE_USER_UPDATE	event_id; user_list	{"event_id":0,"user_list":""}

房间中禁言某成员

将某个 ID 加入音频数据黑名单，即不接受某人的语音， 只对本端生效，不会影响其他端。返回值为 0 表示调用成功。例如： A、B、C 都在同一个房间开麦说话：

- 如果 A 设置了 C 的黑名单， 则 A 只能听见 B 的声音。
- B 因为没有设置黑名单， 仍旧可以听见 A 和 C 的声音。
- C 同样因为没有设置黑名单， 可以听见 A 和 B 的声音。

此接口适用于在语音房间中将某用户禁言的场景。

接口原型

```
ITMGContext GetAudioCtrl -(QAVResult)AddAudioBlackList:(NSString*)openID;
```

参数	类型	含义
openId	NSString	需添加黑名单的用户 openid

示例代码

```
[[[ITMGContext GetInstance]GetAudioCtrl ] AddAudioBlackList[id]];
```

移除禁言

将某个 Id 移除音频数据黑名单。返回值为0表示调用成功。

接口原型

```
-(QAVResult)RemoveAudioBlackList:(NSString*)openID;
```

参数	类型	含义
openId	NSString	需移除黑名单的用户 openid

示例代码

```
[[[ITMGContext GetInstance]GetAudioCtrl ] RemoveAudioBlackList[openId]];
```

实时语音采集相关接口

- 初始化 SDK 之后进房，在房间中，才可以调用实时音频语音相关接口。
- 当用户界面单击打开/关闭麦克风/扬声器按钮时，建议采用 EnableMic 以及 EnableSpeaker 接口进行调用。
- 当用户进入实时语音房间，打开或者关闭采集设备，会伴随整个设备（采集及播放）重启，如果此时 App 正在播放背景音乐，那么背景音乐的播放也会被中断。利用控制上下行的方式来实现开关麦克风效果，不会中断播放设备。**具体调用方式为：在进房的时候调用 EnableAudioCaptureDevice(true) && EnableAudioPlayDevice(true) 一次，单击开关麦克风时只调用 EnableAudioSend/Recv 来控制音频流是否发送/接收。**

接口	接口含义
EnableMic	开关麦克风
GetMicState	获取麦克风状态
EnableAudioCaptureDevice	开关采集设备
IsAudioCaptureDeviceEnabled	获取采集设备状态
EnableAudioSend	打开关闭音频上行
IsAudioSendEnabled	获取音频上行状态
GetMicLevel	获取实时麦克风音量
GetSendStreamLevel	获取音频上行实时音量
SetMicVolume	设置麦克风音量
GetMicVolume	获取麦克风音量

开启或关闭麦克风

此接口用来开启关闭麦克风。加入房间默认不打开麦克风及扬声器。**EnableMic = EnableAudioCaptureDevice + EnableAudioSend如果有使用伴奏的情况，请参考 实时语音伴奏流程图 进行调用。**

接口原型

```
-(QAVResult)EnableMic:(BOOL)enable;
```

参数	类型	含义
enable	Boolean	• 如果需要打开麦克风，则传入的参数为 YES • 如果关闭麦克风，则传入的参数为 NO

示例代码

```
//打开麦克风
[[[ITMGContext GetInstance] GetAudioCtrl] EnableMic:YES];
```

麦克风状态获取

此接口用于获取麦克风状态，返回值0为关闭麦克风状态，返回值1为打开麦克风状态。

接口原型

```
-(int)GetMicState;
```

示例代码

```
[[[ITMGContext GetInstance] GetAudioCtrl] GetMicState];
```

开启或关闭采集设备

此接口用来开启/关闭采集设备。加入房间默认不打开设备。

- 只能在进房后调用此接口，退房会自动关闭设备。
- 在移动端，打开采集设备通常会伴随权限申请，音量类型调整等操作。

接口原型

```
-(QAVResult)EnableAudioCaptureDevice:(BOOL)enabled;
```

参数	类型	含义
enabled	BOOL	• 如果需要打开采集设备，则传入的参数为 YES • 如果关闭采集设备，则参数为 NO

示例代码

```
//打开采集设备
[[[ITMGContext GetInstance]GetAudioCtrl ]EnableAudioCaptureDevice:enabled];
```

采集设备状态获取

此接口用于采集设备状态获取。

接口原型

```
-(BOOL)IsAudioCaptureDeviceEnabled;
```

示例代码

```
BOOL IsAudioCaptureDevice = [[[ITMGContext GetInstance] GetAudioCtrl]
IsAudioCaptureDeviceEnabled];
```

打开或关闭音频上行

此接口用于打开/关闭音频上行。如果采集设备已经打开，那么会发送采集到的音频数据。如果采集设备没有打开，那么仍旧无声。采集设备的打开关闭请参见接口 EnableAudioCaptureDevice。

接口原型

```
-(QAVResult)EnableAudioSend:(BOOL)enable;
```

参数	类型	含义
enable	BOOL	• 如果需要打开音频上行，则传入的参数为 YES • 如果关闭音频上行，则传入的参数为 NO

示例代码

```
[[[ITMGContext GetInstance]GetAudioCtrl ]EnableAudioSend:enabled];
```

音频上行状态获取

此接口用于音频上行状态获取。

接口原型

```
-(BOOL)IsAudioSendEnabled;
```

示例代码

```
BOOL IsAudioSend = [[[ITMGContext GetInstance] GetAudioCtrl] IsAudioSendEnabled];
```

获取麦克风实时音量

此接口用于获取麦克风实时音量，返回值为 int 类型。建议20ms获取一次。值域为0 – 100，通过此接口可以获取到麦克风采集到的实时音量情况。

接口原型

```
-(int)GetMicLevel;
```

示例代码

```
[[[ITMGContext GetInstance] GetAudioCtrl] GetMicLevel];
```

获取音频上行实时音量

此接口用于获取自己音频上行实时音量，返回值为 int 类型，取值范围为0 – 100。

接口原型

```
-(int)GetSendStreamLevel();
```

示例代码

```
[[[ITMGContext GetInstance] GetAudioCtrl] GetSendStreamLevel];
```

设置麦克风软件音量

此接口用于设置麦克风的音量。参数 volume 用于设置麦克风的音量，相当于对采集的声音做衰减或增益。

接口原型

```
-(QAVResult)SetMicVolume:(int) volume;
```

参数	类型	含义
volume	Int	取值范围为 0-200，数值为0的时候表示静音，当数值为100的时候表示音量不增不减，默认数值为100。

示例代码

```
[[[ITMGContext GetInstance] GetAudioCtrl] SetMicVolume:100];
```

获取麦克风软件音量

此接口用于获取麦克风的音量。返回值为一个int类型数值，返回值为101代表没调用过接口 SetMicVolume。

接口原型

```
-(int) GetMicVolume;
```

示例代码

```
[[[ITMGContext GetInstance] GetAudioCtrl] GetMicVolume];
```

实时语音播放相关接口

接口	接口含义
EnableSpeaker	开关扬声器
GetSpeakerState	获取扬声器状态
EnableAudioPlayDevice	开关播放设备
IsAudioPlayDeviceEnabled	获取播放设备状态
EnableAudioRecv	打开关闭音频下行
IsAudioRecvEnabled	获取音频下行状态
GetSpeakerLevel	获取实时扬声器音量
GetRecvStreamLevel	获取房间内其他成员下行实时音量
SetSpeakerVolume	设置扬声器音量
GetSpeakerVolume	获取扬声器音量
TrackingVolume	开始监听房间内用户的音量变化
StopTrackingVolume	停止监听房间内用户的音量变化

开启或关闭扬声器

此接口用于开启关闭扬声器。EnableSpeaker = EnableAudioPlayDevice + EnableAudioRecv如果有使用伴奏的情况，请参考 实时语音伴奏流程图 进行调用。

接口原型

```
-(void)EnableSpeaker:(BOOL)enable;
```

参数	类型	含义
enable	Boolean	• 如果需要关闭扬声器，则传入的参数为 NO • 如果打开扬声器，则传入的参数为 YES

示例代码

```
//打开扬声器
[[[ITMGContext GetInstance] GetAudioCtrl] EnableSpeaker:YES];
```

扬声器状态获取

此接口用于扬声器状态获取。返回值0为关闭扬声器状态，返回值1为打开扬声器状态。

函数原型

```
-(int)GetSpeakerState;
```

示例代码

```
[[[ITMGContext GetInstance] GetAudioCtrl] GetSpeakerState];
```

开启或关闭播放设备

此接口用于开启关闭播放设备。

接口原型

```
-(QAVResult)EnableAudioPlayDevice:(BOOL)enabled;
```

参数	类型	含义
enabled	BOOL	• 如果需要关闭播放设备，则传入的参数为 NO • 如果打开播放设备，则传入的参数为 YES

示例代码

```
//打开播放设备
[[[ITMGContext GetInstance]GetAudioCtrl ]EnableAudioPlayDevice:enabled];
```

播放设备状态获取

此接口用于播放设备状态获取。

接口原型

```
-(BOOL)IsAudioPlayDeviceEnabled;
```

示例代码

```
BOOL IsAudioPlayDevice =  [[[ITMGContext GetInstance] GetAudioCtrl] IsAudioPlayDeviceEnabled];
```

打开或关闭音频下行

此接口用于打开/关闭音频下行。如果播放设备已经打开，那么会播放房间里其他人的音频数据。如果播放设备没有打开，那么仍旧无声。播放设备的打开关闭参见接口请参见 EnableAudioPlayDevice。

接口原型

```
-(QAVResult)EnableAudioRecv:(BOOL)enabled;
```

参数	类型	含义
enabled	BOOL	• 如果需要打开音频下行，则传入的参数为 YES • 如果关闭音频下行，则传入的参数为 NO

示例代码

```
[[[ITMGContext GetInstance]GetAudioCtrl ]EnableAudioRecv:enabled];
```

音频下行状态获取

此接口用于音频下行状态获取。

接口原型

```
-(BOOL)IsAudioRecvEnabled;
```

示例代码

```
BOOL IsAudioRecv = [[[ITMGContext GetInstance] GetAudioCtrl] IsAudioRecvEnabled];
```

获取扬声器实时音量

此接口用于获取扬声器实时音量。返回值为 int 类型数值，表示扬声器实时音量。建议20ms获取一次。

接口原型

```
-(int)GetSpeakerLevel;
```

示例代码

```
[[[ITMGContext GetInstance] GetAudioCtrl] GetSpeakerLevel];
```

获取房间内其他成员下行实时音量

此接口用于获取房间内其他成员下行实时音量，返回值为 int 类型，取值范围为0 – 200。

接口原型

```
-(int)GetRecvStreamLevel:(NSString*) openID;
```

参数	类型	含义
openID	NSString	房间其他成员的 openId

示例代码

```
[[[ITMGContext GetInstance] GetAudioCtrl] GetRecvStreamLevel:(NSString*) openId];
```

动态设置房间内某成员音量

此接口用于设置房间内某成员的说话音量大小，此设置只在本端生效。

接口原型

```
-(int) SetSpeakerVolumeByOpenID:(NSString *)openId volume:(int)volume;
```

参数	类型	含义
openId	String *	需要调节音量大小的OpenID
volume	Int	百分比，建议[0-200]，其中100为默认值

获取设置的声音百分比

调用此接口获取SetSpeakerVolumeByOpenID设置的能量值

接口原型

```
-(int) GetSpeakerVolumeByOpenID:(NSString *)openId;
```

返回值

接口返回 OpenID 设置的能量百分比， 默认返回100。

设置扬声器的音量

此接口用于设置扬声器的音量。
参数 volume 用于设置扬声器的音量，当数值为0时，表示静音，当数值为100时，表示音量不增不减，默认数值为100。

函数原型

```
-(QAVResult)SetSpeakerVolume:(int)vol;
```

参数	类型	含义
vol	Int	设置音量，范围0 – 200

示例代码

```
[[[ITMGContext GetInstance] GetAudioCtrl] SetSpeakerVolume:100];
```

获取扬声器的音量

此接口用于获取扬声器的音量。返回值为 int 类型数值，代表扬声器的音量，返回值为101代表没调用过接口 SetSpeakerVolume。Level 是实时音量，Volume 是扬声器的音量，最终声音音量 = Level × Volume %。例如实时音量是数值是100，此时 Volume 的数值是60，那么最终发出来的声音数值也是60。

函数原型

```
-(int)GetSpeakerVolume;
```

示例代码

```
[[[ITMGContext GetInstance] GetAudioCtrl] GetSpeakerVolume];
```

开始监听房间内用户的音量变化

开始监听房间类用户（包括自己）的音量变化，当音量有变化时会触发ITMG_MAIN_EVENT_TYPE_USER_VOLUMES时间回调

接口原型

```
(QAVResult)TrackingVolume:(float)interval;
```

参数	类型	含义
interval	Float	音量变化事件回调的间隔，单位为秒

消息	含义	data
ITMG_MAIN_EVENT_TYPE_USER_VOLUMES	房间内用户音量发生变化	[{"open_id":"abc", "volume":2}, {"open_id":"def", "volume":2}]

示例代码

```
[[[ITMGContext GetInstance] GetAudioCtrl] TrackingVolume:1.0];
```

停止监听房间内用户的音量变化

停止监听房间内用户（包括自己）的音量变化

接口原型

```
-(QAVResult)StopTrackingVolume;
```

示例代码

```
[[[ITMGContext GetInstance] GetAudioCtrl] StopTrackingVolume];
```

高级 API

启动耳返

此接口用于启动耳返，需要 `EnableLoopBack + EnableSpeaker` 才可以听到自己声音。

函数原型

```
-(QAVResult)EnableLoopBack:(BOOL)enable;
```

参数	类型	含义
enable	Boolean	设置是否启动

示例代码

```
[[[ITMGContext GetInstance] GetAudioCtrl] EnableLoopBack:YES];
```

修改用户房间音频类型

此接口用于修改用户房间音频类型，结果参见回调事件，事件类型为 ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE。房间的音频类型由第一个进房的人确定，此后房间里有成员修改房间类型，将对此房间所有成员生效。

函数原型

```
-(int)ChangeRoomType:(int)nRoomType;
```

参数	类型	含义
nRoomType	Int	房间切换成的目标类型，房间音频类型请参见 EnterRoom 接口

示例代码

```
[[[ITMGContext GetInstance]GetRoom ]ChangeRoomType:_roomType];
```

获取用户房间音频类型

此接口用于获取用户房间音频类型，返回值为房间音频类型，返回值为0时代表获取用户房间音频类型发生错误，房间音频类型参考 EnterRoom 接口。

函数原型

```
-(int)GetRoomType;
```

示例代码

```
[[[ITMGContext GetInstance]GetRoom ]GetRoomType];
```

房间类型修改回调

房间类型设置完成后，回调的事件消息为 ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE，返回的参数为 result、error_info 及 new_room_type，new_room_type 代表的信息如下，在 OnEvent 函数中对事件消息进行判断。

事件子类型	代表参数	含义

ITMG_ROOM_CHANGE_EVENT_ENTERROOM	1	表示在进房的过程中，自带的音频类型与房间不符合，被修改为所进入房间的音频类型
ITMG_ROOM_CHANGE_EVENT_START	2	表示已经在房间内，音频类型开始切换（例如调用 ChangeRoomType 接口后切换音频类型 ）
ITMG_ROOM_CHANGE_EVENT_COMPLETE	3	表示已经在房间，音频类型切换完成
ITMG_ROOM_CHANGE_EVENT_REQUEST	4	表示房间成员调用 ChangeRoomType 接口，请求切换房间音频类型

示例代码

```
-(void)OnEvent:(ITMG_MAIN_EVENT_TYPE)eventType data:(NSDictionary *)data {
    NSLog(@"OnEvent:%lu,data:%@",(unsigned long)eventType,data);
    switch (eventType) {
         case ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE:
            NSLog(@"ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE:%@ ",data);
            int result = ((NSNumber*)[data objectForKey:@"result"]).intValue;
            int newRoomType = ((NSNumber*) [data objectForKey:@"new_room_type"]).intValue;
            int subEventType = ((NSNumber*) [data objectForKey:@"sub_event_type"]).intValue;
     }
}
```

Data 详情

消息	Data	例子
ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE	result;error_info;new_room_type;subEventType	{"error_info":"","new_room_type":0,"subEventType":0,"result":0}

房间通话质量监控事件

质量监控事件，在进房后触发，2秒回调一次，事件消息为 ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_QUALITY，返回的参数为 weight、loss 及 delay，代表的信息如下。
此接口适用于监听网络质量，如果用户网络差的话，业务层将通过 UI 提醒用户切换网络。

参数	类型	含义
weight	Int	范围是1 – 50，数值为50是音质评分极好，数值为1是音质评分很差，几乎不能使用，数值为0代表初始值，无含义。一般数值在 30 以下就可以提醒用户网络较差，建议切换网络。
loss	Double	上行丢包率。
delay	Int	音频触达延迟时间（ms）。

获取版本号

获取 SDK 版本号，用于分析。

函数原型

```
-(NSString*)GetSDKVersion;
```

示例代码

```
[[ITMGContext GetInstance] GetSDKVersion];
```

检查麦克风权限

返回麦克风权限状态。

函数原型

```
-(ITMG_RECORD_PERMISSION)CheckMicPermission;
```

参数含义

参数	数值	含义
ITMG_PERMISSION_GRANTED	0	麦克风已授权
ITMG_PERMISSION_Denied	1	麦克风被禁用
ITMG_PERMISSION_NotDetermined	2	尚未弹出权限框向用户申请权限
ITMG_PERMISSION_ERROR	3	接口调用错误

示例代码

```
[[ITMGContext GetInstance] CheckMicPermission];
```

设置打印日志等级

用于设置打印日志等级。建议保持默认等级。需要在 Init 之前调用。

函数原型

```
-(void)SetLogLevel:(ITMG_LOG_LEVEL)levelWrite (ITMG_LOG_LEVEL)levelPrint;
```

参数含义

参数	类型	含义
levelWrite	ITMG_LOG_LEVEL	设置写入日志的等级，TMG_LOG_LEVEL_NONE 表示不写入，默认为 TMG_LOG_LEVEL_INFO
levelPrint	ITMG_LOG_LEVEL	设置打印日志的等级，TMG_LOG_LEVEL_NONE 表示不打印，默认为 TMG_LOG_LEVEL_ERROR

ITMG_LOG_LEVEL	含义
TMG_LOG_LEVEL_NONE	不打印日志
TMG_LOG_LEVEL_ERROR	打印错误日志（默认）
TMG_LOG_LEVEL_INFO	打印提示日志
TMG_LOG_LEVEL_DEBUG	打印开发调试日志
TMG_LOG_LEVEL_VERBOSE	打印高频日志

示例代码

```
[[ITMGContext GetInstance] SetLogLevel:TMG_LOG_LEVEL_INFO TMG_LOG_LEVEL_INFO];
```

设置打印日志路径

用于设置打印日志路径。默认路径为：`Application/********-****-****-****-************/Documents`。需要在 Init 之前调用。

函数原型

```
-(void)SetLogPath:(NSString*)logDir;
```

参数	类型	含义
logDir	NSString	路径

示例代码

```
[[ITMGContext GetInstance] SetLogPath:Path];
```

获取诊断信息

获取音视频通话的实时通话质量的相关信息。该接口主要用来查看实时通话质量、排查问题等，业务侧可以忽略。

函数原型

```
-(NSString*)GetQualityTips;
```

示例代码

```
[[[ITMGContext GetInstance]GetRoom ] GetQualityTips];
```

回调消息

消息列表

消息	消息代表的含义
ITMG_MAIN_EVENT_TYPE_ENTER_ROOM	进入音频房间消息
ITMG_MAIN_EVENT_TYPE_EXIT_ROOM	退出音频房间消息
ITMG_MAIN_EVENT_TYPE_ROOM_DISCONNECT	房间因为网络等原因断开消息
ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE	房间类型变化事件
ITMG_MAIN_EVNET_TYPE_USER_UPDATE	房间成员更新消息
ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_QUALITY	房间质量信息

Data 列表

消息	Data	例子

ITMG_MAIN_EVENT_TYPE_ENTER_ROOM	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_EXIT_ROOM	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_ROOM_DISCONNECT	result; error_info	{"error_info":"waiting timeout, please check your network","result":0}
ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE	result; error_info; sub_event_type; new_room_type	{"error_info":"","new_room_type":0,"result":0}
ITMG_MAIN_EVENT_TYPE_SPEAKER_NEW_DEVICE	result; error_info	{"deviceID":"{0.0.0.00000000}.{a4f1e8be-49fa-43e2-b8cf-dd00542b47ae}","deviceName":"扬声器 (Realtek High Definition Audio)","error_info":"","isNewDevice":true,"isUsedDevice":false,"result":0}
ITMG_MAIN_EVENT_TYPE_SPEAKER_LOST_DEVICE	result; error_info	{"deviceID":"{0.0.0.00000000}.{a4f1e8be-49fa-43e2-b8cf-dd00542b47ae}","deviceName":"扬声器 (Realtek High Definition Audio)","error_info":"","isNewDevice":false,"isUsedDevice":false,"result":0}
ITMG_MAIN_EVENT_TYPE_MIC_NEW_DEVICE	result; error_info	{"deviceID":"{0.0.1.00000000}.{5fdf1a5b-f42d-4ab2-890a-7e454093f229}","deviceName":"麦克风 (Realtek High Definition Audio)","error_info":"","isNewDevice":true,"isUsedDevice":true,"result":0}
ITMG_MAIN_EVENT_TYPE_MIC_LOST_DEVICE	result; error_info	{"deviceID":"{0.0.1.00000000}.{5fdf1a5b-f42d-4ab2-890a-7e454093f229}","deviceName":"麦克风 (Realtek High Definition Audio)","error_info":"","isNewDevice":false,"isUsedDevice":true,"result":0}
ITMG_MAIN_EVNET_TYPE_USER_UPDATE	user_list; event_id	{"event_id":1,"user_list":["0"]}
ITMG_MAIN_EVENT_TYPE_NUMBER_OF_USERS_UPDATE	AllUser; AccUser; ProxyUser	{"AllUser":3,"AccUser":2,"ProxyUser":1}
ITMG_MAIN_EVENT_TYPE_NUMBER_OF_AUDIOSTREAMS_UPDATE	AudioStreams	{"AudioStreams":3}
ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_QUALITY	weight; loss; delay	{"weight":5,"loss":0.1,"delay":1}

语音消息及转文本

最近更新时间：2023-03-20 15:36:30

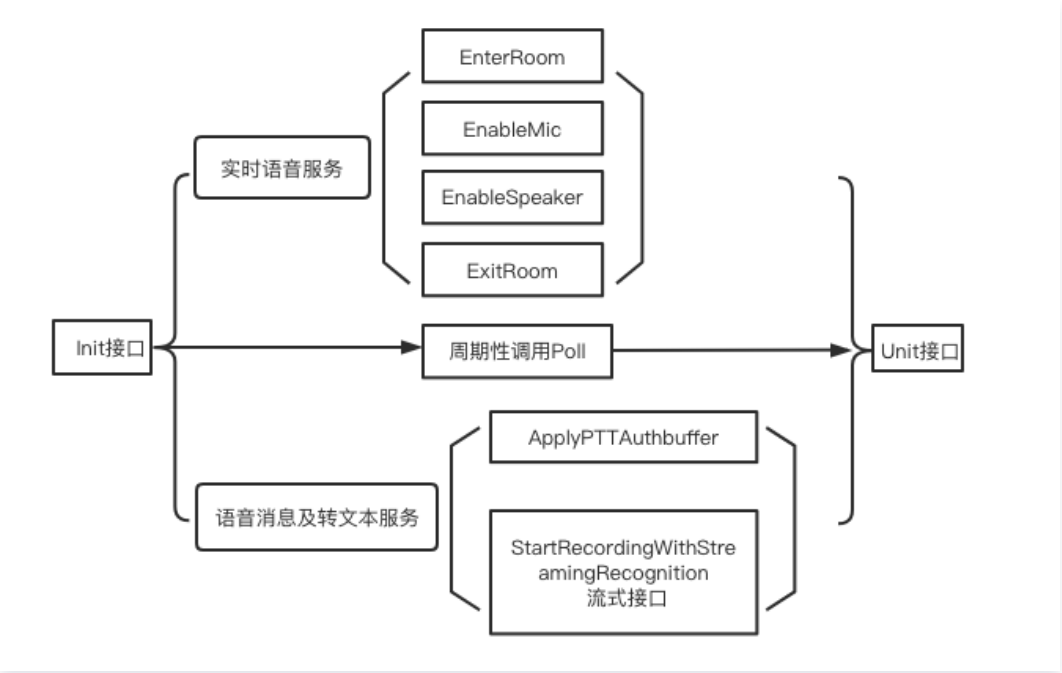
为方便 iOS 开发者调试和接入腾讯云游戏多媒体引擎产品 API，这里向您介绍适用于 iOS 开发的接入技术文档。

> **说明**
> 此文档对应 GME sdk version：2.9。

使用 GME 重要事项

GME 分为两个部分，提供实时语音服务、语音消息及转文本服务，使用这两个服务都依赖 Init 和 Poll 等核心接口。

> **注意**
> 语音转文本相关接口有默认频率限制，限额范围内计费方式请参见 计费文档；若需提升接口频率限额或了解超额计费方式，请联系商务或 提交工单咨询 。
> - 语音消息非流式转文本接口 SpeechToText()：默认单账号限制并发数为10路
> - 语音消息流式转文本接口 StartRecordingWithStreamingRecognition()：默认单账号限制并发数为50路
> - 实时语音流式转文本接口 StartRealTimeASR()：默认单账号限制并发数为50路



> **关于 Init 接口**
> 例如使用了实时语音服务，同时也需要使用语音消息服务，**只需要调用一次 Init 初始化接口**。
> Init 之后不会开始计费，语音消息及转文本服务**收发语音消息**才算作语音消息 DAU。

重要步骤

1. 初始化 GME
2. 周期性调用 Poll 触发回调
3. 鉴权初始化
4. 启动流式语音识别

5. 停止录制
6. 反初始化 GME

重点提示

- GME 使用前请对工程进行配置，否则 SDK 不生效。
- GME 的接口调用成功后返回值为 QAVError.OK，数值为 0。
- GME 的接口调用要在同一个线程下。
- GME 需要周期性的调用 Poll 接口触发事件回调。
- 错误码详情可参考 错误码 。

核心接口

未初始化前，SDK 处于未初始化阶段，需要通过接口 Init 初始化 SDK，才可以使用实时语音服务、语音消息及转文字服务。
在使用 GME 的任何接口之前，都需要先调用 Init 接口。
使用问题可参见 一般性问题 。

接口	接口含义
InitEngine	初始化 GME
Poll	触发事件回调
Pause	系统暂停
Resume	系统恢复
Uninit	反初始化 GME
SetDefaultAudienceAudioCategory	设置设备后台播放声音

引用头文件

```
#import "GMESDK/TMGEngine.h"
#import "GMESDK/QAVAuthBuffer.h"
```

获取单例

在使用语音功能时，需要首先获取 ITMGContext 对象。

```
+ (ITMGContext*) GetInstance;
```

示例代码

```
//TMGSampleViewController.m
ITMGContext* _context = [ITMGContext GetInstance];
```

设置回调

接口类采用 Delegate 方法用于向应用程序发送回调通知。将回调函数注册给 SDK，用于接收回调的信息。

示例代码

声明采用 ITMGDelegate。

```
@interface TMGDemoViewController ()<ITMGDelegate>{}
ITMGDelegate < NSObject >

//TMGSampleViewController.m
ITMGContext* _context = [ITMGContext GetInstance];
_context.TMGDelegate = [DispatchCenter getInstance];
```

在 OnEvent 中处理接口的回调信息，消息类型参考 ITMG_MAIN_EVENT_TYPE，消息内容为一个字典，解析出接口回调的内容。

函数原型

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

示例代码

```
//TMGRealTimeViewController.m
TMGRealTimeViewController ()< ITMGDelegate >


- (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) {
        // Step 6/11 : Perform the enter room event
        case ITMG_MAIN_EVENT_TYPE_ENTER_ROOM: {
            int result = ((NSNumber *)[data objectForKey:@"result"]).intValue;
            NSString *error_info = [data objectForKey:@"error_info"];

            [self showLog:[NSString stringWithFormat:@"OnEnterRoomComplete:%d msg:(%@)", result,
error_info]];

            if (result == 0) {
                [self updateStatusEnterRoom:YES];
            }
        }
        break;
    }
}

//需要参考 DispatchCenter.h、DispatchCenter.m
```

初始化 SDK

- 此接口用于初始化 GME 服务，建议应用侧在应用初始化时候调用，调用此接口不会产生计费。
- **参数 sdkAppID 获取请参见 语音服务开通指引**。
- openID 用于唯一标识一个用户，目前只支持 INT64，规则由 App 开发者自行制定，App 内不重复即可。

> ⚠ **注意**
> 调用 Init 接口的线程必须于其他接口在同一线程。建议都在主线程调用接口。

函数原型

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

参数	类型	含义
sdkAppId	String	来自 腾讯云控制台 的 GME 服务提供的 AppId。
OpenId	String	OpenId 只支持 Int64 类型（转为 string 传入）。

返回值	处理
QAV_OK= 0	初始化 SDK 成功
QAV_ERR_SDK_NOT_FULL_UPDATE= 7015	检查 SDK 文件是否完整，建议删除后重新导入 SDK

出现返回值 AV_ERR_SDK_NOT_FULL_UPDATE 时，此返回值只有提示作用，并不会造成初始化失败。

- 如果在接入过程中提示此错误，请根据提示检查 SDK 文件是否完整、SDK 文件版本是否一致。
- 如果是在导出可执行文件之后出现此返回值，请忽略此错误，并尽量不在 UI 中提示。

示例代码

```
_openId = _userIdText.text;
_appId = _appIdText.text;
[[ITMGContext GetInstance] InitEngine:SDKAPPID openID:_openId];
```

触发事件回调

通过在 update 里面周期的调用 Poll 可以触发事件回调。GME 需要周期性的调用 Poll 接口触发事件回调。如果没有调用 Poll 的话，会导致整个 SDK 服务运行异常。
可参考 Demo 中的 EnginePollHelper.m 文件。

> **注意**
> 务必周期性调用 Poll 接口且在主线程调用，以免接口回调异常。

函数原型

```
-(void)Poll;
```

示例代码

```
[[ITMGContext GetInstance] Poll];
```

系统暂停

当系统发生 Pause 事件时，需要同时通知引擎进行 Pause。
如果需要在退后台的时候暂停语音，可以在退后台的监听代码中调用 Pause 接口，恢复前台的监听事件中调用 Resume 接口。

函数原型

```
-(QAVResult)Pause;
```

系统恢复

当系统发生 Resume 事件时，需要同时通知引擎进行 Resume。Resume 接口只恢复实时语音。

函数原型

```
-(QAVResult)Resume;
```

反初始化 SDK

反初始化 SDK，进入未初始化状态。**切换账号需要反初始化**。

函数原型

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

示例代码

```
[[ITMGContext GetInstance] Uninit];
```

iOS 设备音频设置

此接口用于设置后台播放声音，以及 GME 音频不受物理静音按键、锁屏的影响。例如拉起通知中心或者控制中心时候依然可以接收播放 GME 音频。此接口需要在进房前调用。
同时，应用侧有如下两点需要注意：

- 退后台时没有暂停音频引擎的采集和播放（即 PauseAudio）。
- App 的 Info.plist 中，需要至少增加 key:Required background modes，string:App plays audio or streams audio/video using AirPlay。

> ⚠ **注意**
> 建议开发者调用此接口设置音频。

函数原型

```
-(QAVResult)SetDefaultAudienceAudioCategory:(ITMG_AUDIO_CATEGORY)audioCategory;
```

类型	参数代表	含义
ITMG_CATEGORY_AMBIENT	0	退后台没有声音（默认）
ITMG_CATEGORY_PLAYBACK	1	退后台有声音

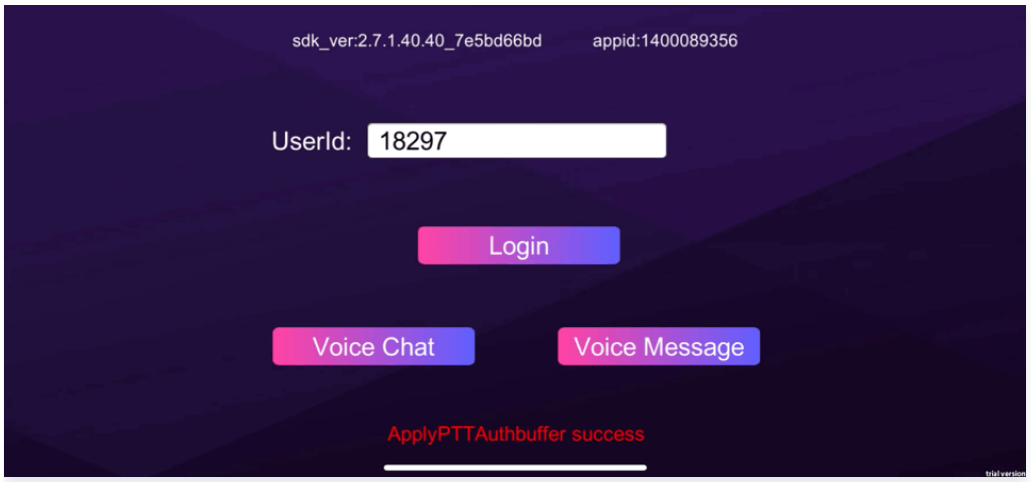
具体实现为修改 kAudioSessionProperty_AudioCategory，相关资料请参见 Apple 官方文档 。

示例代码

```
[[ITMGContext GetInstance]SetDefaultAudienceAudioCategory:ITMG_CATEGORY_AMBIENT];
```

语音消息及转文字

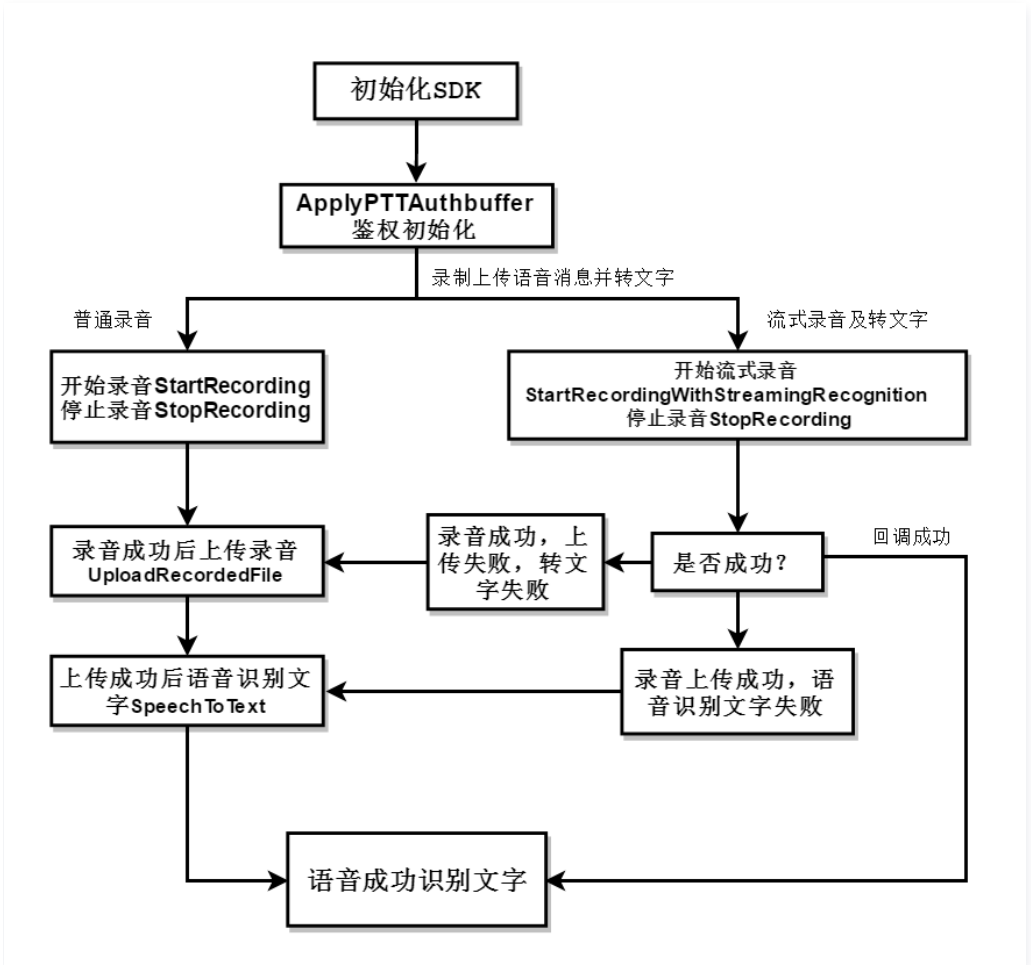
语音消息，录制并发送一段语音消息，同时可以将语音消息转成文字，也可以同时将文字进行翻译。下图演示的是语音消息及转文本服务：



说明

- 建议使用流式语音转文字服务。
- 使用语音消息服务不需要进入实时语音房间。

语音消息及语音转文字流程图



接入语音消息及转文本服务

语音消息及转文字相关接口

接口	接口含义
ApplyPTTAuthbuffer	鉴权初始化
SetMaxMessageLength	限制最大语音信息时长
StartRecording	启动录音
StartRecordingWithStreamingRecognition	启动流式录音
PauseRecording	暂停录音
ResumeRecording	恢复录音
StopRecording	停止录音
CancelRecording	取消录音
GetMicLevel	获取实时麦克风音量
SetMicVolume	设置录制音量
GetMicVolume	获取录制音量
GetSpeakerLevel	获取实时扬声器音量
SetSpeakerVolume	设置播放音量
GetSpeakerVolume	获取播放音量
UploadRecordedFile	上传语音文件
DownloadRecordedFile	下载语音文件
PlayRecordedFile	播放语音
StopPlayFile	停止播放语音
GetFileSize	语音文件的大小
GetVoiceFileDuration	语音文件的时长
SpeechToText	语音识别成文字

最大录制时长
语音消息最大录制时长默认为58秒，最短不能小于1秒。如果需要再加以限制，例如限制为最大录制时长为10秒，请在初始化之后调用 SetMaxMessageLength 接口进行设置。

初始化 SDK

未初始化前，SDK 处于未初始化阶段，需要通过接口 Init 初始化 SDK，才可以使用实时语音及语音消息服务。
使用问题可参见 离线语音相关问题 。

鉴权信息

生成 AuthBuffer，用于相关功能的加密和鉴权，如正式发布请使用后台部署密钥，后台部署请参见 鉴权密钥 。
使用语音消息及转文字服务获取鉴权时，房间号参数必须填 null。

函数原型

```
@interface QAVAuthBuffer : NSObject
+ (NSData*) GenAuthBuffer:(unsigned int)appId roomId:(NSString*)roomId openID:(NSString*)openID
key:(NSString*)key;
+ @end
```

参数	类型	含义
appId	int	来自腾讯云控制台的 AppId 号码。
roomId	NSString	填 null
openID	NSString	用户标识。与 Init 时候的 openID 相同。
key	NSString	来自腾讯云 控制台 的权限密钥。

示例代码

```
#import "GMESDK/QAVAuthBuffer.h"
NSData* authBuffer = [QAVAuthBuffer GenAuthBuffer:SDKAPPID3RD.intValue roomId:_roomId
openID:_openId key:AUTHKEY];
```

鉴权初始化

在初始化 SDK 之后调用鉴权初始化，authBuffer 的获取参见上文实时语音鉴权信息接口 genAuthBuffer。

函数原型

```
public abstract int ApplyPTTAuthbuffer(byte[] authBuffer);
```

参数	类型	含义
authBuffer	NSData*	鉴权

示例代码

```
[[[ITMGContext GetInstance]GetPTT]ApplyPTTAuthbuffer:(NSData *)authBuffer];
```

流式语音识别

启动流式语音识别

此接口用于启动流式语音识别，同时在回调中会有实时的语音转文字返回，可以指定语言进行识别，也可以将语音中识别到的信息翻译成指定的语言返回。**停止录音调用 StopRecording**，停止之后才有回调。

函数原型

```
-(int)StartRecordingWithStreamingRecognition:(NSString *)filePath;
-(int)StartRecordingWithStreamingRecognition:(NSString *)filePath language:
(NSString*)speechLanguage translatelanguage:(NSString*)translateLanguage;
```

参数	类型	含义

filePath	String	存放的语音路径
speechLanguage	String	识别成指定文字的语言参数，参数请参见 语音转文字的语言参数参考列表
translateLanguage	String	翻译成指定文字的语言参数，参数请参见 语音转文字的语言参数参考列表（此参数暂不可用，请填写与 speechLanguage 相同的参数）

示例代码

```
recordfilePath = [docDir stringByAppendingFormat:@"/test_%d.ptt",index++];
[[[ITMGContext GetInstance] GetPTT] StartRecordingWithStreamingRecognition:recordfilePath
language:@"cmn-Hans-CN"];
```

流式语音识别的回调

启动流式语音识别后，需要在回调函数 OnEvent 中监听回调消息，事件消息分为以下两个：

- `ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_COMPLETE` **是在停止录制并完成识别后才返回文字，相当于一段话说完才会返回识别的文字。**
- `ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_IS_RUNNING` **是在录音过程中就会实时返回识别到的文字，相当于边说话边返回识别到的文字。**

根据需求在 OnEvent 函数中对相应事件消息进行判断。传递的参数包含以下四个信息。

消息名称	含义
result	用于判断流式语音识别是否成功的返回码
text	语音转文字识别的文本
file_path	录音存放的本地地址
file_id	录音在后台的 url 地址，录音在服务器存放 90 天。fileid 固定字段为 http://gme-v2-

> **注意**
> 监听 `ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_IS_RUNNING` 消息时，file_id 为空。

错误码

错误码	含义	处理方式
32775	流式语音转文本失败，但是录音成功	调用 UploadRecordedFile 接口上传录音，再调用 SpeechToText 接口进行语音转文字操作
32777	流式语音转文本失败，但是录音成功，上传成功	返回的信息中有上传成功的后台 url 地址，调用 SpeechToText 接口进行语音转文字操作
32786	流式语音转文本失败	在流式录制状态当中，请等待流式录制接口执行结果返回

示例代码

```
- (void)OnEvent:(ITMG_MAIN_EVENT_TYPE)eventType data:(NSDictionary*)data
{
    NSNumber *number = [data objectForKey:@"result"];
```

```
    switch (eventType)
    {
        case ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_COMPLETE:
        {
            if (data != NULL &&[[data objectForKey:@"result"] intValue]== 0)
            {
                donwLoadUrlPath = data[@"file_id"];

                recordfilePath = [data objectForKey:@"file_path"];
                _localFileField.text = recordfilePath;

                _donwloadUrlField.text = [data objectForKey:@"file_id"] ;

                UITextField *_audiotoTextField =(UITextField*)objc_getAssociatedObject(self,
[PTT_AUDIO_TO_TEXT UTF8String]);
                _audiotoTextField.text = [data objectForKey:@"text"] ;
            }

        }
            break;
    }
}
```

语音消息录制

录制的流程为：录音->停止录音->录音回调返回->启动下一次录音。

限制最大语音信息时长

限制最大语音消息的长度，最大支持58秒。

函数原型

```
-(QAVResult)SetMaxMessageLength:(int)msTime
```

参数	类型	含义
msTime	int	语音时长，单位 ms，区间为 1000 < msTime < 58000

示例代码

```
[[[ITMGContext GetInstance]GetPTT]SetMaxMessageLength:(int)msTime];
```

启动录音

此接口用于启动录音。需要将录音文件上传后才可以进行语音转文字等操作。**停止录音调用 StopRecording。**

函数原型

```
-(int)StartRecording:(NSString*)filePath;
```

参数	类型	含义
filePath	NSString	存放的语音路径

示例代码

```
recordfilePath =[docDir stringByAppendingFormat:@"/test_%d.ptt",index++];
[[[ITMGContext GetInstance]GetPTT]StartRecording:recordfilePath];
```

停止录音

此接口用于停止录音。此接口为异步接口，停止录音后会有录音完成回调，成功之后录音文件才可用。

函数原型

```
-(QAVResult)StopRecording;
```

示例代码

```
[[[ITMGContext GetInstance]GetPTT]StopRecording];
```

启动录音的回调

录音完成的回调，通过委托传递消息。
停止录音调用StopRecording。停止录音后才有启动录音的回调。
启动录音完成后的回调调用函数 OnEvent，事件消息为 ITMG_MAIN_EVNET_TYPE_PTT_RECORD_COMPLETE，在 OnEvent 函数中对事件消息进行判断。
传递的参数包含两个信息，一个是 result，另一个是 file_path。

错误码

错误码值	原因	建议方案
4097	参数为空	检查代码中接口参数是否正确
4098	初始化错误	检查设备是否被占用，或者权限是否正常，是否初始化正常
4099	正在录制中	确保在正确的时机使用 SDK 录制功能
4100	没有采集到音频数据	检查麦克风设备是否正常
4101	录音时，录制文件访问错误	确保文件存在，文件路径的合法性
4102	麦克风未授权错误	使用 SDK 需要麦克风权限，添加权限请参考对应引擎或平台的 SDK 工程配置文档
4103	录音时间太短错误	首先，限制录音时长的单位为毫秒，检查参数是否正确；其次，录音时长要1000毫秒以上才能成功录制
4104	没有启动录音操作	检查是否已经调用启动录音接口

示例代码

```
-(void)OnEvent:(ITMG_MAIN_EVENT_TYPE)eventType data:(NSDictionary *)data{
    NSLog(@"OnEvent:%lu,data:%@",(unsigned long)eventType,data);
    switch (eventType) {
        case ITMG_MAIN_EVNET_TYPE_PTT_RECORD_COMPLETE:
        {
        //录音回调
```

```
        }
            break;
    }
}
```

暂停录音

此接口用于暂停录音。如需恢复录音请调用接口 ResumeRecording。

函数原型

```
-(int)PauseRecording;
```

示例代码

```
[[[ITMGContext GetInstance]GetPTT]PauseRecording];
```

恢复录音

此接口用于恢复录音。

函数原型

```
-(int)ResumeRecording;
```

示例代码

```
[[[ITMGContext GetInstance]GetPTT]ResumeRecording];
```

取消录音

调用此接口取消录音。取消之后没有回调。

函数原型

```
-(QAVResult)CancelRecording;
```

示例代码

```
[[[ITMGContext GetInstance]GetPTT]CancelRecording];
```

获取语音消息麦克风实时音量

此接口用于获取麦克风实时音量，返回值为 int 类型，值域为0 – 200。

> **说明**
> 此接口不同于实时语音接口，此接口在 ITMGPTT 下。

函数原型

```
-(QAVResult)GetMicLevel;
```

示例代码

```
[[[ITMGContext GetInstance]GetPTT]GetMicLevel];
```

设置语音消息录制音量

此接口用于设置离线语音录制音量，值域为0 – 200。

> **说明**
> 此接口不同于实时语音接口，此接口在 ITMGPTT 下。

函数原型

```
-(QAVResult)SetMicVolume:(int) volume;
```

示例代码

```
[[[ITMGContext GetInstance]GetPTT]SetMicVolume:100];
```

获取语音消息录制音量

此接口用于获取离线语音录制音量。返回值为 int 类型，值域为0 – 200。

> **说明**
> 此接口不同于实时语音接口，此接口在 ITMGPTT 下。

函数原型

```
-(int)GetMicVolume;
```

示例代码

```
[[[ITMGContext GetInstance]GetPTT]GetMicVolume];
```

获取语音消息扬声器实时音量

此接口用于获取扬声器实时音量。返回值为 int 类型，值域为0 – 200。

> **说明**
> 此接口不同于实时语音接口，此接口在 ITMGPTT 下。

函数原型

```
-(QAVResult)GetSpeakerLevel;
```

示例代码

```
[[[ITMGContext GetInstance]GetPTT]GetSpeakerLevel];
```

设置语音消息播放音量

此接口用于设置语音消息播放音量，值域为0 – 200。

> **说明**
> 此接口不同于实时语音接口，此接口在 ITMGPTT 下。

函数原型

```
-(QAVResult)SetSpeakerVolume:(int)volume;
```

示例代码

```
[[[ITMGContext GetInstance]GetPTT]SetSpeakerVolume:100];
```

获取语音消息播放音量

此接口用于获取语音消息播放音量。返回值为 int 类型，值域为0 – 200。

> **说明**
> 此接口不同于实时语音接口，此接口在 ITMGPTT 下。

函数原型

```
-(int)GetSpeakerVolume;
```

示例代码

```
[[[ITMGContext GetInstance]GetPTT]GetSpeakerVolume];
```

语音消息播放

播放语音

此接口用于播放语音。

函数原型

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

参数	类型	含义
downloadFilePath	NSString	本地语音文件的路径
type	ITMG_VOICE_TYPE	变声类型，请参见 变声接入文档

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

示例代码

```
[[[ITMGContext GetInstance]GetPTT]PlayRecordedFile:path];
```

播放语音的回调

播放语音的回调，事件消息为 ITMG_MAIN_EVNET_TYPE_PTT_PLAY_COMPLETE， 在 OnEvent 函数中对事件消息进行判断。传递的参数包含两个信息，result 和 file_path。

错误码

错误码值	原因	建议方案
20481	初始化错误	检查设备是否被占用，或者权限是否正常，是否初始化正常
20482	正在播放中，试图打断并播放下一个失败了（正常是可以打断的）	检查代码逻辑是否正确
20483	参数为空	检查代码中接口参数是否正确
20484	内部错误	初始化播放器错误，解码失败等问题产生此错误码，需要结合日志定位问题

示例代码

```
-(void)OnEvent:(ITMG_MAIN_EVENT_TYPE)eventType data:(NSDictionary *)data{
    NSLog(@"OnEvent:%lu,data:%@",(unsigned long)eventType,data);
    switch (eventType) {
        case ITMG_MAIN_EVNET_TYPE_PTT_PLAY_COMPLETE:
        {
        //播放语音的回调
        }
            break;
    }
}
```

停止播放语音

此接口用于停止播放语音。停止播放语音也会有播放完成的回调。

函数原型

```
-(int)StopPlayFile;
```

示例代码

```
[[[ITMGContext GetInstance]GetPTT]StopPlayFile];
```

获取语音文件的大小

通过此接口，获取语音文件的大小。

函数原型

```
-(int)GetFileSize:(NSString*)filePath;
```

参数	类型	含义
filePath	NSString	语音文件的路径，此路径为本地路径

示例代码

```
[[[ITMGContext GetInstance]GetPTT]GetFileSize:path];
```

获取语音文件的时长

此接口用于获取语音文件的时长，单位毫秒。

函数原型

```
-(int)GetVoiceFileDuration:(NSString*)filePath;
```

参数	类型	含义
filePath	NSString	语音文件的路径，此路径为本地路径

示例代码

```
[[[ITMGContext GetInstance]GetPTT]GetVoiceFileDuration:path];
```

语音消息上传及下载

上传语音文件

此接口用于上传语音文件。

函数原型

```
-(void)UploadRecordedFile:(NSString*)filePath;
```

参数	类型	含义
filePath	NSString	上传的语音路径，此路径为本地路径

示例代码

```
[[[ITMGContext GetInstance]GetPTT]UploadRecordedFile:path];
```

上传语音完成的回调

上传语音完成后，事件消息为 ITMG_MAIN_EVNET_TYPE_PTT_UPLOAD_COMPLETE，在 OnEvent 函数中对事件消息进行判断。传递的参数包含三个信息，result，file_path 和 file_id。

错误码

错误码值	原因	建议方案

8193	上传文件时，文件访问错误	确保文件存在，文件路径的合法性
8194	签名校验失败错误	检查鉴权密钥是否正确，检查是否有初始化离线语音
8195	网络错误	检查设备网络是否可以正常访问外网环境
8196	获取上传参数过程中网络失败	检查鉴权是否正确，检查设备网络是否可以正常访问外网环境
8197	获取上传参数过程中回包数据为空	检查鉴权是否正确，检查设备网络是否可以正常访问外网环境
8198	获取上传参数过程中回包解包失败	检查鉴权是否正确，检查设备网络是否可以正常访问外网环境
8200	没有设置 appinfo	检查 apply 接口是否有调用，或者入参是否为空

示例代码

```
-(void)OnEvent:(ITMG_MAIN_EVENT_TYPE)eventType data:(NSDictionary *)data{
    NSLog(@"OnEvent:%lu,data:%@",(unsigned long)eventType,data);
    switch (eventType) {
        case ITMG_MAIN_EVNET_TYPE_PTT_UPLOAD_COMPLETE:
        {
            if (data != NULL &&[[data objectForKey:@"result"] intValue]== 0)
            {
                _donwloadUrlField.text = [data objectForKey:@"file_id"] ;
                donwLoadUrlPath = [data objectForKey:@"file_id"] ;
            }
        }
            break;
    }
}
```

下载语音文件

此接口用于下载语音文件。

函数原型

```
-(void)DownloadRecordedFile:(NSString*)fileId downloadFilePath:(NSString*)downloadFilePath
```

参数	类型	含义
fileID	NSString	文件的 url 路径
downloadFilePath	NSString	文件的本地保存路径

示例代码

```
[[[ITMGContext GetInstance]GetPTT]DownloadRecordedFile:fileIdpath downloadFilePath:path];
```

下载语音文件完成回调

下载语音完成后，事件消息为 ITMG_MAIN_EVNET_TYPE_PTT_DOWNLOAD_COMPLETE， 在 OnEvent 函数中对事件消息进行判断。
传递的参数包含三个信息，result、file_path 和 file_id。

错误码

错误码值	原因	建议方案
12289	下载文件时，文件访问错误	检查文件路径是否合法
12290	签名校验失败	检查鉴权密钥是否正确，检查是否有初始化离线语音
12291	网络存储系统异常	服务器获取语音文件失败，检查接口参数 fileid 是否正确，检查网络是否正常，检查 cos 文件存不存在
12292	服务器文件系统错误	检查设备网络是否可以正常访问外网环境，检查服务器上是否有此文件
12293	获取下载参数过程中，HTTP 网络失败	检查设备网络是否可以正常访问外网环境
12294	获取下载参数过程中，回包数据为空	检查设备网络是否可以正常访问外网环境
12295	获取下载参数过程中，回包解包失败	检查设备网络是否可以正常访问外网环境
12297	没有设置 appinfo	检查鉴权密钥是否正确，检查是否有初始化离线语音

示例代码

```
-(void)OnEvent:(ITMG_MAIN_EVENT_TYPE)eventType data:(NSDictionary *)data{
    NSLog(@"OnEvent:%lu,data:%@",(unsigned long)eventType,data);
    switch (eventType) {
        case ITMG_MAIN_EVNET_TYPE_PTT_DOWNLOAD_COMPLETE:
        {
            if (data != NULL &&[[data objectForKey:@"result"] intValue]== 0)
            {
                _audiofileToPlayField.text = [data objectForKey:@"file_path"] ;
                donwLoadLocalPath = [data objectForKey:@"file_path"];
            }
            else
            {
                donwLoadLocalPath = NULL;
            }
        }
            break;
    }
}
```

语音转文字服务

将指定的语音文件识别成文字

此接口用于将指定的语音文件识别成文字。

函数原型

```
-(void)SpeechToText:(NSString*)fileID;
```

参数	类型	含义
fileID	NSString	语音文件 url

示例代码

```
[[[ITMGContext GetInstance]GetPTT]SpeechToText:fileID];
```

将指定的语音文件翻译成文字（指定语言）

此接口可以指定语言进行识别，也可以将语音中识别到的信息翻译成指定的语言返回。

函数原型

```
-(void)SpeechToText:(NSString*)fileID (NSString*)speechLanguage (NSString*)translateLanguage;
```

参数	类型	含义
fileID	NSString*	语音文件 url，录音在服务器存放 90 天
speechLanguage	NSString*	识别出指定文字的语言参数，参数请参见 语音转文字的语言参数参考列表
translateLanguage	NSString*	翻译成指定文字的语言参数，参数请参见 语音转文字的语言参数参考列表（此参数暂时无效，填入参数应与 speechLanguage 一致）

示例代码

```
[[[ITMGContext GetInstance]GetPTT]SpeechToText:fileID speechLanguage:"cmn-Hans-CN"
translateLanguage:"cmn-Hans-CN"];
```

识别回调

将指定的语音文件识别成文字的回调，事件消息为 ITMG_MAIN_EVNET_TYPE_PTT_SPEECH2TEXT_COMPLETE，在 OnEvent 函数中对事件消息进行判断。
传递的参数包含三个信息，result、file_path 和 text，其中 text 为识别的文本。

错误码

错误码值	原因	建议方案
32769	内部错误	分析日志，获取后台返回给客户端的真正错误码，并联系后台同事协助解决。
32770	网络失败	检查设备网络是否可以正常访问外网环境
32772	回包解包失败	分析日志，获取后台返回给客户端的真正错误码，并联系后台同事协助解决。
32774	没有设置 appinfo	检查鉴权密钥是否正确，检查是否有初始化离线语音
32776	authbuffer 校验失败	检查 authbuffer 是否正确
32784	语音转文本参数错误	检查代码中接口参数 fileid 是否为空
32785	语音转文本翻译返回错	离线语音后台错误，请分析日志，获取后台返回给客户端的真正错误码，并联系后台同事协助解

	误	决

示例代码

```
-(void)OnEvent:(ITMG_MAIN_EVENT_TYPE)eventType data:(NSDictionary *)data{
    NSLog(@"OnEvent:%lu,data:%@",(unsigned long)eventType,data);
    switch (eventType) {
        case ITMG_MAIN_EVNET_TYPE_PTT_SPEECH2TEXT_COMPLETE:
        {
            if (data != NULL &&[[data objectForKey:@"result"] intValue]== 0)
            {
                UITextField *_audiotoTextField =(UITextField*)objc_getAssociatedObject(self,
[PTT_AUDIO_TO_TEXT UTF8String]);
                _audiotoTextField.text = [data objectForKey:@"text"] ;
            }
        }
            break;
    }
}
```

高级 API

获取版本号

获取 SDK 版本号，用于分析。

函数原型

```
-(NSString*)GetSDKVersion;
```

示例代码

```
[[ITMGContext GetInstance] GetSDKVersion];
```

检查麦克风权限

返回麦克风权限状态。

函数原型

```
-(ITMG_RECORD_PERMISSION)CheckMicPermission;
```

参数含义

参数	数值	含义
ITMG_PERMISSION_GRANTED	0	麦克风已授权
ITMG_PERMISSION_Denied	1	麦克风被禁用
ITMG_PERMISSION_NotDetermined	2	尚未弹出权限框向用户申请权限
ITMG_PERMISSION_ERROR	3	接口调用错误

示例代码

```
[[ITMGContext GetInstance] CheckMicPermission];
```

设置打印日志等级

用于设置打印日志等级。建议保持默认等级。需要在 Init 之前调用。

函数原型

```
-(void)SetLogLevel:(ITMG_LOG_LEVEL)levelWrite (ITMG_LOG_LEVEL)levelPrint;
```

参数含义

参数	类型	含义
levelWrite	ITMG_LOG_LEVEL	设置写入日志的等级，TMG_LOG_LEVEL_NONE 表示不写入，默认为 TMG_LOG_LEVEL_INFO
levelPrint	ITMG_LOG_LEVEL	设置打印日志的等级，TMG_LOG_LEVEL_NONE 表示不打印，默认为 TMG_LOG_LEVEL_ERROR

ITMG_LOG_LEVEL 说明

ITMG_LOG_LEVEL	含义
TMG_LOG_LEVEL_NONE	不打印日志
TMG_LOG_LEVEL_ERROR	打印错误日志（默认）
TMG_LOG_LEVEL_INFO	打印提示日志
TMG_LOG_LEVEL_DEBUG	打印开发调试日志
TMG_LOG_LEVEL_VERBOSE	打印高频日志

示例代码

```
[[ITMGContext GetInstance] SetLogLevel:TMG_LOG_LEVEL_INFO TMG_LOG_LEVEL_INFO];
```

设置打印日志路径

用于设置打印日志路径。默认路径为：`Application/********-****-****-************/Documents` 。需要在 Init 之前调用。

函数原型

```
-(void)SetLogPath:(NSString*)logDir;
```

参数	类型	含义
logDir	NSString	路径

示例代码

```
[[ITMGContext GetInstance] SetLogPath:Path];
```

回调消息

消息列表

消息	消息代表的含义
ITMG_MAIN_EVNET_TYPE_PTT_RECORD_COMPLETE	PTT 录音完成
ITMG_MAIN_EVNET_TYPE_PTT_UPLOAD_COMPLETE	上传 PTT 完成
ITMG_MAIN_EVNET_TYPE_PTT_DOWNLOAD_COMPLETE	下载 PTT 完成
ITMG_MAIN_EVNET_TYPE_PTT_PLAY_COMPLETE	播放 PTT 完成
ITMG_MAIN_EVNET_TYPE_PTT_SPEECH2TEXT_COMPLETE	语音转文字完成

Data 列表

消息	Data	例子
ITMG_MAIN_EVNET_TYPE_PTT_RECORD_COMPLETE	result; file_path	{"file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_UPLOAD_COMPLETE	result; file_path;file_id	{"file_id":"","file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_DOWNLOAD_COMPLETE	result; file_path;file_id	{"file_id":"","file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_PLAY_COMPLETE	result; file_path	{"file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_SPEECH2TEXT_COMPLETE	result; text;file_id	{"file_id":"","text":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_COMPLETE	result; file_path; text;file_id	{"file_id":"","file_path":","text":"","result":0}

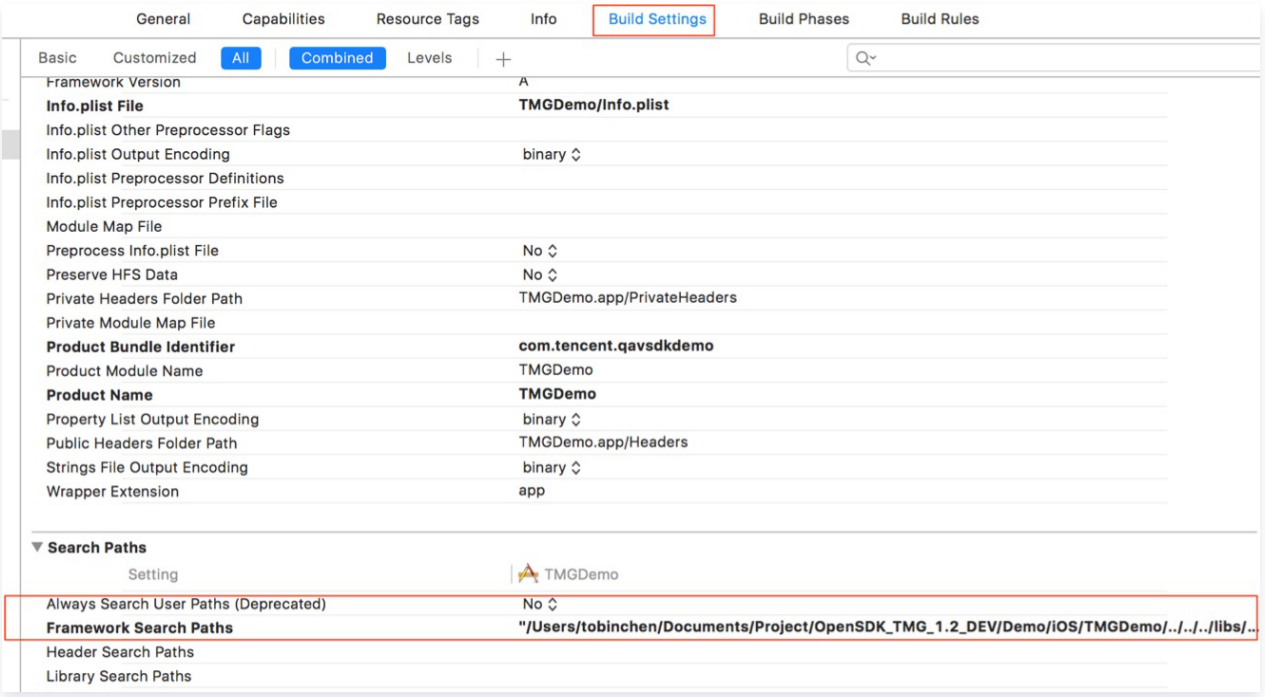
工程导出

最近更新时间：2023-03-20 15:36:30

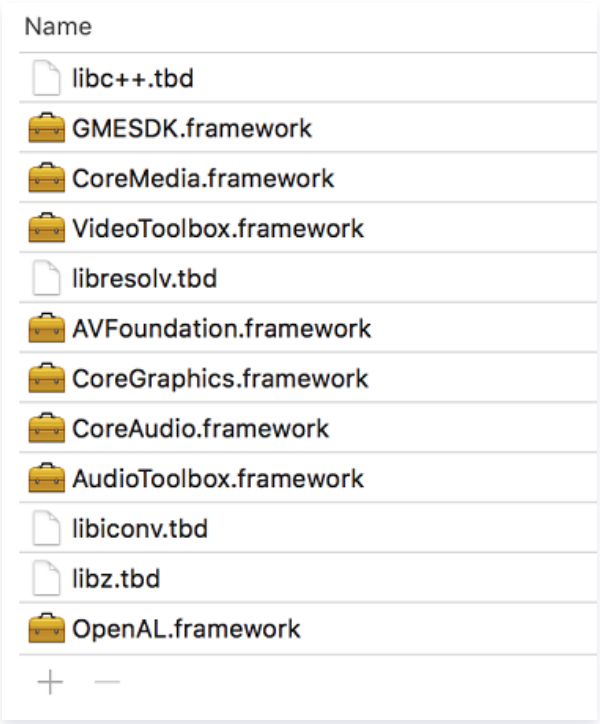
为方便 iOS 开发者调试和接入腾讯云游戏多媒体引擎产品 API，本文档主要为您介绍 iOS 项目工程导出注意事项。

导出配置指引

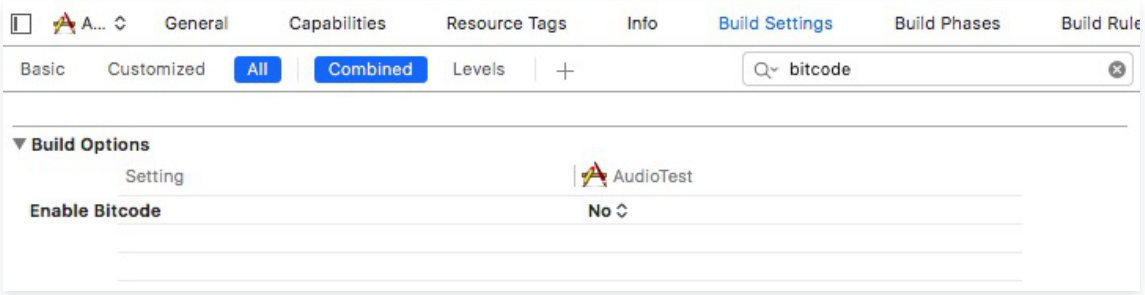
1. 根据实际情况在 **Xcode > Link Binary With Libraries > Build Setting** 里，加入以下依赖库，并设置 Framework Search Paths 指向 SDK 所在目录，如下图所示：



2. 添加依赖库，如下图所示：



3. Bitcode 需要工程依赖的所有类库同时支持，SDK 暂时还不支持 Bitcode，可以先临时关闭。 关闭此设置，只需在 Targets – Build Settings 中搜索 Bitcode 即可，找到相应选项，设置为 NO。



4. 游戏多媒体引擎 iOS 平台所需要的隐私权限如下：
   - Required background modes：允许后台运行（可选）。
   - Microphone Usage Description：允许麦克风权限。

GME 2.9 以上版本

如果接入的SDK 是2.9及以上版本，需要参见 iOS 工程升级指引 文档进行配置。

iOS 工程升级指引

最近更新时间：2023-06-05 16:30:52

操作场景

GME SDK 已升级到2.9版本，如果您将 GME 升级到2.9版本，需要在 Xcode 工程中做以下几个步骤：

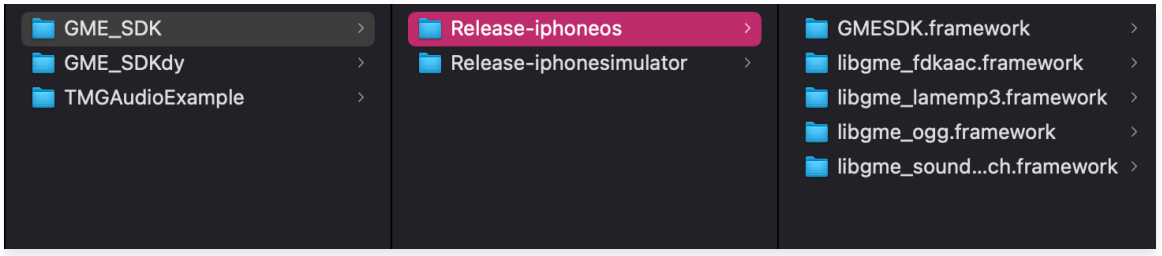
升级步骤

1. 下载 SDK

新版本中将 SDK 进行动态库拆分，多出以下几个文件：

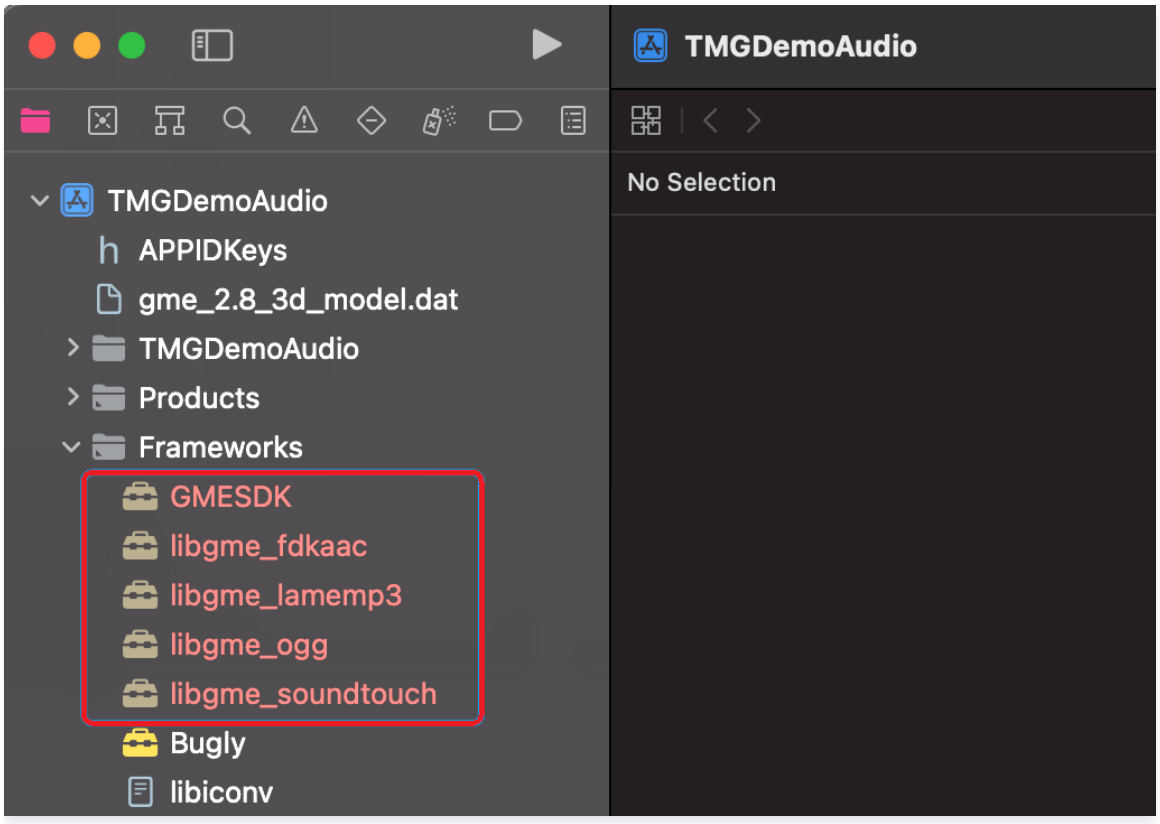
- libgmefdkaac.framework
- libgmeogg.framework
- libgmelamemp3.framework
- libgmesoundtouch.framework

确保下载下来的 SDK 中包含这几个文件。下载后与 GMESDK.framework 放于工程目录下。Release-iphoneos 为真机使用的SDK文件，Release-iphonesimulator 为模拟器使用的 SDK 文件。



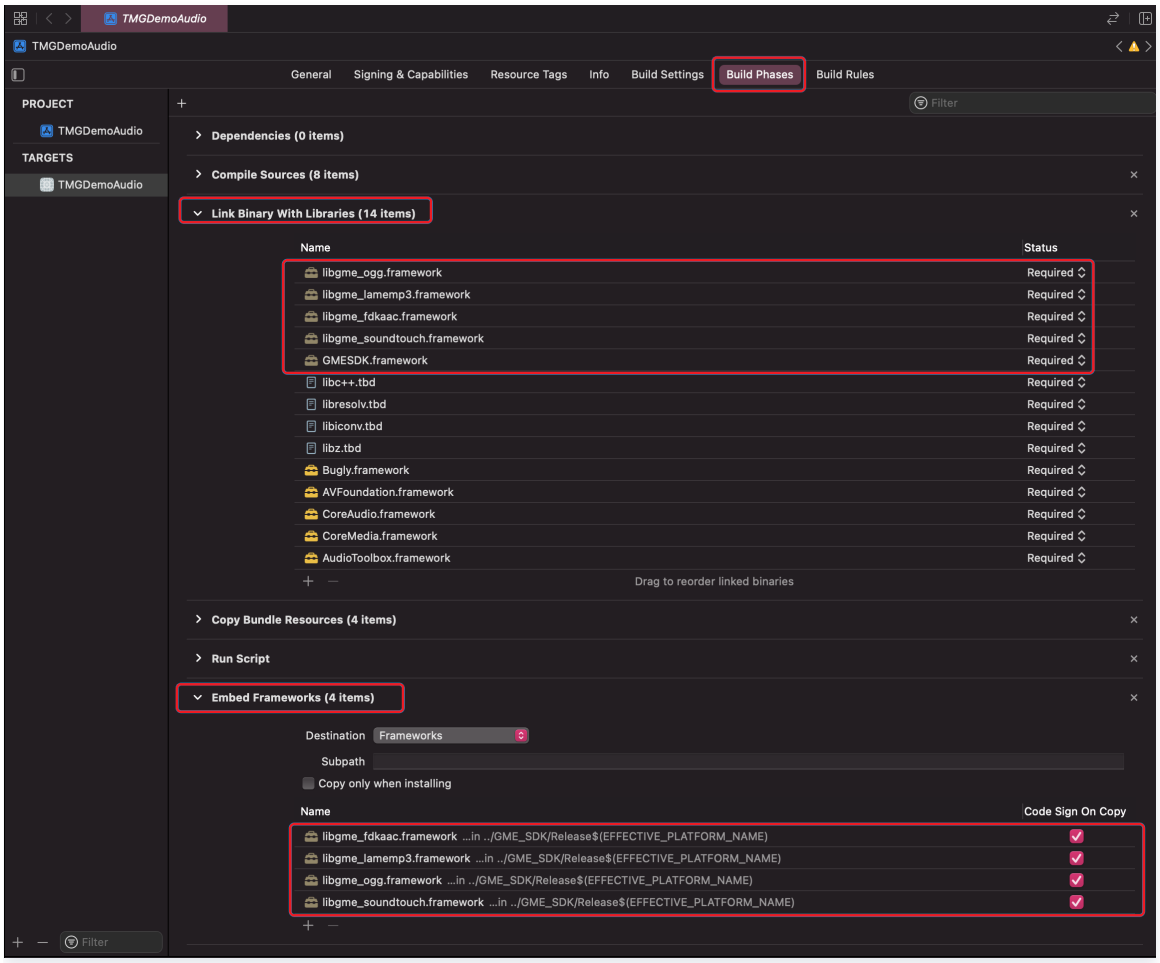
2. 工程导入 SDK

在工程中导入所有的 framework。如下图所示：



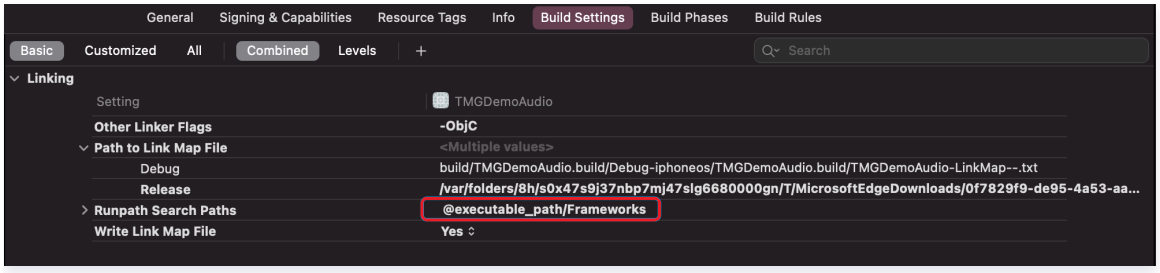
3. 设置 framework 并签名

1. 在 xcode 工程中，单击 **Build Phases**，将 Link Binary With Libraries 展开，将 GME 的所有 framework 导入。
2. 将 Embed Framework 展开，将 GME 的所有 framework 导入，勾选 **Code Sign On Copy**。



4. rpath 改动

需要在 rpath 增加 @executable_path/Frameworks。如果已经增加，便无需再修改。



Android SDK
集成 SDK

最近更新时间：2023-03-29 11:14:10

为方便 Android 开发者调试和接入腾讯云游戏多媒体引擎产品 API，本文档主要为您介绍如何在 Android 项目工程中集成 GME SDK。

SDK 文件准备

1. 请下载相关 Demo 及 SDK，详情请参见 下载指引 。
2. 解压获取到的 SDK 资源。
3. 文件夹内容 libs 为开发 SDK 资源。

> **说明**
> SDK 支持 在 Android 5.0 及以上系统上运行。

配置指引

方法一

1. 将 libs 目录下的 gmesdk.jar 文件复制到 Android 工程的 libs 目录下。
2. **按照工程需求复制相应架构的库文件**，例如工程需要 armeabi-v7a 架构，请将 armeabi-v7a 目录下的库文件拷贝至工程中的 armeabi-v7a 目录下（如果工程没有 armeabi-v7a 目录，请自行创建）。

方法二

将so和jar放在工程的任意文件夹中，并在 surceSets 中指定到存放 so 及 jar 文件的目录。

工程配置

在工程 App 目录下的 build.gradle 中，添加引用库的代码。

```
sourceSets {
        main {
            jniLibs.srcDirs = ['libs']
        }
}
```

实时语音

最近更新时间：2023-10-13 17:09:53

为方便 Android 开发者调试和接入腾讯云游戏多媒体引擎客户端 API，本文为您介绍适用于 Android 实时语音功能的开发接入技术文档。

使用 GME 重要事项

GME 提供实时语音服务、语音消息服务及转文本服务，使用 GME 服务都依赖 Init 和 Poll 等核心接口。

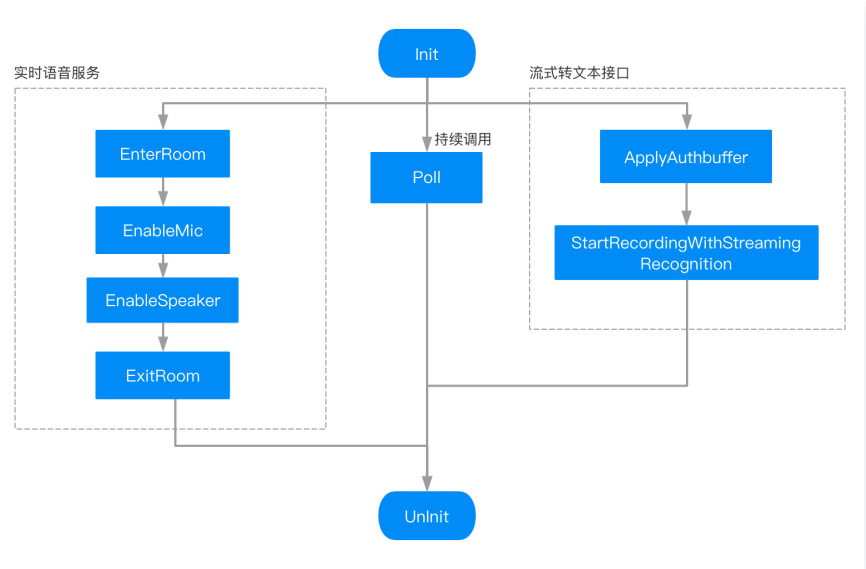
重点提示

- 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、语音消息服务以及转文本服务**。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。
- GME 的接口调用成功后返回值为 QAVError.OK，数值为 0。
- GME 的接口调用要在同一个线程下。
- GME 需要周期性的调用 Poll 接口触发事件回调。
- 错误码详情可参考 错误码 。

接入 SDK

重要步骤

接入 SDK 重要流程如下：



1. 初始化 GME
2. 周期性调用 Poll 触发回调
3. 进入实时语音房间
4. 打开麦克风
5. 打开扬声器
6. 退出语音房间
7. 反初始化 GME

实时语音功能 Android 类

类	含义

ITMGContext	核心接口
ITMGRoom	房间相关接口
ITMGRoomManager	房间管理接口
ITMGAudioCtrl	音频相关接口
ITMGAudioEffectCtrl	音效及伴奏相关接口

核心接口

接口	接口含义
Init	初始化 GME
Poll	触发事件回调
Pause	系统暂停
Resume	系统恢复
Uninit	反初始化 GME

> ⚠ **注意**
> 如果切换账号，请调用 UnInit 反初始化 SDK。Init 接口调用不会产生计费。

获取单例

在使用语音功能时，需要首先获取 ITMGContext 对象。

示例代码

```
com.gme.TMG.ITMGContext
ITMGContext gmeCtx = ITMGContext.GetInstance(this);
```

注册回调

接口类采用 Delegate 方法用于向应用程序发送回调通知。将回调函数注册给 SDK，用于接收回调的信息。

接口原型

```
static public abstract class ITMGDelegate {
    public void OnEvent(ITMG_MAIN_EVENT_TYPE type, Intent data){}
}
```

在构造函数中重写这个回调函数，对回调的参数进行处理。

参数	类型	含义
type	ITMGContext.ITMG_MAIN_EVENT_TYPE	回调的事件类型
data	Intent 消息类型	回调的相关信息，事件数据

示例代码

```
private ITMGContext.ITMGDelegate itmgDelegate = null;
itmgDelegate = new ITMGContext.ITMGDelegate() {
    @Override
    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");
                }
        }
}
```

将回调函数注册给 SDK，要在进房之前设置。

接口原型

```
public abstract int SetTMGDelegate(ITMGDelegate delegate);
```

参数	类型	含义
delegate	ITMGDelegate	SDK 回调函数

示例代码

```
ITMGContext.GetInstance(this).SetTMGDelegate(itmgDelegate);
```

初始化 SDK

未初始化前，SDK 处于未初始化阶段，**需要通过接口 Init 初始化 SDK**，才可以使用实时语音服务、语音消息服务及转文本服务，**启动后会收集个人信息**。调用 Init 接口的线程必须于其他接口在同一线程,建议都在主线程调用接口。

> **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

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

参数	类型	含义
sdkAppId	String	来自 腾讯云控制台 的 GME 服务提供的 AppID，获取请参考 语音服务开通指引 。
openId。	String	openID 只支持 Int64 类型（转为 const char* 传入）。规则由 App 开发者自行制定，App 内不重复即可。如需使用字符串作为 Openid 传入，可 通过工单 联系开发者。

返回值

返回值	处理
QAVError.OK= 0	初始化 SDK 成功

AV_ERR_SDK_NOT_FULL_UPDATE=7015	检查 SDK 文件是否完整，建议删除后重新导入 SDK

关于7015错误提示

- 7015错误码是通过 md5 进行判断，在接入过程中若出现此错误，请根据提示检查 SDK 文件是否完整、SDK 文件版本是否一致。
- 出现返回值 AV_ERR_SDK_NOT_FULL_UPDATE 时，此返回值**只有提示作用**，并不会造成初始化失败。
- 由于第三方加固、Unity 打包机制等因素会影响库文件 md5，造成误判，所以**正式发布请在逻辑中忽略此错误**，并尽量不在 UI 中提示。

示例代码

```
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);
}
```

触发事件回调

通过在 update 里面周期的调用 Poll 可以触发事件回调。Poll 是 GME 的消息泵，GME 需要周期性的调用 Poll 接口触发事件回调。如果没有调用 Poll ，将会导致整个 SDK 服务运行异常。
可参考 Demo 中的 EnginePollHelper.java 文件。

务必周期性调用 Poll 接口

务必周期性调用 Poll 接口且在主线程调用，以免接口回调异常。

接口原型

```
public abstract int Poll();
```

示例代码

```
private Handler mhandler = new Handler();private Runnable mRunnable = new Runnable() {
@Override    public void run() {        if (s_pollEnabled) {            if
(ITMGContext.GetInstance(null) != null)                ITMGContext.GetInstance(null).Poll();
}        mhandler.postDelayed(mRunnable, 33);    }};
```

系统暂停

当系统发生 Pause 事件时，需要同时通知引擎进行 Pause。如果不需要后台播放房间内声音，请调用 Pause 接口暂停整个 GME 服务。

接口原型

```
public abstract int Pause();
```

系统恢复

当系统发生 Resume 事件时，需要同时通知引擎进行 Resume。Resume 接口只恢复实时语音。

接口原型

```
public abstract int Resume();
```

反初始化 SDK

反初始化 SDK，进入未初始化状态。**如果游戏业务侧账号与 openid 是绑定的，那切换游戏账号需要反初始化 GME，再用新的 openid 初始化。**

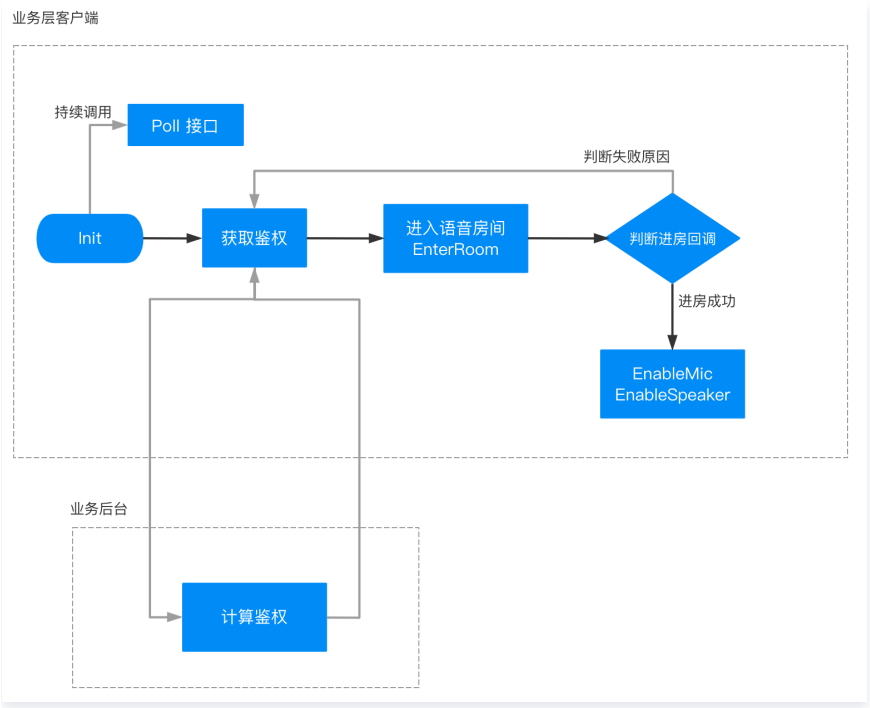
> **注意**
> 终端用户撤销同意处理其个人信息的授权时，您可通过调用 Uninit 接口停止使用SDK功能并停止采集与关闭功能相应的用户数据。

接口原型

```
public abstract int Uninit();
```

实时语音房间相关接口

初始化之后，SDK 调用进房后进去了房间，才可以进行实时语音通话。
使用问题可参考 实时语音相关问题 。



接口	接口含义
GenAuthBuffer	本地鉴权计算
EnterRoom	加入房间
ExitRoom	退出房间

IsRoomEntered	判断是否已经进入房间
SwitchRoom	快速切换房间
StartRoomSharing	跨房连麦

本地鉴权计算

生成 AuthBuffer，用于相关功能的加密和鉴权，如正式发布请使用后台部署密钥，后台部署请参考 鉴权密钥 。

接口原型

```
AuthBuffer public native byte[] genAuthBuffer(int sdkAppId, String roomId, String openId, String
key)
```

参数	类型	含义
appId	Int	来自腾讯云控制台的 AppId 号码。
roomId	String	房间号，最大支持127字符。
openId	String	用户标识。与 Init 时候的 OpenId相同。
key	String	来自腾讯云 控制台 的权限密钥。

示例代码

```
import com.tencent.av.sig.AuthBuffer;//头文件
byte[] authBuffer = AuthBuffer.getInstance().genAuthBuffer(Integer.parseInt(sdkAppId),
strRoomID,openId, key);
```

加入房间

用生成的鉴权信息进房，加入房间默认不打开麦克风及扬声器。

> **注意**
> - 加入房间事件回调结果 result 为 0 代表进房成功，进房接口 EnterRoom 返回值为 0 不代表进房成功。
> - 房间的音频类型由第一个进房的人确定，此后房间里有成员修改房间类型，将对此房间所有成员生效。例如第一个进入房间的人使用的房间音频类型是流畅音质，第二个进房的是即使进房时候调用接口的音频类型参数是高清音质，进入房间之后也会变成流畅音质。需要有成员调用 ChangeRoomType 才会修改房间的音频类型。

接口原型

```
public abstract int EnterRoom(String roomID, int roomType, byte[] authBuffer);
```

参数	类型	含义
roomId	String	房间号，最大支持127字符
roomType	Int	房间类型，建议填 ITMG_ROOM_TYPE_FLUENCY 。房间音频类型请参考 音质选择 。
authBuffer	byte[]	鉴权码

示例代码

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

加入房间的事件回调

加入房间完成后会发送信息 ITMG_MAIN_EVENT_TYPE_ENTER_ROOM，在 OnEvent 函数中进行判断回调后处理。如果回调为成功，即此时进房成功，开始进行计费。

> **计费问题参考：**
> - 购买指南。
> - 计费相关问题。
> - 使用实时语音后，如果客户端掉线了，是否还会继续计费？

函数原型

```
private ITMGContext.ITMGDelegate itmgDelegate = null;
itmgDelegate= new ITMGContext.ITMGDelegate() {
            @Override
             public void OnEvent(ITMGContext.ITMG_MAIN_EVENT_TYPE type, Intent data) {
                }
};
```

示例代码

回调处理相关参考代码，包括加入房间事件以及断网事件。

```
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!");
            }
        }
    if (ITMGContext.ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVENT_TYPE_ROOM_DISCONNECT == type)
        {
            //waiting timeout, please check your network
        }
    }
```

Data 详情

消息	Data	例子
ITMG_MAIN_EVENT_TYPE_ENTER_ROOM	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_ROOM_DISCONNECT	result; error_info	{"error_info":"waiting timeout, please check your network","result":0}

如果断网，将会有断网的回调提示 `ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVENT_TYPE_ROOM_DISCONNECT`，此时 SDK 会自动进行重连，回调是 `ITMG_MAIN_EVENT_TYPE_RECONNECT_START`，当重连成功时，会有 `ITMG_MAIN_EVENT_TYPE_RECONNECT_SUCCESS` 回调。

错误码

错误码值	原因及建议方案
7006	鉴权失败原因。 • AppID 不存在或者错误 • authbuff 鉴权错误 • 鉴权过期 • OpenId 不符合规范
7007	已经在其它房间
1001	已经在进房过程中，然后又重复了此操作。建议在进房回调返回之前不要再调用进房接口
1003	已经进房了在房间中，又调用一次进房接口
1101	确保已经初始化 SDK，确保 OpenId 是否符合规则，或者确保在同一线程调用接口，以及确保 Poll 接口正常调用

退出房间

通过调用此接口可以退出所在房间。这是一个异步接口，返回值为 AV_OK 的时候代表异步投递成功。如果应用中有退房后立即进房的场景，在接口调用流程上，开发者无需要等待 ExitRoom 的回调 RoomExitComplete 通知，只需直接调用接口。

接口原型

```
public abstract int ExitRoom();
```

示例代码

```
ITMGContext.GetInstance(this).ExitRoom();
```

退出房间事件回调

退出房间完成后会有回调，消息为 ITMG_MAIN_EVENT_TYPE_EXIT_ROOM。

示例代码

```
public void OnEvent(ITMGContext.ITMG_MAIN_EVENT_TYPE type, Intent data) {
    if (ITMGContext.ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVENT_TYPE_EXIT_ROOM == type)
        {
            //收到退房成功事件
        }
}
```

Data详情

消息	Data	例子
ITMG_MAIN_EVENT_TYPE_EXIT_ROOM	result; error_info	{"error_info":"","result":0}

判断是否已经进入房间

通过调用此接口可以判断是否已经进入房间，返回值为 bool 类型。请勿在进房过程中调用。

接口原型

```
public abstract boolean IsRoomEntered();
```

示例代码

```
ITMGContext.GetInstance(this).IsRoomEntered();
```

快速切换房间

调用此接口快速切换实时语音房间。此接口在进房后调用。切换房间后，不重置设备，即如果在此房间已经是打开麦克风状态，在切换房间后也会是打开麦克风状态。
快速切换房间的回调是 ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVENT_TYPE_SWITCH_ROOM，字段是 error_info 以及 result。

接口原型

```
public abstract int SwitchRoom(String targetRoomID, byte[] authBuffer);
```

类型说明

参数	类型	含义
targetRoomID	String	将要进入的房间号
authBuffer	Byte[]	用将要进入的房间号生成的新鉴权

回调示例代码

```
if(ITMGContext.ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVENT_TYPE_SWITCH_ROOM == type) {
int result = data.getIntExtra("result", 1);
String errorInfo = data.getStringExtra("error_info");
if (result == 0) {
            Toast.makeText(getActivity(), "switch room success.", Toast.LENGTH_SHORT).show();
            }
else {
            Toast.makeText(getActivity(), "switch room failed.. error info=" + errorInfo,
Toast.LENGTH_SHORT).show();
            }
}
```

跨房连麦

调用此接口进行跨房连麦，此接口在进房后调用。调用接口后，本端可以与目标房间的目标 OpenID 用户进行连麦交流。目标房间与本端房间类型相同才能成功。

场景示例

a 用户在 A 房间中，b 用户在 B 房间中，a 用户可以通过跨房接口与 b 进行通话，A 房间中的用户 c 说话，B 房间的 b 与 d 无法听到；A 房间中的用户 c 只能听到 A 房间的声音以及 B 房间中 b 的声音，B 房间其他人说话 c 无法听到。

接口原型

```
/// <summary> 开启房间共享，与另外的房间中的OpenID进行连麦</summary>
public abstract int StartRoomSharing(String targetRoomID, String targetOpenID, byte[]
authBuffer);
/// <summary> 结束已经开启的房间共享</summary>
public abstract int StopRoomSharing();
```

类型说明

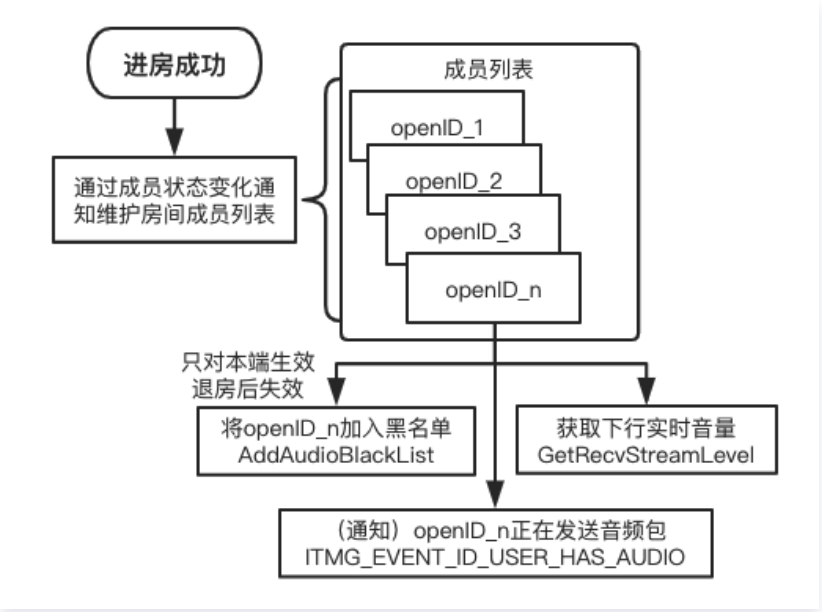
参数	类型	含义
targetRoomID	String	将要连麦的房间号
targetOpenID	String	将要连麦的目标 OpenID
authBuffer	byte[]	保留标志位，只需填 NULL

示例代码

```
if (mSwtichRoomShareStart.isChecked())
    {
        String strRoomID = mEditRoomShareRoomID.getText().toString();
        String strOpenID = mEditRoomShareOpenID.getText().toString();
        int nRet = ITMGContext.GetInstance(getActivity()).GetRoom().StartRoomSharing(strRoomID,
strOpenID, null);
        if (nRet != 0)
            {
                Toast.makeText(getActivity(), String.format("StartRoomSharing failed nRet=" +
nRet), Toast.LENGTH_SHORT).show();
            }else
            {
                int nRet = ITMGContext.GetInstance(getActivity()).GetRoom().StopRoomSharing();
                    if (nRet != 0)
                        {
                            Toast.makeText(getActivity(), String.format("StopRoomSharing failed
nRet=" + nRet), Toast.LENGTH_SHORT).show();
                        }
                }
}
```

房间内状态维护

此部分接口用于业务层显示说话成员、进退房成员，以及将房间内某成员禁言等功能。



接口/通知	含义
ITMG_MAIN_EVNET_TYPE_USER_UPDATE	成员状态变化通知
AddAudioBlackList	房间中禁言某成员
RemoveAudioBlackList	移除禁言

成员进房、开关麦克风、说话状态通知事件

此接口适用于获取房间中说话的人并在 UI 中展示，以及有人进入、退出语音房间的一个通知。
该事件在状态变化才通知，状态不变化的情况下不通知。如需实时获取成员状态，请在上层收到通知时缓存，事件消息为 ITMG_MAIN_EVNET_TYPE_USER_UPDATE，其中 data 包含两个信息，event_id 及 user_list，在 OnEvent 函数中对事件消息进行判断。
音频事件的通知有一个阈值，超过这个阈值才会发送通知。超过两秒没有收到音频包才通知“有成员停止发送音频包”消息。

event_id	含义	应用侧维护内容
ITMG_EVENT_ID_USER_ENTER	有成员进入房间，返回此时进房的 openid	应用侧维护成员列表
ITMG_EVENT_ID_USER_EXIT	有成员退出房间，返回此时退房的 openid	应用侧维护成员列表
ITMG_EVENT_ID_USER_HAS_AUDIO	有成员发送音频包，返回此时房间内说话的 openid，通过此事件可以判断用户是否说话，并展示声纹效果	应用侧维护通话成员列表
ITMG_EVENT_ID_USER_NO_AUDIO	有成员停止发送音频包，返回此时房间内停止说话的 openid	应用侧维护通话成员列表
ITMG_EVENT_ID_USER_MIC_OPENED	有成员打开麦克风设备（2.9.10及以后版本支持）	应用侧维护通话成员列表
ITME_EVENT_ID_USER_MIC_CLOSED	有成员关闭麦克风设备（2.9.10及以后版本支持）	应用侧维护通话成员列表

示例代码

```
public void OnEvent(ITMGContext.ITMG_MAIN_EVENT_TYPE type, Intent data) {
```

```
    if (ITMGContext.ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVNET_TYPE_USER_UPDATE == type)
        {
        //更新成员状态
        int nEventID = data.getIntExtra("event_id", 0);
        String[] openIdList =data.getStringArrayExtra("user_list");
         switch (nEventID)
             {
                    case ITMG_EVENT_ID_USER_ENTER:
                  //有成员进入房间
                  break;
                    case ITMG_EVENT_ID_USER_EXIT:
                  //有成员退出房间
                        break;
                    case ITMG_EVENT_ID_USER_HAS_AUDIO:
                        //有成员发送音频包
                        break;
                    case ITMG_EVENT_ID_USER_NO_AUDIO:
                        //有成员停止发送音频包
                        break;
                    case ITMG_EVENT_ID_USER_MIC_OPENED:
                        //有成员开启麦克风设备
                        break;
                    case ITME_EVENT_ID_USER_MIC_CLOSED:
                        //有成员关闭麦克风设备
                        break;
                    default:
                        break;
             }
        }
}
```

Data 详情

消息	Data	例子
ITMG_MAIN_EVNET_TYPE_USER_UPDATE	event_id; user_list	{"event_id":0,"user_list":""}

房间中禁言某成员

将某个 ID 加入音频数据黑名单，即不接受某人的语音， 只对本端生效，不会影响其他端。返回值为 0 表示调用成功。例如： A、B、C 都在同一个房间开麦说话：

- 如果 A 设置了 C 的黑名单， 则 A 只能听见 B 的声音。
- B 因为没有设置黑名单， 仍旧可以听见 A 和 C 的声音。
- C 同样因为没有设置黑名单， 可以听见 A 和 B 的声音。

此接口适用于在语音房间中将某用户禁言的场景。

接口原型

```
public abstract int AddAudioBlackList(String openId);
```

参数	类型	含义
openId	String	需添加黑名单的用户 openid

示例代码

```
ITMGContext.GetInstance(this).GetAudioCtrl().AddAudioBlackList(openId);
```

移除禁言

将某个 Id 移除音频数据黑名单。返回值为0表示调用成功。

接口原型

```
public abstract int RemoveAudioBlackList(String openId);
```

参数	类型	含义
openId	String	需移除黑名单的用户 openid

示例代码

```
ITMGContext.GetInstance(this).GetAudioCtrl().RemoveAudioBlackList(openId);
```

实时语音采集相关接口

- 初始化 SDK 之后进房，在房间中，才可以调用实时音频语音相关接口。
- 当用户界面单击打开/关闭麦克风/扬声器按钮时，建议采用 EnableMic 以及 EnableSpeaker 接口进行调用。
- 当用户进入实时语音房间，打开或者关闭采集设备，会伴随整个设备（采集及播放）重启，如果此时 App 正在播放背景音乐，那么背景音乐的播放也会被中断。利用控制上下行的方式来实现开关麦克风效果，不会中断播放设备。**具体调用方式为：在进房的时候调用 EnableAudioCaptureDevice(true) && EnableAudioPlayDevice(true) 一次，单击开关麦克风时只调用 EnableAudioSend/Recv 来控制音频流是否发送/接收。**

接口	接口含义
EnableMic	开关麦克风
GetMicState	获取麦克风状态
EnableAudioCaptureDevice	开关采集设备
IsAudioCaptureDeviceEnabled	获取采集设备状态
EnableAudioSend	打开关闭音频上行
IsAudioSendEnabled	获取音频上行状态
GetMicLevel	获取实时麦克风音量
GetSendStreamLevel	获取音频上行实时音量
SetMicVolume	设置麦克风音量
GetMicVolume	获取麦克风音量

开启或关闭麦克风

此接口用来开启关闭麦克风。加入房间默认不打开麦克风及扬声器。**EnableMic = EnableAudioCaptureDevice + EnableAudioSend如果有使用伴奏的情况，请参考** 实时语音伴奏流程图 **进行调用。**

接口原型

```
public abstract int EnableMic(boolean isEnabled);
```

参数	类型	含义
isEnabled	Boolean	如果需要打开麦克风，则传入的参数为 true，如果关闭麦克风，则参数为 false

示例代码

```
//打开麦克风
ITMGContext.GetInstance(this).GetAudioCtrl().EnableMic(true);
```

麦克风状态获取

此接口用于获取麦克风状态，返回值0为关闭麦克风状态，返回值1为打开麦克风状态。

接口原型

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

示例代码

```
int micState = ITMGContext.GetInstance(this).GetAudioCtrl().GetMicState();
```

开启或关闭采集设备

此接口用来开启/关闭采集设备。加入房间默认不打开设备。

- 只能在进房后调用此接口，退房会自动关闭设备。
- 在移动端，打开采集设备通常会伴随权限申请，音量类型调整等操作。

接口原型

```
public abstract int EnableAudioCaptureDevice(boolean isEnabled);
```

参数	类型	含义
isEnabled	Boolean	如果需要打开采集设备，则传入的参数为 true，如果关闭采集设备，则参数为 false

示例代码

```
//打开采集设备
ITMGContext.GetInstance(this).GetAudioCtrl().EnableAudioCaptureDevice(true);
```

采集设备状态获取

此接口用于采集设备状态获取。

接口原型

```
public abstract boolean IsAudioCaptureDeviceEnabled();
```

示例代码

```
bool IsAudioCaptureDevice =
ITMGContext.GetInstance(this).GetAudioCtrl().IsAudioCaptureDeviceEnabled();
```

打开或关闭音频上行

此接口用于打开/关闭音频上行。如果采集设备已经打开，那么会发送采集到的音频数据。如果采集设备没有打开，那么仍旧无声。采集设备的打开关闭请参见接口 EnableAudioCaptureDevice。

接口原型

```
public abstract int EnableAudioSend(boolean isEnabled);
```

参数	类型	含义
isEnabled	Boolean	如果需要打开音频上行，则传入的参数为 true，如果关闭音频上行，则参数为 false

示例代码

```
ITMGContext.GetInstance(this).GetAudioCtrl().EnableAudioSend(true);
```

音频上行状态获取

此接口用于音频上行状态获取。

接口原型

```
public abstract boolean IsAudioSendEnabled();
```

示例代码

```
bool IsAudioSend = ITMGContext.GetInstance(this).GetAudioCtrl().IsAudioSendEnabled();
```

获取麦克风实时音量

此接口用于获取麦克风实时音量，返回值为 int 类型。建议20ms获取一次。值域为0 – 100，通过此接口可以获取到麦克风采集到的实时音量情况。

接口原型

```
public abstract int GetMicLevel();
```

示例代码

```
int micLevel = ITMGContext.GetInstance(this).GetAudioCtrl().GetMicLevel();
```

获取音频上行实时音量

此接口用于获取自己音频上行实时音量，返回值为 int 类型，取值范围为0 – 100。

接口原型

```
ITMGContext TMGAudioCtrl int GetSendStreamLevel()
```

示例代码

```
int Level = ITMGContext.GetInstance(this).GetAudioCtrl().GetSendStreamLevel();
```

设置麦克风软件音量

此接口用于设置麦克风的音量。参数 volume 用于设置麦克风的音量，相当于对采集的声音做衰减或增益。

接口原型

```
public abstract int SetMicVolume(int volume);
```

参数	类型	含义
volume	int	取值范围为 0-200，数值为0的时候表示静音，当数值为100的时候表示音量不增不减，默认数值为100。

示例代码

```
ITMGContext.GetInstance(this).GetAudioCtrl().SetMicVolume(volume);
```

获取麦克风软件音量

此接口用于获取麦克风的音量。返回值为一个int类型数值，返回值为101代表没调用过接口 SetMicVolume。

接口原型

```
public abstract int GetMicVolume();
```

示例代码

```
ITMGContext.GetInstance(this).GetAudioCtrl().GetMicVolume();
```

实时语音播放相关接口

接口	接口含义
EnableSpeaker	开关扬声器
GetSpeakerState	获取扬声器状态
EnableAudioPlayDevice	开关播放设备
IsAudioPlayDeviceEnabled	获取播放设备状态
EnableAudioRecv	打开关闭音频下行
IsAudioRecvEnabled	获取音频下行状态
GetSpeakerLevel	获取实时扬声器音量

版权所有：腾讯云计算（北京）有限责任公司

GetRecvStreamLevel	获取房间内其他成员下行实时音量
SetSpeakerVolume	设置扬声器音量
GetSpeakerVolume	获取扬声器音量
SetSpeakerVolumeByOpenID	设置房间内某成员的说话音量大小
TrackingVolume	开始监听房间内用户的音量变化
StopTrackingVolume	停止监听房间内用户的音量变化

开启或关闭扬声器

此接口用于开启关闭扬声器。EnableSpeaker = EnableAudioPlayDevice + EnableAudioRecv如果有使用伴奏的情况，请参考 实时语音伴奏流程图 进行调用。

接口原型

```
public abstract int EnableSpeaker(boolean isEnabled);
```

参数	类型	含义
isEnabled	Boolean	如果需要关闭扬声器，则传入的参数为 false，如果打开扬声器，则参数为 true

示例代码

```
//打开扬声器
ITMGContext.GetInstance(this).GetAudioCtrl().EnableSpeaker(true);
```

扬声器状态获取

此接口用于扬声器状态获取。返回值0为关闭扬声器状态，返回值1为打开扬声器状态。

接口原型

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

示例代码

```
int micState = ITMGContext.GetInstance(this).GetAudioCtrl().GetSpeakerState();
```

开启或关闭播放设备

此接口用于开启关闭播放设备。

接口原型

```
public abstract int EnableAudioPlayDevice(boolean isEnabled);
```

参数	类型	含义
isEnabled	Boolean	如果需要关闭播放设备，则传入的参数为 false，如果打开播放设备，则参数为 true

示例代码

```
//打开播放设备
ITMGContext.GetInstance(this).GetAudioCtrl().EnableAudioPlayDevice(true);
```

播放设备状态获取

此接口用于播放设备状态获取。

接口原型

```
public abstract boolean IsAudioPlayDeviceEnabled();
```

示例代码

```
bool IsAudioPlayDevice =
ITMGContext.GetInstance(this).GetAudioCtrl().IsAudioPlayDeviceEnabled();
```

打开或关闭音频下行

此接口用于打开/关闭音频下行。如果播放设备已经打开，那么会播放房间里其他人的音频数据。如果播放设备没有打开，那么仍旧无声。播放设备的打开关闭参见接口请参见 EnableAudioPlayDevice。

接口原型

```
public abstract int EnableAudioRecv(boolean isEnabled);
```

参数	类型	含义
isEnabled	Boolean	如果需要打开音频下行，则传入的参数为 true，如果关闭音频下行，则参数为 false

示例代码

```
ITMGContext.GetInstance(this).GetAudioCtrl().EnableAudioRecv(true);
```

音频下行状态获取

此接口用于音频下行状态获取。

接口原型

```
public abstract boolean IsAudioRecvEnabled();
```

示例代码

```
bool IsAudioRecv = ITMGContext.GetInstance(this).GetAudioCtrl().IsAudioRecvEnabled();
```

获取扬声器实时音量

此接口用于获取扬声器实时音量。返回值为 int 类型数值，表示扬声器实时音量。建议20ms获取一次。

接口原型

```
public abstract int GetSpeakerLevel();
```

示例代码

```
int SpeakLevel = ITMGContext.GetInstance(this).GetAudioCtrl().GetSpeakerLevel();
```

获取房间内其他成员下行实时音量

此接口用于获取房间内其他成员下行实时音量，返回值为 int 类型，取值范围为0 – 200。

接口原型

```
public abstract int GetRecvStreamLevel(String openId);
```

参数	类型	含义
openId	String	房间其他成员的 openId

示例代码

```
int Level = ITMGContext.GetInstance(this).GetAudioCtrl().GetRecvStreamLevel(openId);
```

动态设置房间内某成员音量

此接口用于设置房间内某成员的说话音量大小，此设置只在本端生效。

接口原型

```
public abstract int SetSpeakerVolumeByOpenID(String openId, int volume);
```

参数	类型	含义
openId	String	需要调节音量大小的OpenID
volume	Int	百分比，建议[0-200]，其中100为默认值

示例代码

执行语句

```
// 将123333的声音压低到现在声音的80%
String strOpenID = "1233333";
int nOpenVolume = Integer.valueOf(80);
int nRet =
ITMGContext.GetInstance(getActivity()).GetAudioCtrl().SetSpeakerVolumeByOpenID(strOpenID,
nOpenVolume);
if (nRet != 0)
{
  // Toast error occured
}
else
{
  // Toast set successfully
```

```
}
```

获取设置的声音百分比

调用此接口获取SetSpeakerVolumeByOpenID设置的能量值

接口原型

```
public abstract int GetSpeakerVolumeByOpenID(String openId);
```

参数	类型	含义
openId	String	需要调节音量大小的OpenID

返回值

接口返回 OpenID 设置的能量百分比， 默认返回100。

设置扬声器的音量

此接口用于设置扬声器的音量。

接口原型

```
public abstract int SetSpeakerVolume(int volume);
```

参数	类型	含义
volume	Int	设置音量，范围0 – 200，当数值为0时，表示静音，当数值为100时，表示音量不增不减，默认数值为100。

示例代码

```
int speVol = (int)(value *
100);ITMGContext.GetInstance(this).GetAudioCtrl().SetSpeakerVolume(volume);
```

获取扬声器的音量

此接口用于获取扬声器的音量。返回值为 int 类型数值，代表扬声器的音量，返回值为101代表没调用过接口 SetSpeakerVolume。
Level 是实时音量，Volume 是扬声器的音量，最终声音音量 = Level × Volume %。例如实时音量是数值是100，此时 Volume 的数值是60，那么最终发出来的声音数值也是60。

接口原型

```
public abstract int GetSpeakerVolume();
```

示例代码

```
ITMGContext.GetInstance(this).GetAudioCtrl().GetSpeakerVolume();
```

开始监听房间内用户的音量变化

开始监听房间类用户（包括自己）的音量变化，当音量有变化时会触发ITMG_MAIN_EVENT_TYPE_USER_VOLUMES时间回调

接口原型

```
public abstract int TrackingVolume(float fTrackingTimeS);
```

参数	类型	含义
fTrackingTimeS	Float	音量变化事件回调的间隔，单位为秒

消息	含义	data
ITMG_MAIN_EVENT_TYPE_USER_VOLUMES	房间内用户音量发生变化	[{"open_id":"abc", "volume":2}, {"open_id":"def", "volume":2}]

示例代码

```
ITMGContext.GetInstance(this).GetAudioCtrl().TrackingVolume(1.0);
```

停止监听房间内用户的音量变化

停止监听房间内用户（包括自己）的音量变化

接口原型

```
public abstract int StopTrackingVolume();
```

示例代码

```
ITMGContext.GetInstance(this).GetAudioCtrl().StopTrackingVolume();
```

高级 API

启动耳返

此接口用于启动耳返，需要 `EnableLoopBack + EnableSpeaker` 才可以听到自己声音。

接口原型

```
public abstract int EnableLoopBack(boolean enable);
```

参数	类型	含义
enable	Boolean	设置是否启动

示例代码

```
ITMGContext.GetInstance(this).GetAudioCtrl().EnableLoopBack(true);
```

获取用户房间音频类型

此接口用于获取用户房间音频类型，返回值为房间音频类型，返回值为0时代表获取用户房间音频类型发生错误，房间音频类型参考 EnterRoom 接口。

接口原型

```
public abstract int GetRoomType();
```

示例代码

```
ITMGContext.GetInstance(this).GetRoom().GetRoomType();
```

获取房间号

此接口用于获取实时语音房间号，只能在进房成功之后使用。返回值为 string 字符串。

接口原型

```
public abstract String GetRoomID();
```

修改用户房间音频类型

此接口用于修改用户房间音频类型，结果参见回调事件，事件类型为 ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE。房间的音频类型由第一个进房的人确定，此后房间里有成员修改房间类型，将对此房间所有成员生效。

接口原型

```
public abstract int ChangeRoomType(int nRoomType);
```

参数	类型	含义
nRoomType	Int	房间切换成的目标类型，房间音频类型参考 EnterRoom 接口

示例代码

```
ITMGContext.GetInstance(this).GetRoom().ChangeRoomType(nRoomType);
```

房间类型修改回调

房间类型设置完成后，回调的事件消息为 ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE，返回的参数为 result、error_info 及 new_room_type，new_room_type 代表的信息如下，在 OnEvent 函数中对事件消息进行判断。

事件子类型	代表参数	含义
ITMG_ROOM_CHANGE_EVENT_ENTERROOM	1	表示在进房的过程中，自带的音频类型与房间不符合，被修改为所进入房间的音频类型。
ITMG_ROOM_CHANGE_EVENT_START	2	表示已经在房间内，音频类型开始切换（例如调用 ChangeRoomType 接口后切换音频类型）。
ITMG_ROOM_CHANGE_EVENT_COMPLETE	3	表示已经在房间，音频类型切换完成。
ITMG_ROOM_CHANGE_EVENT_REQUEST	4	表示房间成员调用 ChangeRoomType 接口，请求切换房间音频类型。

示例代码

```
public void OnEvent(ITMGContext.ITMG_MAIN_EVENT_TYPE type, Intent data) {    if
(ITMGContext.ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE == type)        {
//对房间类型事件进行处理      }}
```

Data 详情

消息	Data	例子
ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE	result;error_info;new_room_type;subEventType	{"error_info":"","new_room_type":0,"subEventType":0,"result":0}

房间通话质量监控事件

质量监控事件，在进房后触发，2秒回调一次，事件消息为 ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_QUALITY，返回的参数为 weight、loss 及 delay，代表的信息如下：

参数	类型	含义
weight	Int	范围是1 – 50，数值为50是音质评分极好，数值为1是音质评分很差，几乎不能使用，数值为0代表初始值，无含义。一般数值在30以下就可以提醒用户网络较差，建议切换网络。
loss	Double	上行丢包率。
delay	Int	音频触达延迟时间（ms）。

获取版本号

获取 SDK 版本号，用于分析。

接口原型

```
public abstract String GetSDKVersion();
```

示例代码

```
ITMGContext.GetInstance(this).GetSDKVersion();
```

检查麦克风权限

返回麦克风权限状态。

函数原型

```
public abstract ITMG_RECORD_PERMISSION  CheckMicPermission();
```

参数含义

参数	数值	含义
ITMG_PERMISSION_GRANTED	0	麦克风已授权
ITMG_PERMISSION_Denied	1	麦克风被禁用
ITMG_PERMISSION_NotDetermined	2	尚未弹出权限框向用户申请权限

版权所有：腾讯云计算（北京）有限责任公司

ITMG_PERMISSION_ERROR	3	接口调用错误

示例代码

```
ITMGContext.GetInstance(this).CheckMicPermission();
```

检查麦克风设备状态

函数原型

```
public abstract ITMG_CHECK_MIC_STATUS CheckMic();
```

返回值处理

返回值	含义	处理
ITMG_CHECK_MIC_STATUS_AVAILABLE = 0	正常可用	无需处理
ITMG_CHECK_MIC_STATUS_NO_GRANTED = 2	未获得/拒绝授权权限	需要在打开麦克风之前获取下权限
ITMG_CHECK_MIC_STATUS_INVALID_MIC = 3	没有可用的设备	一般是 PC 设备上，没有可用的麦克风设备会报此错误，请提示插入耳机或麦克风
ITMG_CHECK_MIC_STATUS_NOT_INIT = 5	没有初始化	在Init之后调用 EnableMic 接口

设置打印日志等级

用于设置打印日志等级。建议保持默认等级。需要在 Init 之前调用。

接口原型

```
public abstract int SetLogLevel(int levelWrite, int levelPrint);
```

参数含义

参数	类型	含义
levelWrite	ITMG_LOG_LEVEL	设置写入日志的等级，TMG_LOG_LEVEL_NONE 表示不写入，默认为 TMG_LOG_LEVEL_INFO
levelPrint	ITMG_LOG_LEVEL	设置打印日志的等级，TMG_LOG_LEVEL_NONE 表示不打印，默认为 TMG_LOG_LEVEL_ERROR

ITMG_LOG_LEVEL 说明如下：

ITMG_LOG_LEVEL	含义
TMG_LOG_LEVEL_NONE	不打印日志
TMG_LOG_LEVEL_ERROR	打印错误日志（默认）
TMG_LOG_LEVEL_INFO	打印提示日志

TMG_LOG_LEVEL_DEBUG	打印开发调试日志
TMG_LOG_LEVEL_VERBOSE	打印高频日志

示例代码

```
ITMGContext.GetInstance(this).SetLogLevel(TMG_LOG_LEVEL_INFO,TMG_LOG_LEVEL_INFO);
```

设置打印日志路径

用于设置打印日志路径。默认路径为： /sdcard/Android/data/xxx.xxx.xxx/files。需要在 Init 之前调用。

接口原型

```
public abstract int SetLogPath(String logDir);
```

参数	类型	含义
logDir	String	路径

示例代码

```
ITMGContext.GetInstance(this).SetLogPath(path);
```

获取诊断信息

获取音视频通话的实时通话质量的相关信息。该接口主要用来查看实时通话质量、排查问题等，业务侧可以忽略。

接口原型

```
public abstract String GetQualityTips();
```

示例代码

```
ITMGContext.GetInstance(this).GetRoom().GetQualityTips();
```

回调消息

消息	含义	Data	例子
ITMG_MAIN_EVENT_TYPE_ENTER_ROOM	进入音频房间消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_EXIT_ROOM	退出音频房间消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_ROOM_DISCONNECT	房间因为网络等原因断开消息	result; error_info	{"error_info":"waiting timeout, please check your network","result":0}
ITMG_MAIN_EVNET_TYPE_USER_UPDATE	房间成员更新消息	user_list; event_id	{"event_id":1,"user_list":["0"]}

ITMG_MAIN_EVENT_TYPE_RECONNECT_START	房间重连开始消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_RECONNECT_SUCCESS	房间重连成功消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_SWITCH_ROOM	快速切换房间消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE	房间状态切换消息	result; error_info; sub_event_type; new_room_type	{"error_info":"","new_room_type":0,"result":0}
ITMG_MAIN_EVENT_TYPE_ROOM_SHARING_START	开始跨房连麦消息	result;	{"result":0}
ITMG_MAIN_EVENT_TYPE_ROOM_SHARING_STOP	跨房连麦停止消息	result;	{"result":0}
ITMG_MAIN_EVENT_TYPE_SPEAKER_DEFAULT_DEVICE_CHANGED	默认扬声器设备修改消息	result; error_info	{"deviceID":"{0.0.0.00000000}.{a4f1e8be-49fa-43e2-b8cf-dd00542b47ae}","deviceName":"扬声器 (Realtek High Definition Audio)","error_info":"","isNewDevice":true,"isUsedDevice":false,"result":0}
ITMG_MAIN_EVENT_TYPE_SPEAKER_NEW_DEVICE	新增扬声器设备消息	result; error_info	{"deviceID":"{0.0.0.00000000}.{a4f1e8be-49fa-43e2-b8cf-dd00542b47ae}","deviceName":"扬声器 (Realtek High Definition Audio)","error_info":"","isNewDevice":true,"isUsedDevice":false,"result":0}
ITMG_MAIN_EVENT_TYPE_SPEAKER_LOST_DEVICE	丢失扬声器设备消息	result; error_info	{"deviceID":"{0.0.0.00000000}.{a4f1e8be-49fa-43e2-b8cf-dd00542b47ae}","deviceName":"扬声器 (Realtek High Definition Audio)","error_info":"","isNewDevice":false,"isUsedDevice":false,"result":0}
ITMG_MAIN_EVENT_TYPE_MIC_NEW_DEVICE	新增麦克风设备消息	result; error_info	{"deviceID":"{0.0.1.00000000}.{5fdf1a5b-f42d-4ab2-890a-7e454093f229}","deviceName":"麦克风 (Realtek High Definition Audio)","error_info":"","isNewDevice":true,"isUsedDevice":true,"result":0}
ITMG_MAIN_EVENT_TYPE_MIC_LOST_DEVICE	丢失麦克风设备消息	result; error_info	{"deviceID":"{0.0.1.00000000}.{5fdf1a5b-f42d-4ab2-890a-7e454093f229}","deviceName":"麦克风 (Realtek High Definition

			Audio)","error_info":"","isNewDevice":false,"isUsedDevice":true,"result":0}
ITMG_MAIN_EVENT_TYPE_MIC_DEFAULT_DEVICE_CHANGED	默认麦克风设备修改消息	result; error_info	{"deviceID":"{0.0.1.00000000}.{5fdf1a5b-f42d-4ab2-890a-7e454093f229}","deviceName":"麦克风 (Realtek High Definition Audio)","error_info":"","isNewDevice":false,"isUsedDevice":true,"result":0}
ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_QUALITY	房间质量消息	weight; loss; delay	{"weight":5,"loss":0.1,"delay":1}
ITMG_MAIN_EVNET_TYPE_PTT_RECORD_COMPLETE	语音消息录制完成消息	result; file_path	{"file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_UPLOAD_COMPLETE	语音消息上传完成消息	result; file_path;file_id	{"file_id":"","file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_DOWNLOAD_COMPLETE	语音消息下载完成消息	result; file_path;file_id	{"file_id":"","file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_PLAY_COMPLETE	语音消息播放完成消息	result; file_path	{"file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_SPEECH2TEXT_COMPLETE	语音消息极速转文本完成消息	result; text;file_id	{"file_id":"","text":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_COMPLETE	语音消息流式转文本完成消息	result; file_path; text;file_id	{"file_id":"","file_path":","text":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_IS_RUNNING	语音消息正在流式转文本中	result; file_path; text;file_id	{"file_id":"","file_path":","text":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_TEXT2SPEECH_COMPLETE	文本转语音完成消息	result; text;file_id	{"file_id":"","text":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_TRANSLATE_TEXT_COMPLETE	文本翻译完成消息	result; text;file_id	{"file_id":"","text":"","result":0}

版权所有：腾讯云计算（北京）有限责任公司

语音消息及转文本

最近更新时间：2025-06-18 14:16:01

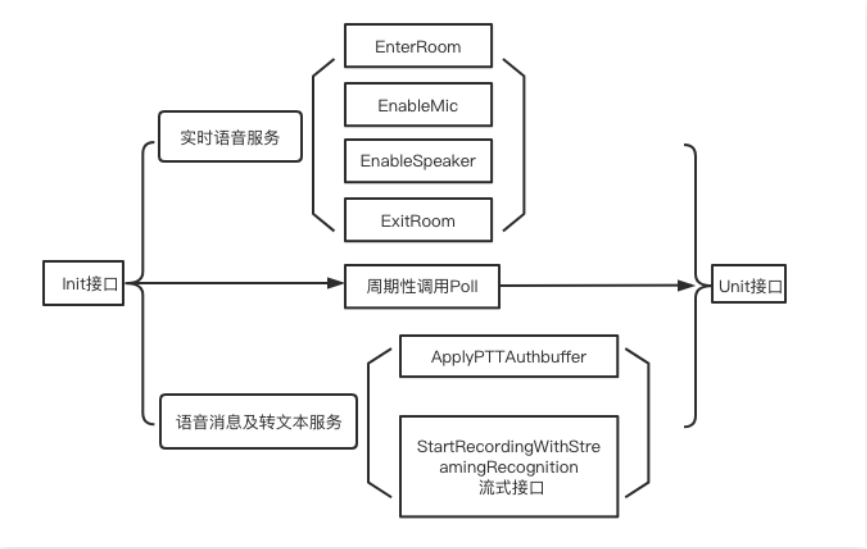
为方便 Android 开发者调试和接入腾讯云游戏多媒体引擎产品 API，这里向您介绍适用于 Android 开发的语音消息及转文本接入技术文档。

> **说明**
> 此文档对应 GME sdk version：2.9。

使用 GME 重要事项

GME 分为两个部分，提供实时语音服务、语音消息及转文本服务，使用这两个服务都依赖 Init 和 Poll 等核心接口。

> **注意**
> 语音转文本相关接口有默认频率限制，限额范围内计费方式请参见 计费文档；若需提升接口频率限额或了解超额计费方式，请联系商务或 提交工单咨询 。
> - 语音消息非流式转文本接口 SpeechToText()：默认单账号限制并发数为10路
> - 语音消息流式转文本接口 StartRecordingWithStreamingRecognition()：默认单账号限制并发数为50路
> - 实时语音流式转文本接口 StartRealTimeASR()：默认单账号限制并发数为50路



> **关于 Init 接口**
> 例如使用了实时语音服务，同时也需要使用语音消息服务，**只需要调用一次 Init 初始化接口**。
> Init 之后不会开始计费，语音消息及转文本服务**收发语音消息**才算作语音消息 DAU。

重要步骤

1. 初始化 GME
2. 周期性调用 Poll 触发回调
3. 鉴权初始化
4. 启动流式语音识别
5. 停止录制
6. 反初始化 GME

重点提示

- GME 使用前请对工程进行配置，否则 SDK 不生效。
- GME 的接口调用成功后返回值为 QAVError.OK，数值为 0。
- GME 的接口调用要在同一个线程下。
- GME 需要周期性的调用 Poll 接口触发事件回调。
- 错误码详情可参见 错误码 。

语音消息 Android 类

类	含义
ITMGContext	核心接口
ITMGPTT	语音消息及转文字相关接口

核心接口

未初始化前，SDK 处于未初始化阶段，需要通过接口 Init 初始化 SDK，才可以使用实时语音服务、语音消息及转文字服务。
在使用 GME 的任何接口之前，都需要先调用 Init 接口。
使用问题可参见 一般性问题 。

接口	接口含义
Init	初始化 GME
Poll	触发事件回调
Pause	系统暂停
Resume	系统恢复
Uninit	反初始化 GME

> ⚠ **注意**
> 如果切换账号，请调用 UnInit 反初始化 SDK。Init 接口调用不会产生计费。

获取单例

在使用语音功能时，需要首先获取 ITMGContext 对象。

示例代码

```
import com.tencent.TMG.ITMGContext;
ITMGContext.getInstance(this);
```

注册回调

接口类采用 Delegate 方法用于向应用程序发送回调通知。将回调函数注册给 SDK，用于接收回调的信息。

函数原型

在构造函数中重写这个回调函数，对回调的参数进行处理。

```
static public abstract class ITMGDelegate {
    public void OnEvent(ITMG_MAIN_EVENT_TYPE type, Intent data){}
}
```

参数	类型	含义
type	ITMGContext.ITMG_MAIN_EVENT_TYPE	回调的事件类型
data	Intent 消息类型	回调的相关信息，事件数据

示例代码

将回调函数注册给 SDK，要在进房之前设置。

```
private ITMGContext.ITMGDelegate itmgDelegate = null;
itmgDelegate = new ITMGContext.ITMGDelegate() {
    @Override
    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");
                }
        }
}
```

函数原型

```
public abstract int SetTMGDelegate(ITMGDelegate delegate);
```

参数	类型	含义
delegate	ITMGDelegate	SDK 回调函数

示例代码

```
ITMGContext.GetInstance(this).SetTMGDelegate(itmgDelegate);
```

初始化 SDK

- 此接口用于初始化 GME 服务，建议应用侧在应用初始化时候调用，调用此接口不会产生计费。
- **参数 sdkAppID 获取请参见 语音服务开通指引**。
- openID 用于唯一标识一个用户，目前只支持 INT64，规则由 App 开发者自行制定，App 内不重复即可。

> ⚠ 注意
> 调用 Init 接口的线程必须于其他接口在同一线程。建议都在主线程调用接口。

函数原型

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

参数	类型	含义
sdkAppId	String	来自 腾讯云控制台 的 GME 服务提供的 AppId。

openId	String	OpenId 只支持 Int64 类型（转为 string 传入）。

返回值	处理
QAVError.OK= 0	初始化 SDK 成功
AV_ERR_SDK_NOT_FULL_UPDATE=7015	检查 SDK 文件是否完整，建议删除后重新导入 SDK

出现返回值 AV_ERR_SDK_NOT_FULL_UPDATE 时，此返回值只有提示作用，并不会造成初始化失败。

- 如果在接入过程中提示此错误，请根据提示检查 SDK 文件是否完整、SDK 文件版本是否一致。
- 如果是在导出可执行文件之后出现此返回值，请忽略此错误，并尽量不在 UI 中提示。

示例代码

```
String sdkAppID = "14000*****";
String openID = "100";
int ret = ITMGContext.GetInstance(this).Init(sdkAppId, openId);
if(ret != 0){
                Log.e(TAG,"初始化 SDK 失败");
}
```

触发事件回调

通过在 update 里面周期的调用 Poll 可以触发事件回调。GME 需要周期性的调用 Poll 接口触发事件回调。如果没有调用 Poll 的话，会导致整个 SDK 服务运行异常。
可参考 Demo 中的 EnginePollHelper.java 文件。

> **注意**
> 务必周期性调用 Poll 接口且在主线程调用，以免接口回调异常。

函数原型

```
public abstract int Poll();
```

示例代码

```
private Handler mhandler = new Handler();
private Runnable mRunnable = new Runnable() {
    @Override
    public void run() {
        if (s_pollEnabled) {
            if (ITMGContext.GetInstance(null) != null)
                ITMGContext.GetInstance(null).Poll();
        }
        mhandler.postDelayed(mRunnable, 33);
    }
};
```

系统暂停

当系统发生 Pause 事件时，需要同时通知引擎进行 Pause。
如果需要在退后台的时候暂停语音，可以在退后台的监听代码中调用 Pause 接口，恢复前台的监听事件中调用 Resume 接口。

函数原型

```
public abstract int Pause();
```

系统恢复

当系统发生 Resume 事件时，需要同时通知引擎进行 Resume。Resume 接口只恢复实时语音。

函数原型

```
public abstract int Resume();
```

反初始化 SDK

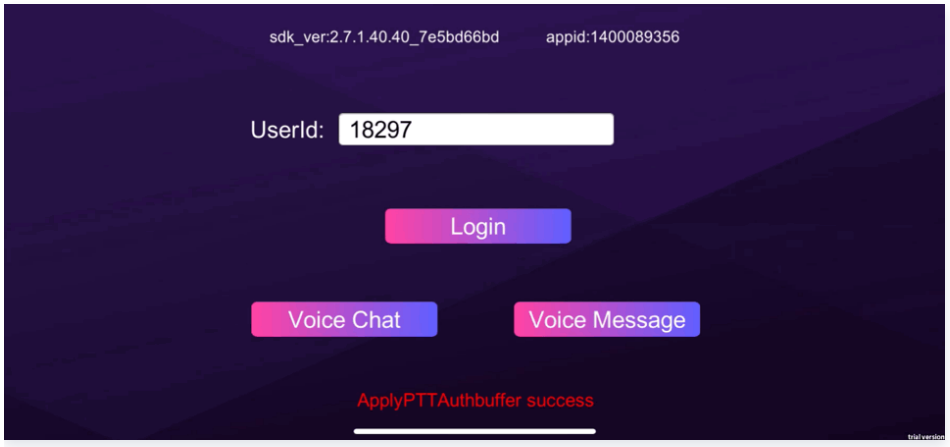
反初始化 SDK，进入未初始化状态。**切换账号需要反初始化**。

函数原型

```
public abstract int Uninit();
```

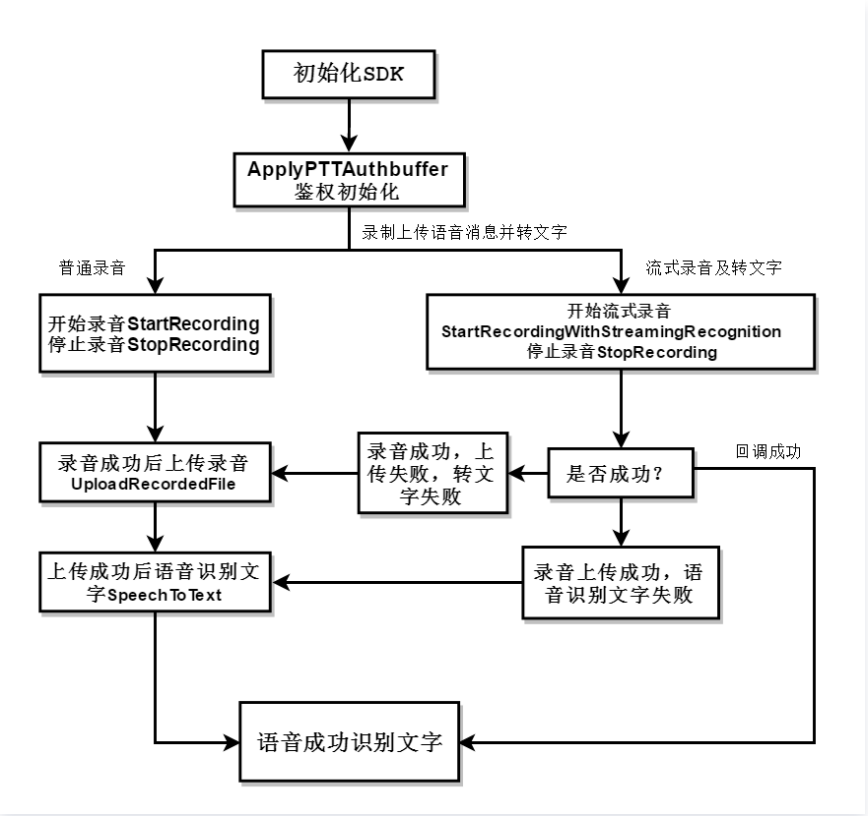
语音消息及转文字

语音消息，录制并发送一段语音消息，同时可以将语音消息转成文字，也可以同时将文字进行翻译。下图演示的是语音消息及转文本服务：



> **说明**
> - 建议使用流式语音转文字服务。
> - 使用语音消息服务不需要进入实时语音房间。

语音消息及语音转文字流程图



接入语音消息及转文本服务

语音消息及转文字相关接口

接口	接口含义
ApplyPTTAuthbuffer	鉴权初始化
SetMaxMessageLength	限制最大语音信息时长
StartRecording	启动录音
StartRecordingWithStreamingRecognition	启动流式录音
PauseRecording	暂停录音
ResumeRecording	恢复录音
StopRecording	停止录音
CancelRecording	取消录音
GetMicLevel	获取实时麦克风音量
SetMicVolume	设置录制音量
GetMicVolume	获取录制音量
GetSpeakerLevel	获取实时扬声器音量
SetSpeakerVolume	设置播放音量
GetSpeakerVolume	获取播放音量

UploadRecordedFile	上传语音文件
DownloadRecordedFile	下载语音文件
PlayRecordedFile	播放语音
StopPlayFile	停止播放语音
GetFileSize	语音文件的大小
GetVoiceFileDuration	语音文件的时长
SpeechToText	语音识别成文字

最大录制时长
语音消息最大录制时长默认为58秒，最短不能小于1秒。如果需要再加以限制，例如限制为最大录制时长为10秒，请在初始化之后调用 SetMaxMessageLength 接口进行设置。

初始化 SDK

未初始化前，SDK 处于未初始化阶段，需要通过接口 Init 初始化 SDK，才可以使用实时语音及语音消息服务。
使用问题可参考 离线语音相关问题 。

鉴权信息

生成 AuthBuffer，用于相关功能的加密和鉴权，如正式发布请使用后台部署密钥，后台部署请参考 鉴权密钥 。使用语音消息及转文字服务获取鉴权时，房间号参数必须填 null。

函数原型

```
AuthBuffer public native byte[] genAuthBuffer(int sdkAppId, String roomId, String openId, String key)
```

参数	类型	含义
appId	int	来自腾讯云控制台的 AppId 号码。
roomId	string	房间号参数必须填 null。
openId	string	用户标识。与 Init 时候的 openId 相同。
key	string	来自腾讯云 控制台 的权限密钥。

示例代码

```
import com.tencent.av.sig.AuthBuffer;//头文件
byte[] authBuffer = AuthBuffer.getInstance().genAuthBuffer(Integer.parseInt(sdkAppId), "0",openId, key);
```

鉴权初始化

在初始化 SDK 之后调用鉴权初始化，authBuffer 的获取参见上文实时语音鉴权信息接口 genAuthBuffer。

函数原型

```
public abstract int ApplyPTTAuthbuffer(byte[] authBuffer);
```

参数	类型	含义
authBuffer	String	鉴权

示例代码

```
byte[] authBuffer =  AuthBuffer.getInstance().genAuthBuffer(Integer.parseInt(sdkAppId), "0",
openId, key);
ITMGContext.GetInstance(this).GetPTT().ApplyPTTAuthbuffer(authBuffer);
```

流式语音识别

启动流式语音识别

此接口用于启动流式语音识别，同时在回调中会有实时的语音转文字返回，可以指定语言进行识别，也可以将语音中识别到的信息翻译成指定的语言返回。停止录音调用 StopRecording，停止之后才有回调。

函数原型

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

参数	类型	含义
filePath	String	存放的语音路径
speechLanguage	String	识别成指定文字的语言参数，参数请参见 语音转文字的语言参数参考列表
translateLanguage	String	翻译成指定文字的语言参数，参数请参见 语音转文字的语言参数参考列表（此参数暂不可用,请填写与 speechLanguage 相同的参数）

示例代码

```
String temple = getActivity().getExternalFilesDir(null).getAbsolutePath() + "/test_"+
(index++)+".ptt";
ITMGContext.GetInstance(getActivity()).GetPTT().StartRecordingWithStreamingRecognition(temple,"c
mn-Hans-CN","cmn-Hans-CN");
```

流式语音识别的回调

启动流式语音识别后，需要在回调函数 OnEvent 中监听回调消息，事件消息分为以下两个：

- `ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_COMPLETE` **是在停止录制并完成识别后才返回文字，相当于一段话说完才会返回识别的文字。**
- `ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_IS_RUNNING` **是在录音过程中就会实时返回识别到的文字，相当于边说话边返回识别到的文字。**

根据需求在 OnEvent 函数中对相应事件消息进行判断。传递的参数包含以下四个信息。

消息名称	含义
result	用于判断流式语音识别是否成功的返回码
text	语音转文字识别的文本
file_path	录音存放的本地地址
file_id	录音在后台的 url 地址，录音在服务器存放 90 天。fileid 固定字段为 http://gme-v2-

> **注意**
> 监听 `ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_IS_RUNNING` 消息时，file_id 为空。

错误码

错误码	含义	处理方式
32775	流式语音转文本失败，但是录音成功	调用 UploadRecordedFile 接口上传录音，再调用 SpeechToText 接口进行语音转文字操作
32777	流式语音转文本失败，但是录音成功，上传成功	返回的信息中有上传成功的后台 url 地址，调用 SpeechToText 接口进行语音转文字操作
32786	流式语音转文本失败	在流式录制状态当中，请等待流式录制接口执行结果返回

示例代码

```
public void OnEvent(ITMGContext.ITMG_MAIN_EVENT_TYPE type, Intent data) {
    if (ITMGContext.ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_COMPLETE
== type)
            {
                /流式离线语音回调。
                Handler mainHander = new Handler(Looper.getMainLooper());
                mainHander.post(new Runnable() {
                    @Override
                    public void run() {
                if (nErrCode ==0) {

                    String textString = templeData.getStringExtra("text");
                    EditText _editText = (EditText) root.findViewById(R.id.edit_audio_to_text);
                    _editText.setText(textString);

                    recordfilePath = templeData.getStringExtra("file_path");
                    mEditTextfiletoupload.setText(recordfilePath);

                    donwLoadUrlPath = templeData.getStringExtra("file_id");
                    mEditTextDownloadurl.setText(donwLoadUrlPath);

                    Log.e(TARGET, "STREAMINGRECOGNITION" + "nErrCode=" + nErrCode + ", content="
+ textString+"recordfilePath"+recordfilePath+"donwLoadUrlPath"+donwLoadUrlPath);
                }
                else
                {
```

```
                    Toast.makeText(getActivity(), String.format("流式语音转文本失败，错误代码:
%d",nErrCode), Toast.LENGTH_SHORT).show();
                    Log.e(TARGET, "流式语音转文本失败,错误代码"+nErrCode);
                }
                    }
                });
            }
}
```

语音消息录制

录制的流程为：录音->停止录音->录音回调返回->启动下一次录音。

限制最大语音信息时长

限制最大语音消息的长度，最大支持58秒。

函数原型

```
public abstract int SetMaxMessageLength(int msTime);
```

参数	类型	含义
msTime	int	语音时长，单位 ms，区间为 1000 < msTime ⩽ 58000

示例代码

```
ITMGContext.GetInstance(this).GetPTT().SetMaxMessageLength(msTime);
```

启动录音

此接口用于启动录音。需要将录音文件上传后才可以进行语音转文字等操作。**停止录音调用 StopRecording。**

函数原型

```
public abstract int StartRecording(String filePath);
```

参数	类型	含义
filePath	String	存放的语音路径

示例代码

```
ITMGContext.GetInstance(this).GetPTT().StartRecording(filePath);
```

停止录音

此接口用于停止录音。此接口为异步接口，停止录音后会有录音完成回调，成功之后录音文件才可用。

函数原型

```
public abstract int StopRecording();
```

示例代码

```
ITMGContext.GetInstance(this).GetPTT().StopRecording();
```

启动录音的回调

录音完成的回调，通过委托传递消息。
停止录音调用StopRecording。停止录音后才有启动录音的回调。
启动录音完成后的回调调用函数 OnEvent，事件消息为 ITMG_MAIN_EVNET_TYPE_PTT_RECORD_COMPLETE，在 OnEvent 函数中对事件消息进行判断。
传递的参数包含两个信息，一个是 result，另一个是 file_path。

错误码

错误码值	原因	建议方案
4097	参数为空	检查代码中接口参数是否正确
4098	初始化错误	检查设备是否被占用，或者权限是否正常，是否初始化正常
4099	正在录制中	确保在正确的时机使用 SDK 录制功能
4100	没有采集到音频数据	检查麦克风设备是否正常
4101	录音时，录制文件访问错误	确保文件存在，文件路径的合法性
4102	麦克风未授权错误	使用 SDK 需要麦克风权限，添加权限请参考对应引擎或平台的 SDK 工程配置文档
4103	录音时间太短错误	首先，限制录音时长的单位为毫秒，检查参数是否正确；其次，录音时长要1000毫秒以上才能成功录制
4104	没有启动录音操作	检查是否已经调用启动录音接口

示例代码

```
public void OnEvent(ITMGContext.ITMG_MAIN_EVENT_TYPE type, Intent data) {
    if (ITMGContext.ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVNET_TYPE_PTT_RECORD_COMPLETE == type)
            {
                    //启动录音的回调
                     if (nErrCode ==0)
                        {
                            recordfilePath = templeData.getStringExtra("file_path");
                            mEditTextfiletoupload.setText(recordfilePath);
                        }
            }
}
```

暂停录音

此接口用于暂停录音。如需恢复录音请调用接口 ResumeRecording。

函数原型

```
public abstract int PauseRecording();
```

示例代码

```
ITMGContext.GetInstance(this).GetPTT().PauseRecording();
```

恢复录音

此接口用于恢复录音。

函数原型

```
public abstract  int ResumeRecording();
```

示例代码

```
ITMGContext.GetInstance(this).GetPTT().ResumeRecording();
```

取消录音

调用此接口取消录音。取消之后没有回调。

函数原型

```
public abstract int CancelRecording();
```

示例代码

```
ITMGContext.GetInstance(this).GetPTT().CancelRecording();
```

获取语音消息麦克风实时音量

此接口用于获取麦克风实时音量，返回值为 int 类型，值域为0 - 200。
此接口不同于实时语音接口，此接口在 ITMGPTT.java 中。

函数原型

```
public abstract int GetMicLevel();
```

示例代码

```
ITMGContext.GetInstance(this).GetPTT().GetMicLevel();
```

设置语音消息录制音量

此接口用于设置离线语音录制音量，值域为0 - 200。
此接口不同于实时语音接口，此接口在 ITMGPTT.java 中。

函数原型

```
public abstract int SetMicVolume(int volume);
```

示例代码

```
ITMGContext.GetInstance(this).GetPTT().SetMicVolume(100);
```

获取语音消息录制音量

此接口用于获取离线语音录制音量。返回值为 int 类型，值域为0 - 200。
此接口不同于实时语音接口，此接口在 ITMGPTT.java 中。

函数原型

```
public abstract int GetMicVolume();
```

示例代码

```
ITMGContext.GetInstance(this).GetPTT().GetMicVolume();
```

获取语音消息扬声器实时音量

此接口用于获取扬声器实时音量。返回值为 int 类型，值域为0 - 200。
此接口不同于实时语音接口，此接口在 ITMGPTT.java 中。

函数原型

```
public abstract int GetSpeakerLevel();
```

示例代码

```
ITMGContext.GetInstance(this).GetPTT().GetSpeakerLevel();
```

设置语音消息播放音量

此接口用于设置离线语音播放音量，值域为0 - 200。
此接口不同于实时语音接口，此接口在 ITMGPTT.java 中。

函数原型

```
public abstract int SetSpeakerVolume(int volume);
```

示例代码

```
ITMGContext.GetInstance(this).GetPTT().SetSpeakerVolume(100);
```

获取语音消息播放音量

此接口用于获取离线语音播放音量。返回值为 int 类型，值域为0 - 200。
此接口不同于实时语音接口，此接口在 ITMGPTT.java 中。

函数原型

```
public abstract int GetSpeakerVolume();
```

示例代码

```
ITMGContext.GetInstance(this).GetPTT().GetSpeakerVolume();
```

语音消息播放

播放语音

此接口用于播放语音。

函数原型

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

参数	类型	含义
downloadFilePath	String	本地语音文件的路径
voicetype	int	变声类型，请参见 变声接入文档

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

示例代码

```
ITMGContext.GetInstance(this).GetPTT().PlayRecordedFile(downloadFilePath);
```

播放语音的回调

播放语音的回调，事件消息为 ITMG_MAIN_EVNET_TYPE_PTT_PLAY_COMPLETE， 在 OnEvent 函数中对事件消息进行判断。传递的参数包含两个信息，一个是 result，另一个是 file_path。

错误码

错误码值	原因	建议方案
20481	初始化错误	检查设备是否被占用，或者权限是否正常，是否初始化正常
20482	正在播放中，试图打断并播放下一个失败了（正常是可以打断的）	检查代码逻辑是否正确
20483	参数为空	检查代码中接口参数是否正确
20484	内部错误	初始化播放器错误，解码失败等问题产生此错误码，需要结合日志定位问题

示例代码

```
public void OnEvent(ITMGContext.ITMG_MAIN_EVENT_TYPE type, Intent data) {
    if(ITMGContext.ITMG_MAIN_EVNET_TYPE_PTT_PLAY_COMPLETE== type)
                {
            //播放语音的回调
```

```
        }
}
```

停止播放语音

此接口用于停止播放语音。停止播放语音也会有播放完成的回调。

函数原型

```
public abstract int StopPlayFile();
```

示例代码

```
ITMGContext.GetInstance(this).GetPTT().StopPlayFile();
```

获取语音文件的大小

通过此接口，获取语音文件的大小。

函数原型

```
public abstract int GetFileSize(String filePath);
```

参数	类型	含义
filePath	String	语音文件的路径，此路径为本地路径

示例代码

```
ITMGContext.GetInstance(this).GetPTT().GetFileSize(path);
```

获取语音文件的时长

此接口用于获取语音文件的时长，单位毫秒。

函数原型

```
public abstract int GetVoiceFileDuration(String filePath);
```

参数	类型	含义
filePath	String	语音文件的路径，此路径为本地路径

示例代码

```
ITMGContext.GetInstance(this).GetPTT().GetVoiceFileDuration(path);
```

语音消息上传及下载

上传语音文件

此接口用于上传语音文件。

函数原型

```
public abstract int UploadRecordedFile(String filePath);
```

参数	类型	含义
filePath	String	上传的语音路径，此路径为本地路径

示例代码

```
ITMGContext.GetInstance(this).GetPTT().UploadRecordedFile(filePath);
```

上传语音完成的回调

上传语音完成后，事件消息为 ITMG_MAIN_EVNET_TYPE_PTT_UPLOAD_COMPLETE，在 OnEvent 函数中对事件消息进行判断。传递的参数包含三个信息，result，file_path 和 file_id。

错误码

错误码值	原因	建议方案
8193	上传文件时，文件访问错误	确保文件存在，文件路径的合法性
8194	签名校验失败错误	检查鉴权密钥是否正确，检查是否有初始化离线语音
8195	网络错误	检查设备网络是否可以正常访问外网环境
8196	获取上传参数过程中网络失败	检查鉴权是否正确，检查设备网络是否可以正常访问外网环境
8197	获取上传参数过程中回包数据为空	检查鉴权是否正确，检查设备网络是否可以正常访问外网环境
8198	获取上传参数过程中回包解包失败	检查鉴权是否正确，检查设备网络是否可以正常访问外网环境
8200	没有设置 appinfo	检查 apply 接口是否有调用，或者入参是否为空

示例代码

```
public void OnEvent(ITMGContext.ITMG_MAIN_EVENT_TYPE type, Intent data) {
    if(ITMGContext.ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVNET_TYPE_PTT_UPLOAD_COMPLETE== type)
            {
               //上传语音完成的回调
            }
}
```

下载语音文件

此接口用于下载语音文件。

函数原型

```
public abstract int DownloadRecordedFile(String fileID, String filePath);
```

参数	类型	含义
fileID	String	文件的 url 路径

downloadFilePath	String	文件的本地保存路径

示例代码

```
ITMGContext.GetInstance(this).GetPTT().DownloadRecordedFile(url,path);
```

下载语音文件完成回调

下载语音完成后，事件消息为 `ITMG_MAIN_EVNET_TYPE_PTT_DOWNLOAD_COMPLETE`，在 OnEvent 函数中对事件消息进行判断，传递的参数包含三个信息，分别是 result、file_path 和 file_id。

错误码

错误码值	原因	建议方案
12289	下载文件时，文件访问错误	检查文件路径是否合法
12290	签名校验失败	检查鉴权密钥是否正确，检查是否有初始化离线语音
12291	网络存储系统异常	服务器获取语音文件失败，检查接口参数 fileid 是否正确，检查网络是否正常，检查 cos 文件存不存在
12292	服务器文件系统错误	检查设备网络是否可以正常访问外网环境，检查服务器上是否有此文件
12293	获取下载参数过程中，HTTP 网络失败	检查设备网络是否可以正常访问外网环境
12294	获取下载参数过程中，回包数据为空	检查设备网络是否可以正常访问外网环境
12295	获取下载参数过程中，回包解包失败	检查设备网络是否可以正常访问外网环境
12297	没有设置 appinfo	检查鉴权密钥是否正确，检查是否有初始化离线语音

示例代码

```
public void OnEvent(ITMGContext.ITMG_MAIN_EVENT_TYPE type, Intent data) {
    if(ITMGContext.ITMG_MAIN_EVNET_TYPE_PTT_DOWNLOAD_COMPLETE== type)
        {
            //下载成功
    }
}
```

语音转文字服务

将指定的语音文件识别成文字

此接口用于将指定的语音文件识别成文字。

函数原型

```
public abstract int SpeechToText(String fileID);
```

参数	类型	含义

fileID	String	语音文件 url

示例代码

```
ITMGContext.GetInstance(this).GetPTT().SpeechToText(fileID);
```

将指定的语音文件翻译成文字（指定语言）

此接口可以指定语言进行识别，也可以将语音中识别到的信息翻译成指定的语言返回。

函数原型

```
public abstract int SpeechToText(String fileID, String speechLanguage,String translatelanguage);
```

参数	类型	含义
fileID	String	语音文件 url，录音在服务器存放90天
speechLanguage	String	识别出指定文字的语言参数，参数参见 语音转文字的语言参数参考列表
translatelanguage	String	翻译成指定文字的语言参数，参数参见 语音转文字的语言参数参考列表（此参数暂时无效，填入参数应与 speechLanguage 一致）

示例代码

```
ITMGContext.GetInstance(this).GetPTT().SpeechToText(fileID,"cmn-Hans-CN","cmn-Hans-CN");
```

识别回调

将指定的语音文件识别成文字的回调，事件消息为 ITMG_MAIN_EVNET_TYPE_PTT_SPEECH2TEXT_COMPLETE， 在 OnEvent 函数中对事件消息进行判断。
传递的参数包含三个信息，result、file_id 和 text，其中 text 为识别的文本。

错误码

错误码值	原因	建议方案
32769	内部错误	分析日志，获取后台返回给客户端的真正错误码，并联系后台同事协助解决。
32770	网络失败	检查设备网络是否可以正常访问外网环境
32772	回包解包失败	分析日志，获取后台返回给客户端的真正错误码，并联系后台同事协助解决。
32774	没有设置 appinfo	检查鉴权密钥是否正确，检查是否有初始化离线语音
32776	authbuffer 校验失败	检查 authbuffer 是否正确
32784	语音转文本参数错误	检查代码中接口参数 fileid 是否为空
32785	语音转文本翻译返回错误	离线语音后台错误，请分析日志，获取后台返回给客户端的真正错误码，并联系后台同事协助解决

示例代码

```
public void OnEvent(ITMGContext.ITMG_MAIN_EVENT_TYPE type, Intent data) {
    if(ITMGContext.ITMG_MAIN_EVNET_TYPE_PTT_SPEECH2TEXT_COMPLETE == type)
            {
            //成功识别语音文件
     }
}
```

高级 API

获取版本号

获取 SDK 版本号，用于分析。

函数原型

```
public abstract String GetSDKVersion();
```

示例代码

```
ITMGContext.GetInstance(this).GetSDKVersion();
```

检查麦克风权限

返回麦克风权限状态。

函数原型

```
public abstract ITMG_RECORD_PERMISSION  CheckMicPermission();
```

参数含义

参数	数值	含义
ITMG_PERMISSION_GRANTED	0	麦克风已授权
ITMG_PERMISSION_Denied	1	麦克风被禁用
ITMG_PERMISSION_NotDetermined	2	尚未弹出权限框向用户申请权限
ITMG_PERMISSION_ERROR	3	接口调用错误

示例代码

```
ITMGContext.GetInstance(this).CheckMicPermission();
```

设置打印日志等级

用于设置打印日志等级。建议保持默认等级。需要在 Init 之前调用。

函数原型

```
public abstract int SetLogLevel(int levelWrite, int levelPrint);
```

参数含义

参数	类型	含义
levelWrite	ITMG_LOG_LEVEL	设置写入日志的等级，TMG_LOG_LEVEL_NONE 表示不写入，默认为 TMG_LOG_LEVEL_INFO
levelPrint	ITMG_LOG_LEVEL	设置打印日志的等级，TMG_LOG_LEVEL_NONE 表示不打印，默认为 TMG_LOG_LEVEL_ERROR

ITMG_LOG_LEVEL 说明

ITMG_LOG_LEVEL	含义
TMG_LOG_LEVEL_NONE	不打印日志
TMG_LOG_LEVEL_ERROR	打印错误日志（默认）
TMG_LOG_LEVEL_INFO	打印提示日志
TMG_LOG_LEVEL_DEBUG	打印开发调试日志
TMG_LOG_LEVEL_VERBOSE	打印高频日志

示例代码

```
ITMGContext.GetInstance(this).SetLogLevel(TMG_LOG_LEVEL_INFO,TMG_LOG_LEVEL_INFO);
```

设置打印日志路径

用于设置打印日志路径。默认路径为： /sdcard/Android/data/xxx.xxx.xxx/files。

函数原型

```
public abstract int SetLogPath(String logDir);
```

参数	类型	含义
logDir	String	路径

示例代码

```
ITMGContext.GetInstance(this).SetLogPath(path);
```

回调消息

消息列表

消息	消息代表的含义
ITMG_MAIN_EVNET_TYPE_PTT_RECORD_COMPLETE	PTT 录音完成
ITMG_MAIN_EVNET_TYPE_PTT_UPLOAD_COMPLETE	上传 PTT 完成
ITMG_MAIN_EVNET_TYPE_PTT_DOWNLOAD_COMPLETE	下载 PTT 完成

版权所有：腾讯云计算（北京）有限责任公司

ITMG_MAIN_EVNET_TYPE_PTT_PLAY_COMPLETE	播放 PTT 完成
ITMG_MAIN_EVNET_TYPE_PTT_SPEECH2TEXT_COMPLETE	语音转文字完成

Data 列表

消息	Data	例子
ITMG_MAIN_EVNET_TYPE_PTT_RECORD_COMPLETE	result; file_path	{"file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_UPLOAD_COMPLETE	result; file_path;file_id	{"file_id":"","file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_DOWNLOAD_COMPLETE	result; file_path;file_id	{"file_id":"","file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_PLAY_COMPLETE	result; file_path	{"file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_SPEECH2TEXT_COMPLETE	result; text;file_id	{"file_id":"","text":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_COMPLETE	result; file_path; text;file_id	{"file_id":"","file_path":","text":"","result":0}

工程导出

最近更新时间：2023-02-21 15:29:47

为方便 Android 开发者调试和接入腾讯云游戏多媒体引擎产品 API，本文档主要为您介绍 Android 项目工程导出注意事项。

导出工程

游戏多媒体引擎 SDK 默认提供 v7a、v8a、x86 以及 x86_64 架构，可根据项目需要进行删减。

> **警告**
> Android 系统设备运行时缺少对应架构的 so 文件会导致崩溃。

配置 App 权限

必要权限

请务必在工程 AndroidManifest.xml 文件中添加以下权限：

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

根据需求添加权限

请根据需求在工程 AndroidManifest.xml 文件中添加以下权限：

读写权限

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。
- Android 6 以上机器可不添加此权限。

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

蓝牙权限

需要根据以下规则判断如何添加蓝牙权限：

- 如果工程中的 targetSDKVersion 为 30 或者以下：

```
<uses-permission android:name="android.permission.BLUETOOTH"/>
```

- 如果工程中的 targetSDKVersion 为 31 或者以上且 GME 版本低于 v2.9.6：

```
<uses-permission android:name="android.permission.BLUETOOTH" android:maxSdkVersion="30"
/>
<uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />
```

App 混淆相关

如果需要对代码进行混淆，需进行以下配置：

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

注意，在2.9.0版本后，需要用以下配置进行混淆：

```
-dontwarn com.gme.**
-keep class com.gme.** { *;}
-keepclassmembers class com.gme.**{*;}
```

Android 高版本配置

- 根据谷歌 Developers 平台中对 Android 9 的 应用变更行为描述，Android 9 限制后台应用访问用户输入和传感器数据的能力，即后台运行的应用不能访问麦克风或摄像头。
- 如果 Android 9 设备用户有锁屏后采集音频或视频的需求，可以在锁屏或退至后台前起一个 Service，并在退出锁屏或返回前台前终止 Service。

Android 工程导出问题

导出可执行文件过程中、或者导出可执行文件后出现的工程问题：

- 导出到 Android 手机使用的时候，点击 App 会提示不支持该设备，如何解决?
- 集成 GME SDK 并导出 Apk 后，启动程序发生黑屏现象，如何解决?

macOS SDK 工程配置

最近更新时间：2023-03-20 15:36:31

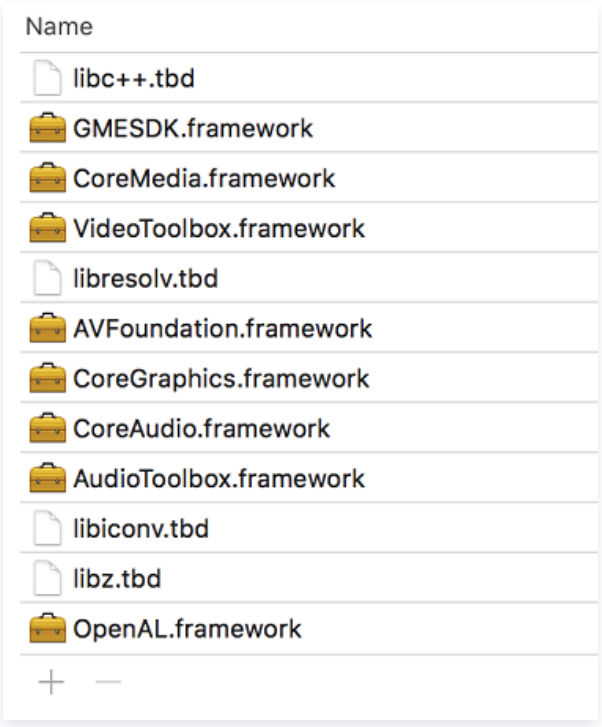
为方便 macOS 开发者调试和接入腾讯云游戏多媒体引擎产品 API，本文档主要为您介绍适用于 macOS 开发的工程配置。

SDK 准备

1. 请下载相关 Demo 及 SDK，详情请参见 下载指引 。
2. 解压获取到的 SDK 资源。
3. 文件夹内容 GMESDK.framework 为游戏多媒体引擎相关资源。

配置指引

根据情况在 Xcode 的 Link Binary With Libraries 里，加上下面的依赖库，并设置 Framework Search Paths 指向 SDK 所在目录，如图所示：



实时语音

最近更新时间：2023-10-13 17:09:53

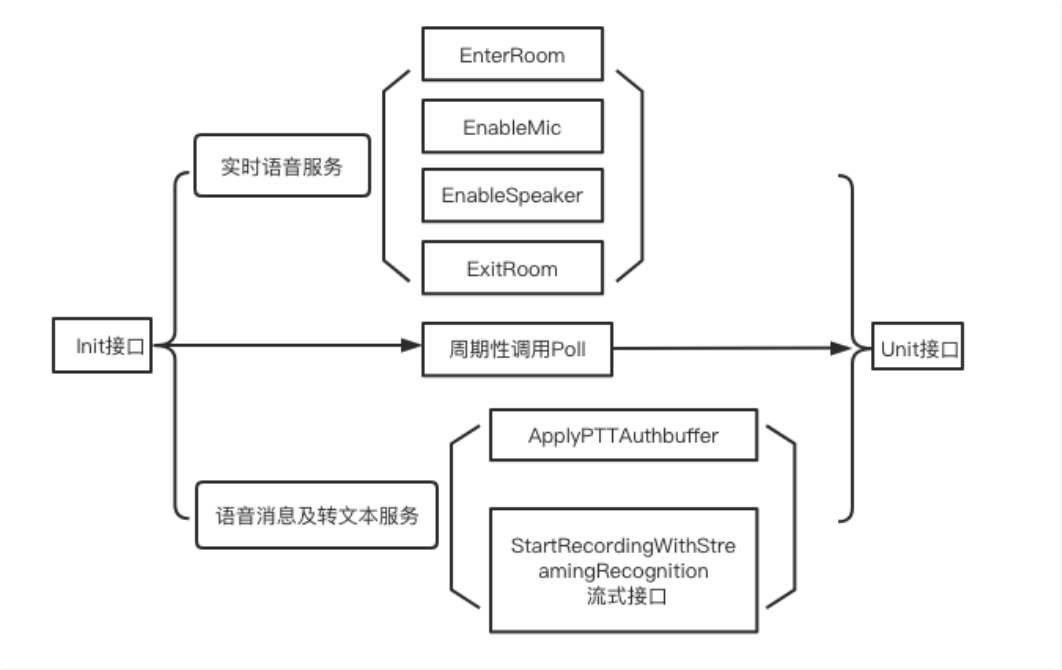
为方便 macOS 开发者调试和接入腾讯云游戏多媒体引擎产品 API，这里向您介绍适用于 macOS 开发的接入技术文档。

> **说明**
> 此文档对应 GME sdk version：2.9。

使用 GME 重要事项

GME 分为两个部分，提供实时语音服务、语音消息及转文本服务，使用这两个服务都依赖 Init 和 Poll 等核心接口。

> **关于 Init 接口**
> 例如使用了实时语音服务，同时也需要使用语音消息服务，**只需要调用一次 Init 初始化接口**。
> Init 之后不会开始计费，调用 进入实时语音房间 接口成功进入实时语音房间后才会开始计费。



重要步骤

1. 初始化 GME
2. 周期性调用 Poll 触发回调
3. 进入实时语音房间
4. 打开麦克风
5. 打开扬声器
6. 退出语音房间
7. 反初始化 GME

重点提示

- GME 使用前请对工程进行配置，否则 SDK 不生效。
- GME 的接口调用成功后返回值为 QAVError.OK，数值为 0。

- GME 的接口调用要在同一个线程下。
- GME 需要周期性的调用 Poll 接口触发事件回调。
- 错误码详情请参见 错误码 。

接口类

```
@class ITMGRoom;//房间相关
@class ITMGAudioCtrl;//音频相关
@class ITMGAudioEffectCtrl;//音效、伴奏相关
```

核心接口

未初始化前，SDK 处于未初始化阶段，需要通过接口 Init 初始化 SDK，才可以使用实时语音服务、语音消息及转文字服务。
在使用 GME 的任何接口之前，都需要先调用 Init 接口。
使用问题请参见 一般性问题 。

接口	接口含义
InitEngine	初始化 GME
Poll	触发事件回调
Pause	系统暂停
Resume	系统恢复
Uninit	反初始化 GME

获取单例

在使用语音功能时，需要首先获取 ITMGContext 对象。

函数原型

```
ITMGContext ITMGDelegate <NSObject>
```

示例代码

```
ITMGContext* _context = [ITMGContext GetInstance];
_context.TMGDelegate =self;
```

消息传递

在 OnEvent 中处理接口的回调信息，消息类型参考 ITMG_MAIN_EVENT_TYPE，消息内容为一个字典，解析出接口回调的内容。

函数原型

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

示例代码

```
-(void)OnEvent:(ITMG_MAIN_EVENT_TYPE)eventType data:(NSDictionary *)data{
        NSLog(@"OnEvent:%lu,data:%@",(unsigned long)eventType,data);
        switch (eventType) {
```

```
            //对 eventType 进行判断
            }
    }
```

初始化 SDK

- 此接口用于初始化 GME 服务，建议应用侧在应用初始化时候调用，调用此接口不会产生计费。
- **参数 sdkAppID 获取请参考 语音服务开通指引**。
- openID 用于唯一标识一个用户，目前只支持 INT64，规则由 App 开发者自行制定，App 内不重复即可。

> ⚠ 注意
> - 初始化 SDK 后才可以进入实时语音房间。
> - 调用 Init 接口的线程必须于其他接口在同一线程。建议都在主线程调用接口。

函数原型

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

参数	类型	含义
sdkAppId	String	来自 腾讯云控制台 的 GME 服务提供的 AppId。
OpenId	String	OpenId 只支持 Int64 类型（转为 string 传入）。

返回值

返回值	处理
QAV_OK= 0	初始化 SDK 成功
QAV_ERR_SDK_NOT_FULL_UPDATE= 7015	检查 SDK 文件是否完整，建议删除后重新导入 SDK

出现返回值 AV_ERR_SDK_NOT_FULL_UPDATE 时，此返回值**只有提示作用**，并不会造成初始化失败。
- 如果在接入过程中提示此错误，请根据提示检查 SDK 文件是否完整、SDK 文件版本是否一致。
- 如果是在导出可执行文件之后出现此返回值，请忽略此错误，并尽量不在 UI 中提示。

示例代码

```
_openId = _userIdText.text;
_appId = _appIdText.text;
[[ITMGContext GetInstance] InitEngine:SDKAPPID openID:_openId];
```

触发事件回调

通过在 update 里面周期的调用 Poll 可以触发事件回调。GME 需要周期性的调用 Poll 接口触发事件回调。如果没有调用 Poll 的话，会导致整个 SDK 服务运行异常。
详情请参见 Demo 中的 EnginePollHelper.m 文件。

> 🔔 务必周期性调用 Poll 接口
> 务必周期性调用 Poll 接口且在主线程调用，以免接口回调异常。

函数原型

```
-(void)Poll;
```

示例代码

```
[[ITMGContext GetInstance] Poll];
```

系统暂停

当系统发生 Pause 事件时，需要同时通知引擎进行 Pause。
如果需要在退后台的时候暂停语音，可以在退后台的监听代码中调用 Pause 接口，恢复前台的监听事件中调用 Resume 接口。

函数原型

```
-(QAVResult)Pause;
```

系统恢复

当系统发生 Resume 事件时，需要同时通知引擎进行 Resume。Resume 接口只恢复实时语音。

函数原型

```
-(QAVResult)Resume;
```

反初始化 SDK

反初始化 SDK，进入未初始化状态。**切换账号需要反初始化**。

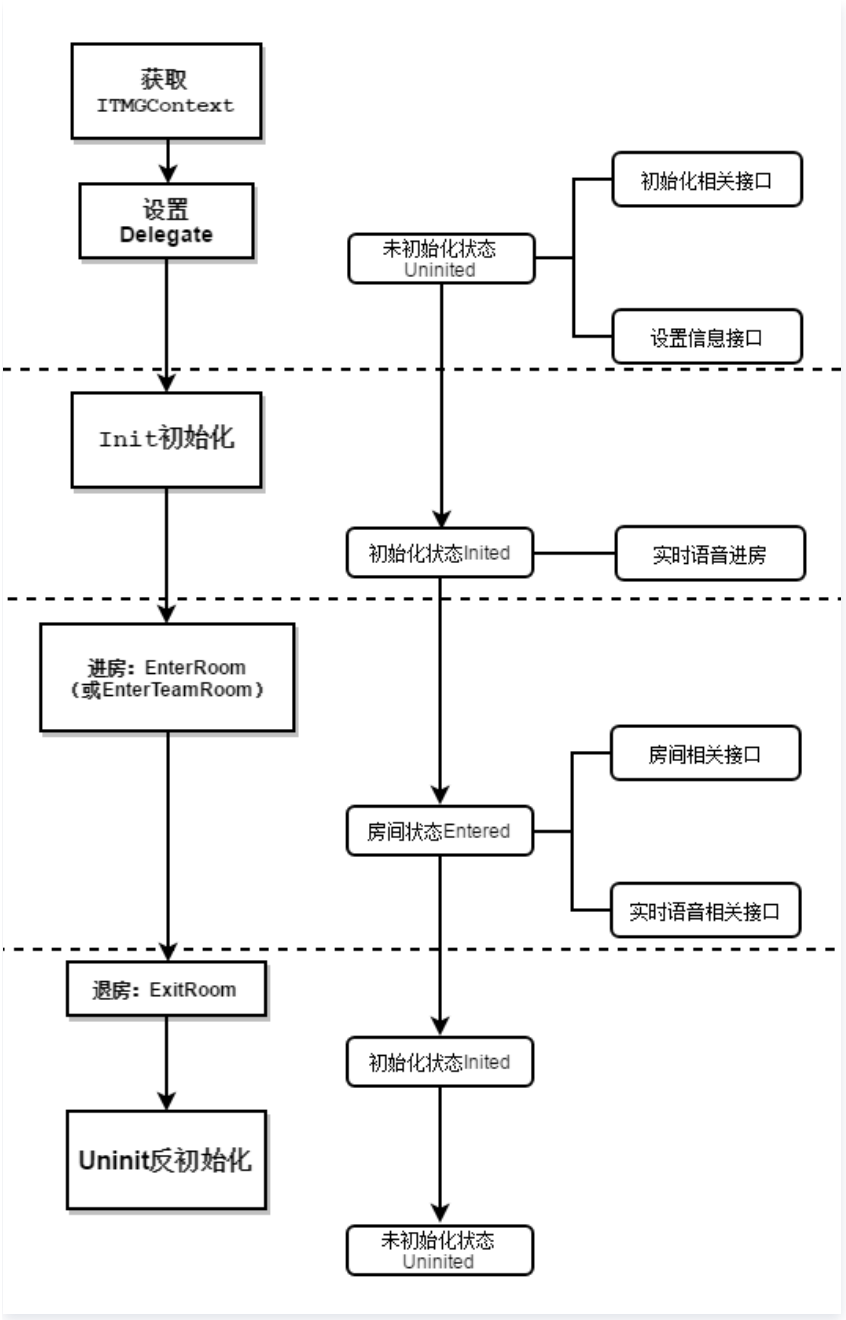
函数原型

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

示例代码

```
[[ITMGContext GetInstance] Uninit];
```

实时语音流程图



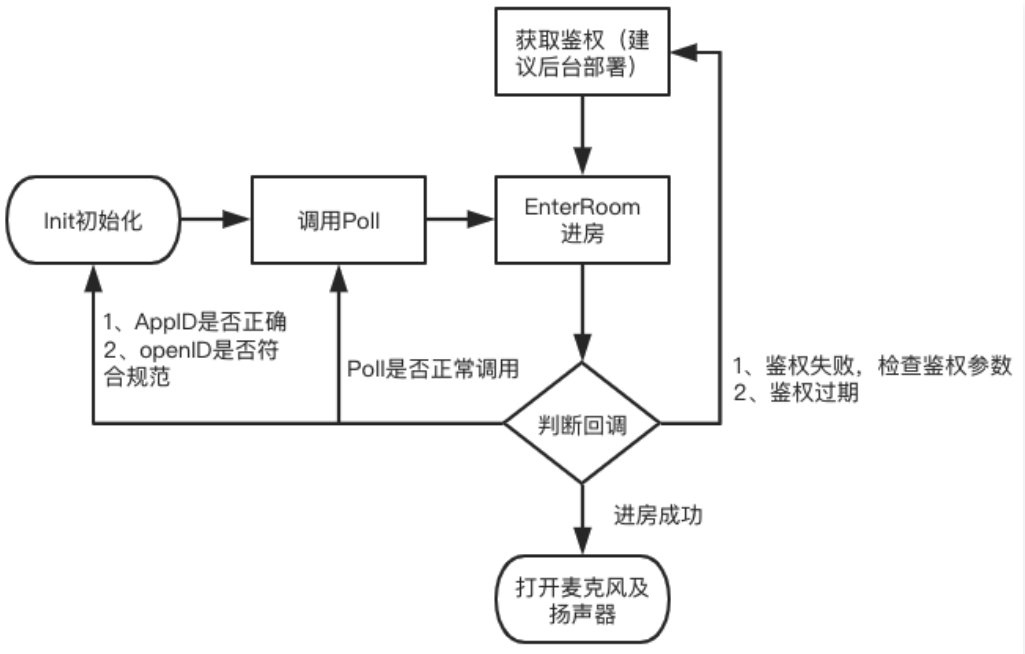
实时语音房间相关接口

初始化之后，SDK 调用进房后进去了房间，才可以进行实时语音通话。
使用问题可参考 实时语音相关问题 。

接口	接口含义
GenAuthBuffer	初始化鉴权
EnterRoom	加入房间
IsRoomEntered	是否已经进入房间
ExitRoom	退出房间
ChangeRoomType	修改用户房间音频类型

GetRoomType	获取用户房间音频类型

实时语音房间调用流程图



进房成功

进房回调结果 result 为0代表进房成功，进房接口 EnterRoom 返回值为0不代表进房成功。

鉴权信息

生成 AuthBuffer，用于相关功能的加密和鉴权，如正式发布请使用后台部署密钥，后台部署请参考 鉴权密钥 。

函数原型

```
@interface QAVAuthBuffer : NSObject
+ (NSData*) GenAuthBuffer:(unsigned int)appId roomId:(NSString*)roomId openID:(NSString*)openID
key:(NSString*)key;
+ @end
```

参数	类型	含义
appId	int	来自腾讯云控制台的 AppId 号码。
roomId	NSString	房间号，最大支持127字符。
openID	NSString	用户标识。与 Init 时候的 openID相同。
key	NSString	来自腾讯云 控制台 的权限密钥。

示例代码

```
#import "GMESDK/QAVAuthBuffer.h"
NSData* authBuffer = [QAVAuthBuffer GenAuthBuffer:SDKAPPID3RD.intValue roomId:_roomId
openID:_openId key:AUTHKEY];
```

加入房间

用生成的鉴权信息进房，会收到消息为 ITMG_MAIN_EVENT_TYPE_ENTER_ROOM 的回调。加入房间默认不打开麦克风及扬声器。返回值为 AV_OK 的时候代表成功。
房间的音频类型由第一个进房的人确定，此后房间里有成员修改房间类型，将对此房间所有成员生效。例如第一个进入房间的人使用的房间音频类型是流畅音质，第二个进房的是即使进房时候调用接口的音频类型参数是高清音质，进入房间之后也会变成流畅音质。需要有成员调用 ChangeRoomType 才会修改房间的音频类型。
房间音频类型请参考 音质选择 。

函数原型

```
-(int)EnterRoom:(NSString*) roomId roomType:(int)roomType authBuffer:(NSData*)authBuffer;
```

参数	类型	含义
roomId	NSString	房间号，最大支持127字符
roomType	int	房间音频类型
authBuffer	NSData	鉴权码

示例代码

```
[[ITMGContext GetInstance] EnterRoom:_roomId roomType:_roomType authBuffer:authBuffer];
```

加入房间事件的回调

加入房间完成后会发送信息 ITMG_MAIN_EVENT_TYPE_ENTER_ROOM，在 OnEvent 函数中进行判断回调后处理。如果回调为成功，即此时进房成功，开始进行计费。

> **计费问题参考：**
> - 购买指南。
> - 计费相关问题。
> - 使用实时语音后，如果客户端掉线了，是否还会继续计费？

示例代码

回调处理相关参考代码，包括加入房间事件以及断网事件。

```
-(void)OnEvent:(ITMG_MAIN_EVENT_TYPE)eventType data:(NSDictionary *)data{
    NSLog(@"OnEvent:%lu,data:%@",(unsigned long)eventType,data);
    switch (eventType) {
        case ITMG_MAIN_EVENT_TYPE_ENTER_ROOM:
        {
            int result = ((NSNumber*)[data objectForKey:@"result"]).intValue;
            NSString* error_info = [data objectForKey:@"error_info"];
                //收到进房成功事件
        }
            break;
    }
}
```

Data 详情

消息	Data	例子
ITMG_MAIN_EVENT_TYPE_ENTER_ROOM	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_ROOM_DISCONNECT	result; error_info	{"error_info":"waiting timeout, please check your network","result":0}

如果断网，将会有断网的回调提示 `ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVENT_TYPE_ROOM_DISCONNECT`，此时 SDK 会自动进行重连，回调是 `ITMG_MAIN_EVENT_TYPE_RECONNECT_START`，当重连成功时，会有 `ITMG_MAIN_EVENT_TYPE_RECONNECT_SUCCESS` 回调。

错误码

错误码值	原因及建议方案
7006	鉴权失败，原因如下： • AppID 不存在或者错误 • authbuff 鉴权错误 • 鉴权过期 • openID不符合规范
7007	已经在其它房间。
1001	已经在进房过程中，然后又重复了此操作。建议在进房回调返回之前不要再调用进房接口。
1003	已经进房了在房间中，又调用一次进房接口。
1101	确保已经初始化 SDK，确保 OpenId 是否符合规则，或者确保在同一线程调用接口，以及确保 Poll 接口正常调用。

退出房间

通过调用此接口可以退出所在房间。这是一个异步接口，返回值为 AV_OK 的时候代表异步投递成功。

> ⚠ **注意**
> 如果应用中有退房后立即进房的场景，在接口调用流程上，开发者无需要等待 ExitRoom 的回调 RoomExitComplete 通知，只需直接调用接口。

函数原型

```
-(int)ExitRoom
```

示例代码

```
[[ITMGContext GetInstance] ExitRoom];
```

退出房间回调

退出房间完成后会有回调，消息为 ITMG_MAIN_EVENT_TYPE_EXIT_ROOM。

示例代码

```
-(void)OnEvent:(ITMG_MAIN_EVENT_TYPE)eventType data:(NSDictionary *)data{
    NSLog(@"OnEvent:%lu,data:%@",(unsigned long)eventType,data);
    switch (eventType) {
        case ITMG_MAIN_EVENT_TYPE_EXIT_ROOM:
```

```
        {
            //收到退房成功事件
        }
            break;
    }
}
```

Data 详情

消息	Data	例子
ITMG_MAIN_EVENT_TYPE_EXIT_ROOM	result; error_info	{"error_info":"","result":0}

判断是否已经进入房间

通过调用此接口可以判断是否已经进入房间，返回值为 bool 类型。进房过程中调用无效果。

函数原型

```
-(BOOL)IsRoomEntered;
```

示例代码

```
[[ITMGContext GetInstance] IsRoomEntered];
```

快速切换房间

调用此接口快速切换实时语音房间。此接口在进房后调用。切换房间后，不重置设备，即如果在此房间已经是打开麦克风状态，在切换房间后也会是打开麦克风状态。
快速切换房间的回调是 ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVENT_TYPE_SWITCH_ROOM，字段是 error_info 以及 result。

接口原型

```
-(int) SwitchRoom:(NSString *)roomID  authBuffer:(NSData*)authBuffer;
```

类型说明

参数	类型	含义
targetRoomID	NSString *	将要进入的房间号
authBuffer	NSData*	用将要进入的房间号生成的新鉴权

回调示例代码

```
- (IBAction)swichRoom:(id)sender {
    NSData* authBuffer = [QAVAuthBuffer GenAuthBuffer:_appId.intValue roomID:_roomIdText.text
openID:_openId key:_key];
    [[[ITMGContext GetInstance]GetRoom]SwitchRoom:_roomIdText.text authBuffer:authBuffer];
}

-(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_SWITCH_ROOM:
        {
            int result = ((NSNumber*)[data objectForKey:@"result"]).intValue;
            NSString* log = nil;
            if (result == QAV_OK) {
                log = [NSString stringWithFormat:@"switch room success."];
            } else {
                log = [NSString stringWithFormat:@"switch room failed."];
            }
            [self showLog:log];
            break;
        }
    }
}
```

跨房连麦

调用此接口进行跨房连麦，此接口在进房后调用。调用接口后，本端可以与目标房间的目标 OpenID 用户进行连麦交流。目标房间与本端房间类型相同才能成功。

场景示例

a 用户在 A 房间中，b 用户在 B 房间中，a 用户可以通过跨房接口与 b 进行通话，A 房间中的用户 c 说话，B 房间的 b 与 d 无法听到；A 房间中的用户 c 只能听到 A 房间的声音以及 B 房间中 b 的声音，B 房间其他人说话 c 无法听到。

接口原型

```
-(int) StartRoomSharing:(NSString *)targetRoomID targetOpenID:(NSString *)targetOpenID
authBuffer:(NSData*)authBuffer;

-(int) StopRoomSharing;
```

类型说明

参数	类型	含义
targetRoomID	NSString *	将要连麦的房间号
targetOpenID	NSString *	将要连麦的目标 OpenID
authBuffer	NSData*	保留标志位，只需填 NULL

示例代码

```
- (IBAction)shareRoom:(id)sender {
    if(_shareRoomSwitch.isOn){
        [[[ITMGContext GetInstance]GetRoom]StartRoomSharing:_shareRoomID.text
targetOpenID:_shareOpenID.text authBuffer:NULL];
    }else{
        [[[ITMGContext GetInstance]GetRoom]StopRoomSharing];
    }
}
```

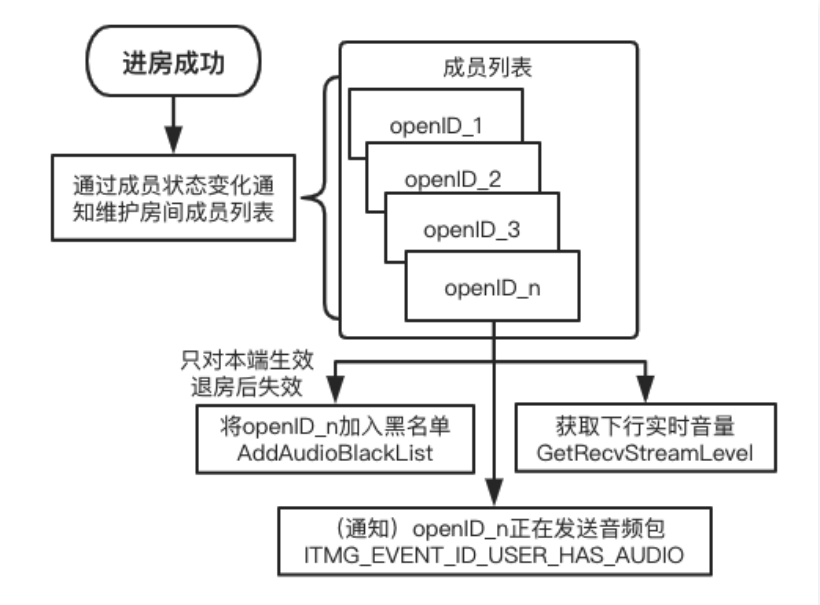
```
}
```

成员进房、说话状态通知

此接口适用于获取房间中说话的人并在 UI 中展示，以及有人进入、退出语音房间的一个通知。
该事件在状态变化时才通知，状态不变化的情况下不通知。如需实时获取成员状态，请在上层收到通知时缓存，事件消息为 ITMG_MAIN_EVNET_TYPE_USER_UPDATE，其中 data 包含两个信息，event_id 及 user_list，在 OnEvent 函数中对事件消息进行判断。
音频事件的通知有一个阈值，超过这个阈值才会发送通知。超过两秒没有收到音频包才通知“有成员停止发送音频包”消息。此事件只会返回成员说话事件，没有返回具体的音量。如需房间内成员具体音量可定期轮询接口 GetRecvStreamLevel 进行获取。

event_id	含义	应用侧维护内容
ITMG_EVENT_ID_USER_ENTER	有成员进入房间，返回此时进房的 openid	应用侧维护成员列表
ITMG_EVENT_ID_USER_EXIT	有成员退出房间，返回此时退房的 openid	应用侧维护成员列表
ITMG_EVENT_ID_USER_HAS_AUDIO	有成员发送音频包，返回此时房间内说话的 openid，通过此事件可以判断用户是否说话，并展示声纹效果	应用侧维护通话成员列表
ITMG_EVENT_ID_USER_NO_AUDIO	有成员停止发送音频包，返回此时房间内停止说话的 openid	应用侧维护通话成员列表
ITMG_EVENT_ID_USER_MIC_OPENED	有成员打开麦克风设备（2.9.10及以后版本支持）	应用侧维护通话成员列表
ITME_EVENT_ID_USER_MIC_CLOSED	有成员关闭麦克风设备（2.9.10及以后版本支持）	应用侧维护通话成员列表

维护房间成员流程图



示例代码

```
-(void)OnEvent:(ITMG_MAIN_EVENT_TYPE)eventType data:(NSDictionary *)data{
    ITMG_EVENT_ID_USER_UPDATE event_id=((NSNumber*)[data objectForKey:@"event_id"]).intValue;
    NSMutableArray* uses = [NSMutableArray arrayWithArray: [data objectForKey:@"user_list"]];
    NSLog(@"OnEvent:%lu,data:%@",(unsigned long)eventType,data);
```

```
    switch (eventType) {
        case ITMG_MAIN_EVNET_TYPE_USER_UPDATE:
        {
        //进行处理
        //开发者对参数进行解析，得到信息 event_id及 user_list
            switch (eventID)
             {
             case ITMG_EVENT_ID_USER_ENTER:
                  //有成员进入房间
                 break;
             case ITMG_EVENT_ID_USER_EXIT:
                  //有成员退出房间
                break;
            case ITMG_EVENT_ID_USER_HAS_AUDIO:
                //有成员发送音频包
                break;
            case ITMG_EVENT_ID_USER_NO_AUDIO:
                //有成员停止发送音频包
                break;
            case ITMG_EVENT_ID_USER_MIC_OPENED:
                //有成员开启了麦克风
                break;
            case ITME_EVENT_ID_USER_MIC_CLOSED:
                //有成员关闭了麦克风
                break;
             }
        break;
        }
    }
}
```

Data 详情

消息	Data	例子
ITMG_MAIN_EVNET_TYPE_USER_UPDATE	event_id; user_list	{"event_id":0,"user_list":""}

房间中禁言

将某个 ID 加入音频数据黑名单，即不接受某人的语音， 只对本端生效，不会影响其他端。返回值为 0 表示调用成功。例如： A，B，C 都在同一个房间开麦说话：

- 如果 A 设置了 C 的黑名单， 则 A 只能听见 B 的声音。
- B 因为没有设置黑名单， 仍旧可以听见 A 和 C 的声音。
- C 同样因为没有设置黑名单， 可以听见 A 和 B 的声音。

此接口适用于在语音房间中将某用户禁言的场景。

函数原型

```
ITMGContext GetAudioCtrl -(QAVResult)AddAudioBlackList:(NSString*)openID;
```

参数	类型	含义
openId	NSString	需添加黑名单的用户 openid

示例代码

```
[[[ITMGContext GetInstance]GetAudioCtrl ] AddAudioBlackList[id]];
```

移除禁言

将某个 ID 移除音频数据黑名单。返回值为 0 表示调用成功。

函数原型

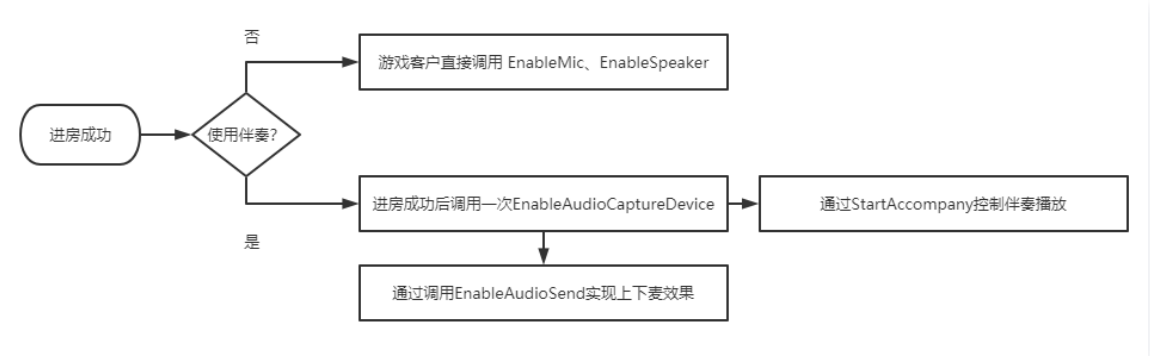
```
-(QAVResult)RemoveAudioBlackList:(NSString*)openID;
```

参数	类型	含义
openId	NSString	需移除黑名单的用户 openid

示例代码

```
[[[ITMGContext GetInstance]GetAudioCtrl ] RemoveAudioBlackList[openId]];
```

实时语音音频接口



实时语音音频接入须知

初始化 SDK 之后进房，在房间中，才可以调用实时音频语音相关接口。
当用户界面单击打开/关闭麦克风/扬声器按钮时，建议采用以下方式：

- **对于大部分的游戏类 App，推荐调用 EnableMic 及 EnableSpeaker 接口**，相当于同时调用 EnableAudioCaptureDevice/EnableAudioSend 和 EnableAudioPlayDevice/EnableAudioRecv 接口。
- 其他类型的移动端 App 例如社交类型 App，打开或者关闭采集设备，会伴随整个设备（采集及播放）重启，如果此时 App 正在播放背景音乐，那么背景音乐的播放也会被中断。利用控制上下行的方式来实现开关麦克风效果，不会中断播放设备。**具体调用方式为：在进房的时候调用 EnableAudioCaptureDevice(true) && EnableAudioPlayDevice(true) 一次，单击开关麦克风时只调用 EnableAudioSend/Recv 来控制音频流是否发送/接收。**
- 如果想单独释放采集或者播放设备，请参考接口 EnableAudioCaptureDevice 及 EnableAudioPlayDevice。
- 调用 pause 暂停音频引擎，调用 resume 恢复音频引擎。

实时语音音频相关接口

接口	接口含义
EnableMic	开关麦克风

GetMicState	获取麦克风状态
EnableAudioCaptureDevice	开关采集设备
IsAudioCaptureDeviceEnabled	获取采集设备状态
EnableAudioSend	打开关闭音频上行
IsAudioSendEnabled	获取音频上行状态
GetMicLevel	获取实时麦克风音量
GetSendStreamLevel	获取音频上行实时音量
SetMicVolume	设置麦克风音量
GetMicVolume	获取麦克风音量
EnableSpeaker	开关扬声器
GetSpeakerState	获取扬声器状态
EnableAudioPlayDevice	开关播放设备
IsAudioPlayDeviceEnabled	获取播放设备状态
EnableAudioRecv	打开关闭音频下行
IsAudioRecvEnabled	获取音频下行状态
GetSpeakerLevel	获取实时扬声器音量
GetRecvStreamLevel	获取房间内其他成员下行实时音量
SetSpeakerVolume	设置扬声器音量
GetSpeakerVolume	获取扬声器音量
EnableLoopBack	开关耳返

实时语音采集相关接口

开启或关闭麦克风

此接口用来开启关闭麦克风。加入房间默认不打开麦克风及扬声器。
如果有使用伴奏的情况，请参考 实时语音伴奏流程图 进行调用。
EnableMic = EnableAudioCaptureDevice + EnableAudioSend

函数原型

```
-(QAVResult)EnableMic:(BOOL)enable;
```

参数	类型	含义
isEnabled	boolean	• 如果需要打开麦克风，则传入的参数为 YES • 如果关闭麦克风，则传入的参数为 NO

示例代码

```
//打开麦克风
[[[ITMGContext GetInstance] GetAudioCtrl] EnableMic:YES];
```

麦克风状态获取

此接口用于获取麦克风状态，返回值0为关闭麦克风状态，返回值1为打开麦克风状态。

函数原型

```
-(int)GetMicState;
```

示例代码

```
[[[ITMGContext GetInstance] GetAudioCtrl] GetMicState];
```

开启或关闭采集设备

此接口用来开启/关闭采集设备。加入房间默认不打开设备。

- 只能在进房后调用此接口，退房会自动关闭设备。
- 在移动端，打开采集设备通常会伴随权限申请，音量类型调整等操作。

函数原型

```
-(QAVResult)EnableAudioCaptureDevice:(BOOL)enabled;
```

参数	类型	含义
enabled	BOOL	• 如果需要打开采集设备，则传入的参数为 YES • 如果关闭采集设备，则参数为 NO

示例代码

```
//打开采集设备
[[[ITMGContext GetInstance]GetAudioCtrl ]EnableAudioCaptureDevice:enabled];
```

采集设备状态获取

此接口用于采集设备状态获取。

函数原型

```
-(BOOL)IsAudioCaptureDeviceEnabled;
```

示例代码

```
BOOL IsAudioCaptureDevice = [[[ITMGContext GetInstance] GetAudioCtrl]
IsAudioCaptureDeviceEnabled];
```

打开或关闭音频上行

此接口用于打开/关闭音频上行。如果采集设备已经打开，那么会发送采集到的音频数据。如果采集设备没有打开，那么仍旧无声。采集设备的打开关闭参见接口 EnableAudioCaptureDevice。

函数原型

```
-(QAVResult)EnableAudioSend:(BOOL)enable;
```

参数	类型	含义
enable	BOOL	• 如果需要打开音频上行，则传入的参数为 YES • 如果关闭音频上行，则传入的参数为 NO

示例代码

```
[[[ITMGContext GetInstance]GetAudioCtrl ]EnableAudioSend:enabled];
```

音频上行状态获取

此接口用于音频上行状态获取。

函数原型

```
-(BOOL)IsAudioSendEnabled;
```

示例代码

```
BOOL IsAudioSend = [[[ITMGContext GetInstance] GetAudioCtrl] IsAudioSendEnabled];
```

获取麦克风实时音量

此接口用于获取麦克风实时音量，返回值为 int 类型。建议20ms获取一次。值域为0 – 100，通过此接口可以获取到麦克风采集到的实时音量情况。
此接口不适用于语音消息服务。

函数原型

```
-(int)GetMicLevel;
```

示例代码

```
[[[ITMGContext GetInstance] GetAudioCtrl] GetMicLevel];
```

获取音频上行实时音量

此接口用于获取自己音频上行实时音量，返回值为 int 类型，取值范围为0 – 100。
此接口不适用于语音消息服务。

函数原型

```
-(int)GetSendStreamLevel();
```

示例代码

```
[[[ITMGContext GetInstance] GetAudioCtrl] GetSendStreamLevel];
```

设置麦克风软件音量

此接口用于设置麦克风的音量。参数 volume 用于设置麦克风的音量，相当于对采集的声音做衰减或增益，当数值为0的时候表示静音，当数值为100的时候表示音量不增不减，默认数值为100。
此接口不适用于语音消息服务。

函数原型

```
-(QAVResult)SetMicVolume:(int) volume;
```

参数	类型	含义
volume	int	设置音量，范围0 – 200

示例代码

```
[[[ITMGContext GetInstance] GetAudioCtrl] SetMicVolume:100];
```

获取麦克风软件音量

此接口用于获取麦克风的音量。返回值为一个int类型数值，返回值为101代表没调用过接口 SetMicVolume。
此接口不适用于语音消息服务。

函数原型

```
-(int) GetMicVolume;
```

示例代码

```
[[[ITMGContext GetInstance] GetAudioCtrl] GetMicVolume];
```

实时语音播放相关接口

开启或关闭扬声器

此接口用于开启关闭扬声器。
如果有使用伴奏的情况，请参考 实时语音伴奏流程图 进行调用。
EnableSpeaker = EnableAudioPlayDevice + EnableAudioRecv

函数原型

```
-(void)EnableSpeaker:(BOOL)enable;
```

参数	类型	含义
isEnabled	boolean	• 如果需要关闭扬声器，则传入的参数为 NO • 如果打开扬声器，则传入的参数为 YES

示例代码

```
//打开扬声器
[[[ITMGContext GetInstance] GetAudioCtrl] EnableSpeaker:YES];
```

扬声器状态获取

此接口用于扬声器状态获取。返回值0为关闭扬声器状态，返回值1为打开扬声器状态。

函数原型

```
-(int)GetSpeakerState;
```

示例代码

```
[[[ITMGContext GetInstance] GetAudioCtrl] GetSpeakerState];
```

开启或关闭播放设备

此接口用于开启关闭播放设备。

函数原型

```
-(QAVResult)EnableAudioPlayDevice:(BOOL)enabled;
```

参数	类型	含义
enabled	BOOL	• 如果需要关闭播放设备，则传入的参数为 NO • 如果打开播放设备，则传入的参数为 YES

示例代码

```
//打开播放设备
[[[ITMGContext GetInstance]GetAudioCtrl ]EnableAudioPlayDevice:enabled];
```

播放设备状态获取

此接口用于播放设备状态获取。

函数原型

```
-(BOOL)IsAudioPlayDeviceEnabled;
```

示例代码

```
BOOL IsAudioPlayDevice =  [[[ITMGContext GetInstance] GetAudioCtrl] IsAudioPlayDeviceEnabled];
```

打开或关闭音频下行

此接口用于打开/关闭音频下行。如果播放设备已经打开，那么会播放房间里其他人的音频数据。如果播放设备没有打开，那么仍旧无声。播放设备的打开关闭参见接口 参见 EnableAudioPlayDevice。

函数原型

```
-(QAVResult)EnableAudioRecv:(BOOL)enabled;
```

参数	类型	含义
enabled	BOOL	• 如果需要打开音频下行，则传入的参数为 YES • 如果关闭音频下行，则传入的参数为 NO

示例代码

```
[[[ITMGContext GetInstance]GetAudioCtrl ]EnableAudioRecv:enabled];
```

音频下行状态获取

此接口用于音频下行状态获取。

函数原型

```
-(BOOL)IsAudioRecvEnabled;
```

示例代码

```
BOOL IsAudioRecv = [[[ITMGContext GetInstance] GetAudioCtrl] IsAudioRecvEnabled];
```

获取扬声器实时音量

此接口用于获取扬声器实时音量。返回值为 int 类型数值，表示扬声器实时音量。建议 20ms 获取一次。

函数原型

```
-(int)GetSpeakerLevel;
```

示例代码

```
[[[ITMGContext GetInstance] GetAudioCtrl] GetSpeakerLevel];
```

获取房间内其他成员下行实时音量

此接口用于获取房间内其他成员下行实时音量，返回值为 int 类型，取值范围为0 – 200。

函数原型

```
-(int)GetRecvStreamLevel:(NSString*) openID;
```

参数	类型	含义
openID	NSString	房间其他成员的 openId

示例代码

```
[[[ITMGContext GetInstance] GetAudioCtrl] GetRecvStreamLevel:(NSString*) openId
```

动态设置房间内某成员音量

此接口用于设置房间内某成员的说话音量大小，此设置只在本端生效。

函数原型

```
-(int) SetSpeakerVolumeByOpenID:(NSString *)openId volume:(int)volume;
```

#参数说明

参数	类型	含义
openId	String *	需要调节音量大小的OpenID
volume	int	百分比，建议[0-200]，其中100为默认值

获取设置的声音百分比

调用此接口获取SetSpeakerVolumeByOpenID设置的能量值

接口原型

```
-(int) GetSpeakerVolumeByOpenID:(NSString *)openId;
```

返回值

接口返回 OpenID 设置的能量百分比， 默认返回100。

设置扬声器的音量

此接口用于设置扬声器的音量。
参数 volume 用于设置扬声器的音量，当数值为0时，表示静音，当数值为100时，表示音量不增不减，默认数值为100。

函数原型

```
-(QAVResult)SetSpeakerVolume:(int)vol;
```

参数	类型	含义
vol	int	设置音量，范围0 – 200

示例代码

```
[[[ITMGContext GetInstance] GetAudioCtrl] SetSpeakerVolume:100];
```

获取扬声器的音量

此接口用于获取扬声器的音量。返回值为 int 类型数值，代表扬声器的音量，返回值为101代表没调用过接口 SetSpeakerVolume。
Level 是实时音量，Volume 是扬声器的音量，最终声音音量 = Level × Volume %。例如实时音量是数值是100，此时 Volume 的数值是60，那么最终发出来的声音数值也是60。

函数原型

```
-(int)GetSpeakerVolume;
```

示例代码

```
[[[ITMGContext GetInstance] GetAudioCtrl] GetSpeakerVolume];
```

高级 API

启动耳返

此接口用于启动耳返，需要 `EnableLoopBack + EnableSpeaker` 才可以听到自己声音。

函数原型

```
-(QAVResult)EnableLoopBack:(BOOL)enable;
```

参数	类型	含义
enable	boolean	设置是否启动

示例代码

```
[[[ITMGContext GetInstance] GetAudioCtrl] EnableLoopBack:YES];
```

修改用户房间音频类型

此接口用于修改用户房间音频类型，结果参见回调事件，事件类型为 ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE。房间的音频类型由第一个进房的人确定，此后房间里有成员修改房间类型，将对此房间所有成员生效。

函数原型

```
-(int)ChangeRoomType:(int)nRoomType;
```

参数	类型	含义
nRoomType	int	房间切换成的目标类型，房间音频类型请参见 EnterRoom 接口

示例代码

```
[[[ITMGContext GetInstance]GetRoom ]ChangeRoomType:_roomType];
```

获取用户房间音频类型

此接口用于获取用户房间音频类型，返回值为房间音频类型，返回值为0时代表获取用户房间音频类型发生错误，房间音频类型参考 EnterRoom 接口。

函数原型

```
-(int)GetRoomType;
```

示例代码

```
[[[ITMGContext GetInstance]GetRoom ]GetRoomType];
```

房间类型修改回调

房间类型设置完成后，回调的事件消息为 ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE，返回的参数为 result、error_info 及 new_room_type，new_room_type 代表的信息如下，在 OnEvent 函数中对事件消息进行判断。

事件子类型	代表参数	含义
ITMG_ROOM_CHANGE_EVENT_ENTERROOM	1	表示在进房的过程中，自带的音频类型与房间不符合，被修改为所进入房间的音频类型
ITMG_ROOM_CHANGE_EVENT_START	2	表示已经在房间内，音频类型开始切换（例如调用 ChangeRoomType 接口后切换音频类型 ）
ITMG_ROOM_CHANGE_EVENT_COMPLETE	3	表示已经在房间，音频类型切换完成
ITMG_ROOM_CHANGE_EVENT_REQUEST	4	表示房间成员调用 ChangeRoomType 接口，请求切换房间音频类型

示例代码

```
-(void)OnEvent:(ITMG_MAIN_EVENT_TYPE)eventType data:(NSDictionary *)data {
    NSLog(@"OnEvent:%lu,data:%@",(unsigned long)eventType,data);
    switch (eventType) {
        case ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE:
            NSLog(@"ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE:%@ ",data);
            int result = ((NSNumber*)[data objectForKey:@"result"]).intValue;
            int newRoomType = ((NSNumber*) [data objectForKey:@"new_room_type"]).intValue;
            int subEventType = ((NSNumber*) [data objectForKey:@"sub_event_type"]).intValue;
     }
}
```

Data 详情

消息	Data	例子
ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE	result;error_info;new_room_type;subEventType	{"error_info":"","new_room_type":0,"subEventType":0,"result":0}

房间通话质量监控事件

质量监控事件，在进房后触发，2秒回调一次，事件消息为 ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_QUALITY，返回的参数为 weight、loss 及 delay，代表的信息如下。
此接口适用于监听网络质量，如果用户网络差的话，业务层将通过 UI 提醒用户切换网络。

参数	类型	含义
weight	int	范围是1 – 50，数值为50是音质评分极好，数值为1是音质评分很差，几乎不能使用，数值为0代表初始值，无含义。一般数值在 30 以下就可以提醒用户网络较差，建议切换网络。
loss	double	上行丢包率。

delay	int	音频触达延迟时间（ms）。

获取版本号

获取 SDK 版本号，用于分析。

函数原型

```
-(NSString*)GetSDKVersion;
```

示例代码

```
[[ITMGContext GetInstance] GetSDKVersion];
```

检查麦克风权限

返回麦克风权限状态。

函数原型

```
-(ITMG_RECORD_PERMISSION)CheckMicPermission;
```

参数含义

参数	数值	含义
ITMG_PERMISSION_GRANTED	0	麦克风已授权
ITMG_PERMISSION_Denied	1	麦克风被禁用
ITMG_PERMISSION_NotDetermined	2	尚未弹出权限框向用户申请权限
ITMG_PERMISSION_ERROR	3	接口调用错误

示例代码

```
[[ITMGContext GetInstance] CheckMicPermission];
```

设置打印日志等级

用于设置打印日志等级。建议保持默认等级。需要在 Init 之前调用。

函数原型

```
-(void)SetLogLevel:(ITMG_LOG_LEVEL)levelWrite (ITMG_LOG_LEVEL)levelPrint;
```

参数含义

参数	类型	含义
levelWrite	ITMG_LOG_LEVEL	设置写入日志的等级，TMG_LOG_LEVEL_NONE 表示不写入，默认为 TMG_LOG_LEVEL_INFO

levelPrint	ITMG_LOG_LEVEL	设置打印日志的等级，TMG_LOG_LEVEL_NONE 表示不打印，默认为 TMG_LOG_LEVEL_ERROR

ITMG_LOG_LEVEL	含义
TMG_LOG_LEVEL_NONE	不打印日志
TMG_LOG_LEVEL_ERROR	打印错误日志（默认）
TMG_LOG_LEVEL_INFO	打印提示日志
TMG_LOG_LEVEL_DEBUG	打印开发调试日志
TMG_LOG_LEVEL_VERBOSE	打印高频日志

示例代码

```
[[ITMGContext GetInstance] SetLogLevel:TMG_LOG_LEVEL_INFO TMG_LOG_LEVEL_INFO];
```

设置打印日志路径

用于设置打印日志路径。默认路径为： /Users/username/Library/Containers/xxx.xxx.xxx/Data/Documents。

函数原型

```
-(void)SetLogPath:(NSString*)logDir;
```

参数	类型	含义
logDir	NSString	路径

示例代码

```
[[ITMGContext GetInstance] SetLogPath:Path];
```

获取诊断信息

获取音视频通话的实时通话质量的相关信息。该接口主要用来查看实时通话质量、排查问题等，业务侧可以忽略。

函数原型

```
-(NSString*)GetQualityTips;
```

示例代码

```
[[[ITMGContext GetInstance]GetRoom ] GetQualityTips];
```

回调消息

消息列表

消息	消息代表的含义

ITMG_MAIN_EVENT_TYPE_ENTER_ROOM	进入音频房间消息
ITMG_MAIN_EVENT_TYPE_EXIT_ROOM	退出音频房间消息
ITMG_MAIN_EVENT_TYPE_ROOM_DISCONNECT	房间因为网络等原因断开消息
ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE	房间类型变化事件
ITMG_MAIN_EVNET_TYPE_USER_UPDATE	房间成员更新消息
ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_QUALITY	房间质量信息

Data 列表

消息	Data	例子
ITMG_MAIN_EVENT_TYPE_ENTER_ROOM	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_EXIT_ROOM	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_ROOM_DISCONNECT	result; error_info	{"error_info":"waiting timeout, please check your network","result":0}
ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE	result; error_info; sub_event_type; new_room_type	{"error_info":"","new_room_type":0,"result":0}
ITMG_MAIN_EVENT_TYPE_SPEAKER_NEW_DEVICE	result; error_info	{"deviceID":"{0.0.0.00000000}.{a4f1e8be-49fa-43e2-b8cf-dd00542b47ae}","deviceName":"扬声器 (Realtek High Definition Audio)","error_info":"","isNewDevice":true,"isUsedDevice":false,"result":0}
ITMG_MAIN_EVENT_TYPE_SPEAKER_LOST_DEVICE	result; error_info	{"deviceID":"{0.0.0.00000000}.{a4f1e8be-49fa-43e2-b8cf-dd00542b47ae}","deviceName":"扬声器 (Realtek High Definition Audio)","error_info":"","isNewDevice":false,"isUsedDevice":false,"result":0}
ITMG_MAIN_EVENT_TYPE_MIC_NEW_DEVICE	result; error_info	{"deviceID":"{0.0.1.00000000}.{5fdf1a5b-f42d-4ab2-890a-7e454093f229}","deviceName":"麦克风 (Realtek High Definition Audio)","error_info":"","isNewDevice":true,"isUsedDevice":true,"result":0}
ITMG_MAIN_EVENT_TYPE_MIC_LOST_DEVICE	result; error_info	{"deviceID":"{0.0.1.00000000}.{5fdf1a5b-f42d-4ab2-890a-7e454093f229}","deviceName":"麦克风 (Realtek High Definition Audio)","error_info":"","isNewDevice":false,"isUsedDevice":true,"result":0}
ITMG_MAIN_EVNET_TYPE_USER_UPDATE	user_list; event_id	{"event_id":1,"user_list":["0"]}
ITMG_MAIN_EVENT_TYPE_NUMBER_OF_USERS_UPDATE	AllUser; AccUser; ProxyUser	{"AllUser":3,"AccUser":2,"ProxyUser":1}
ITMG_MAIN_EVENT_TYPE_NUMBER_OF_AUDIOSTREAMS_UPDATE	AudioStreams	{"AudioStreams":3}

版权所有：腾讯云计算（北京）有限责任公司

ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_QUALITY	weight; loss; delay	{"weight":5,"loss":0.1,"delay":1}

版权所有：腾讯云计算（北京）有限责任公司

语音消息及转文本

最近更新时间：2023-06-30 16:12:11

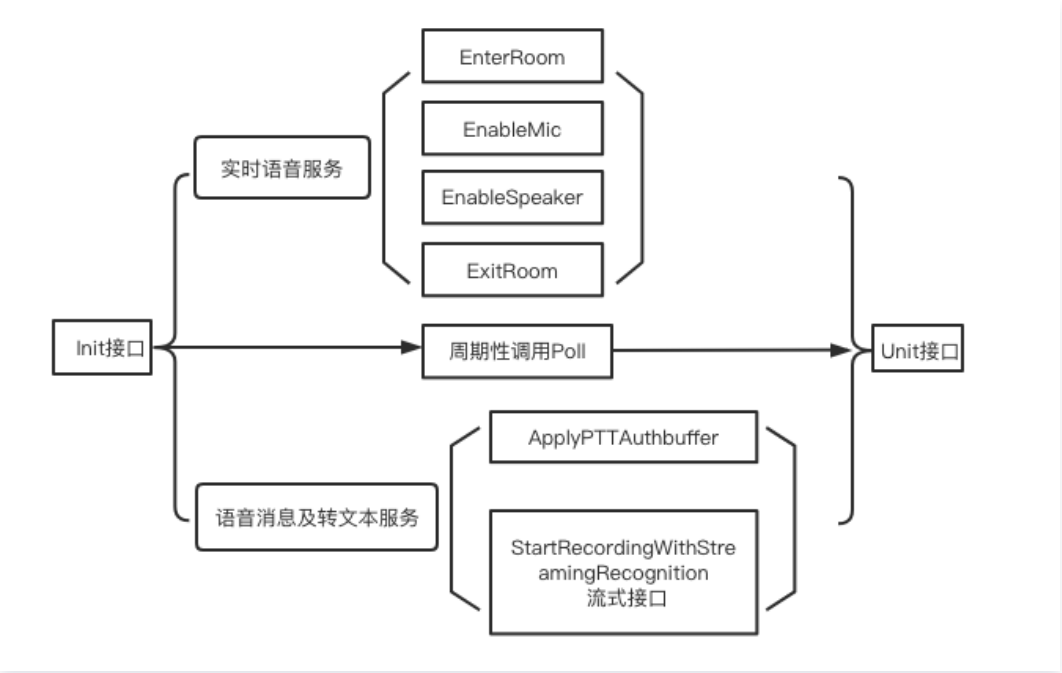
为方便 iOS 开发者调试和接入腾讯云游戏多媒体引擎产品 API，本文为您介绍适用于 macOS 开发的接入流程。

> **说明**
> 此文档对应 GME sdk version：2.9。

使用 GME 重要事项

GME 分为两个部分，提供实时语音服务、语音消息及转文本服务，使用这两个服务都依赖 Init 和 Poll 等核心接口。

> **注意**
> 语音转文本相关接口有默认频率限制，限额范围内计费方式请参见 计费文档；若需提升接口频率限额或了解超额计费方式，请联系商务或 提交工单咨询 。
> - 语音消息非流式转文本接口 **SpeechToText()**：默认单账号限制并发数为10路
> - 语音消息流式转文本接口 **StartRecordingWithStreamingRecognition()**：默认单账号限制并发数为50路
> - 实时语音流式转文本接口 **StartRealTimeASR()**：默认单账号限制并发数为50路



> **关于 Init 接口**
> 例如使用了实时语音服务，同时也需要使用语音消息服务，**只需要调用一次 Init 初始化接口**。
> Init 之后不会开始计费，语音消息及转文本服务**收发语音消息**才算作语音消息 DAU。

重要步骤

1. 初始化 GME
2. 周期性调用 Poll 触发回调
3. 鉴权初始化
4. 启动流式语音识别

5. 停止录制
6. 反初始化 GME

重点提示

- GME 使用前请对工程进行配置，否则 SDK 不生效。
- GME 的接口调用成功后返回值为 QAVError.OK，数值为 0。
- GME 的接口调用要在同一个线程下。
- GME 需要周期性的调用 Poll 接口触发事件回调。
- 错误码详情可参见 错误码 。

核心接口

未初始化前，SDK 处于未初始化阶段，需要通过接口 Init 初始化 SDK，才可以使用实时语音服务、语音消息及转文字服务。
在使用 GME 的任何接口之前，都需要先调用 Init 接口。
使用问题可参见 一般性问题 。

接口	接口含义
InitEngine	初始化 GME
Poll	触发事件回调
Pause	系统暂停
Resume	系统恢复
Uninit	反初始化 GME
SetDefaultAudienceAudioCategory	设置设备后台播放声音

引用头文件

```
#import "GMESDK/TMGEngine.h"
#import "GMESDK/QAVAuthBuffer.h"
```

获取单例

在使用语音功能时，需要首先获取 ITMGContext 对象。

```
+ (ITMGContext*) GetInstance;
```

示例代码

```
//TMGSampleViewController.m
ITMGContext* _context = [ITMGContext GetInstance];
```

设置回调

接口类采用 Delegate 方法用于向应用程序发送回调通知。将回调函数注册给 SDK，用于接收回调的信息。

示例代码

声明采用 ITMGDelegate。

```
@interface TMGDemoViewController ()<ITMGDelegate>{}
ITMGDelegate < NSObject >

//TMGSampleViewController.m
ITMGContext* _context = [ITMGContext GetInstance];
_context.TMGDelegate = [DispatchCenter getInstance];
```

在 OnEvent 中处理接口的回调信息，消息类型参考 ITMG_MAIN_EVENT_TYPE，消息内容为一个字典，解析出接口回调的内容。

函数原型

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

示例代码

```
//TMGRealTimeViewController.m
TMGRealTimeViewController ()< ITMGDelegate >


- (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) {
        // Step 6/11 : Perform the enter room event
        case ITMG_MAIN_EVENT_TYPE_ENTER_ROOM: {
            int result = ((NSNumber *)[data objectForKey:@"result"]).intValue;
            NSString *error_info = [data objectForKey:@"error_info"];

            [self showLog:[NSString stringWithFormat:@"OnEnterRoomComplete:%d msg:(%@)", result,
error_info]];

            if (result == 0) {
                [self updateStatusEnterRoom:YES];
            }
        }
        break;
    }
}

//需要参考 DispatchCenter.h、DispatchCenter.m
```

初始化 SDK

- 此接口用于初始化 GME 服务，建议应用侧在应用初始化时候调用，调用此接口不会产生计费。
- **参数 sdkAppID 获取请参见 语音服务开通指引**。
- openID 用于唯一标识一个用户，目前只支持 INT64，规则由 App 开发者自行制定，App 内不重复即可。

> ⚠ **注意**
> 调用 Init 接口的线程必须于其他接口在同一线程。建议都在主线程调用接口。

函数原型

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

参数	类型	含义
sdkAppId	String	来自 腾讯云控制台 的 GME 服务提供的 AppId。
OpenId	String	OpenId 只支持 Int64 类型（转为 string 传入）。

返回值	处理
QAV_OK= 0	初始化 SDK 成功
QAV_ERR_SDK_NOT_FULL_UPDATE= 7015	检查 SDK 文件是否完整，建议删除后重新导入 SDK

出现返回值 AV_ERR_SDK_NOT_FULL_UPDATE 时，此返回值只有提示作用，并不会造成初始化失败。

- 如果在接入过程中提示此错误，请根据提示检查 SDK 文件是否完整、SDK 文件版本是否一致。
- 如果是在导出可执行文件之后出现此返回值，请忽略此错误，并尽量不在 UI 中提示。

示例代码

```
_openId = _userIdText.text;
_appId = _appIdText.text;
[[ITMGContext GetInstance] InitEngine:SDKAPPID openID:_openId];
```

触发事件回调

通过在 update 里面周期的调用 Poll 可以触发事件回调。GME 需要周期性的调用 Poll 接口触发事件回调。如果没有调用 Poll 的话，会导致整个 SDK 服务运行异常。
可参见 Demo 中的 EnginePollHelper.m 文件。

> **注意**
> 务必周期性调用 Poll 接口且在主线程调用，以免接口回调异常。

函数原型

```
-(void)Poll;
```

示例代码

```
[[ITMGContext GetInstance] Poll];
```

系统暂停

当系统发生 Pause 事件时，需要同时通知引擎进行 Pause。
如果需要在退后台的时候暂停语音，可以在退后台的监听代码中调用 Pause 接口，恢复前台的监听事件中调用 Resume 接口。

函数原型

```
-(QAVResult)Pause;
```

系统恢复

当系统发生 Resume 事件时，需要同时通知引擎进行 Resume。Resume 接口只恢复实时语音。

函数原型

```
-(QAVResult)Resume;
```

反初始化 SDK

反初始化 SDK，进入未初始化状态。**切换账号需要反初始化**。

函数原型

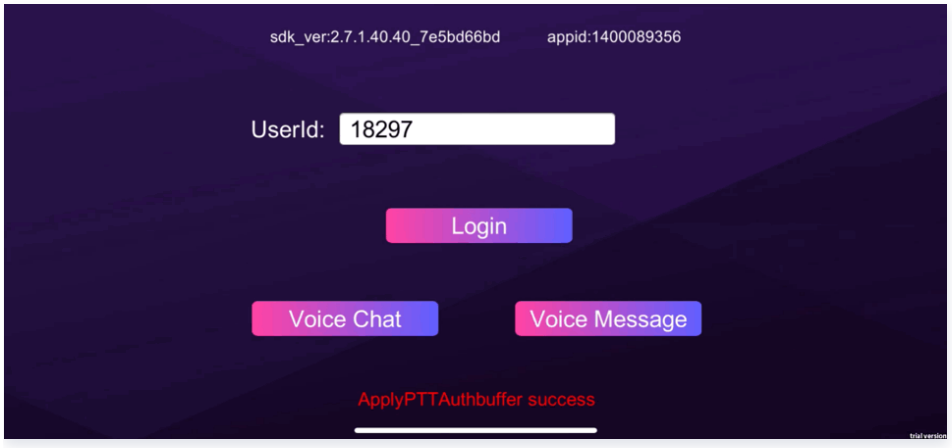
```
-(int)Uninit;
```

示例代码

```
[[ITMGContext GetInstance] Uninit];
```

语音消息及转文字

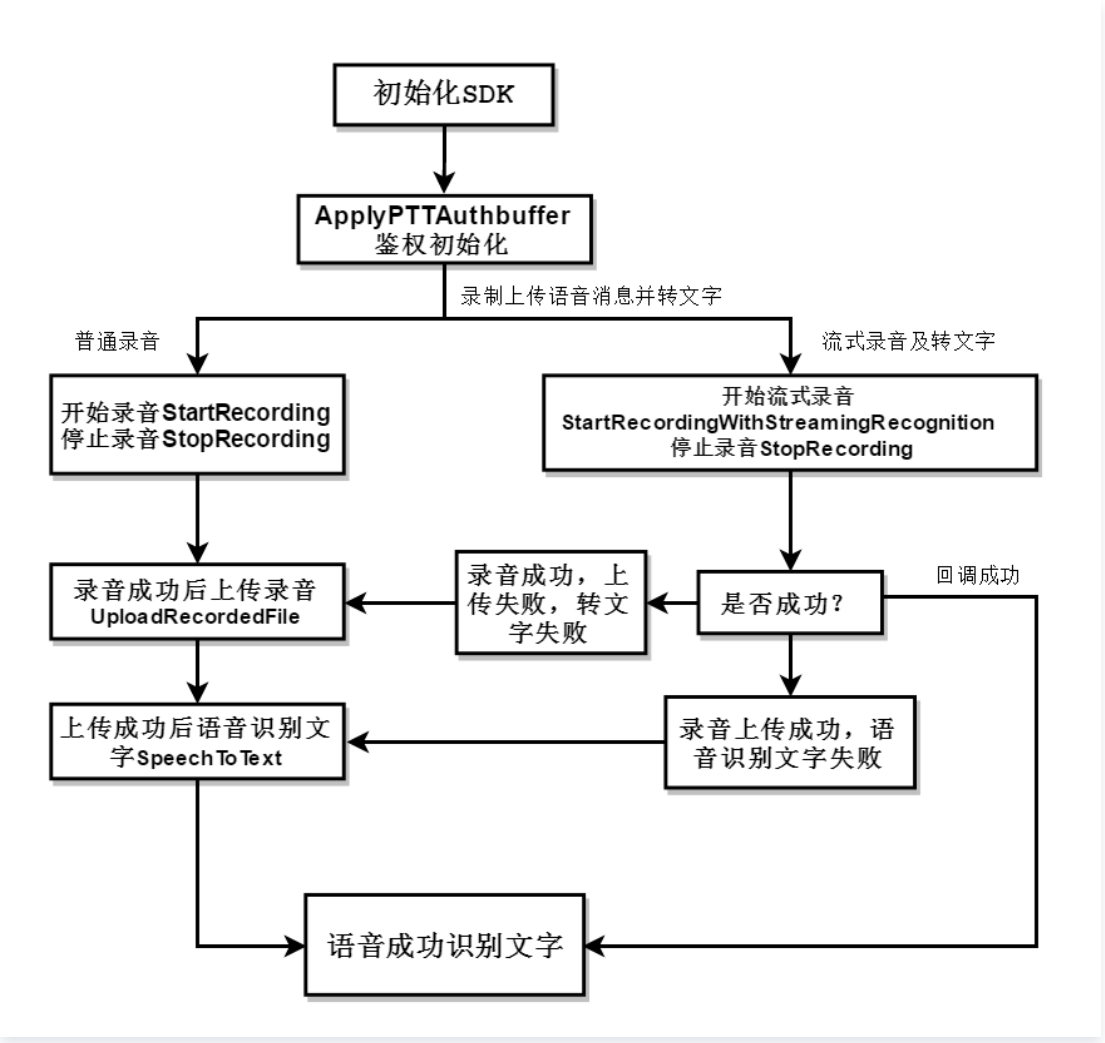
语音消息，录制并发送一段语音消息，同时可以将语音消息转成文字，也可以同时将文字进行翻译。下图演示的是语音消息及转文本服务：



说明

- 建议使用流式语音转文字服务。
- 使用语音消息服务不需要进入实时语音房间。

语音消息及语音转文字流程图



接入语音消息及转文本服务

语音消息及转文字相关接口

接口	接口含义
ApplyPTTAuthbuffer	鉴权初始化
SetMaxMessageLength	限制最大语音信息时长
StartRecording	启动录音
StartRecordingWithStreamingRecognition	启动流式录音
PauseRecording	暂停录音
ResumeRecording	恢复录音
StopRecording	停止录音
CancelRecording	取消录音
GetMicLevel	获取实时麦克风音量
SetMicVolume	设置录制音量

版权所有：腾讯云计算（北京）有限责任公司

GetMicVolume	获取录制音量
GetSpeakerLevel	获取实时扬声器音量
SetSpeakerVolume	设置播放音量
GetSpeakerVolume	获取播放音量
UploadRecordedFile	上传语音文件
DownloadRecordedFile	下载语音文件
PlayRecordedFile	播放语音
StopPlayFile	停止播放语音
GetFileSize	语音文件的大小
GetVoiceFileDuration	语音文件的时长
SpeechToText	语音识别成文字

最大录制时长
语音消息最大录制时长默认为58秒，最短不能小于1秒。如果需要再加以限制，例如限制为最大录制时长为10秒，请在初始化之后调用 SetMaxMessageLength 接口进行设置。

初始化 SDK

未初始化前，SDK 处于未初始化阶段，需要通过接口 Init 初始化 SDK，才可以使用实时语音及语音消息服务。
使用问题可参见 离线语音相关问题 。

鉴权信息

生成 AuthBuffer，用于相关功能的加密和鉴权，如正式发布请使用后台部署密钥，后台部署请参见 鉴权密钥 。
使用语音消息及转文字服务获取鉴权时，房间号参数必须填 null。

函数原型

```
@interface QAVAuthBuffer : NSObject
+ (NSData*) GenAuthBuffer:(unsigned int)appId roomId:(NSString*)roomId openID:(NSString*)openID
key:(NSString*)key;
+ @end
```

参数	类型	含义
appId	int	来自 腾讯云控制台 的 AppId 号码。
roomId	NSString	填 null
openID	NSString	用户标识。与 Init 时候的 openID 相同。
key	NSString	来自 腾讯云控制台 的权限密钥。

示例代码

```
#import "GMESDK/QAVAuthBuffer.h"
```

```
NSData* authBuffer = [QAVAuthBuffer GenAuthBuffer:SDKAPPID3RD.intValue roomId:_roomId
openID:_openId key:AUTHKEY];
```

鉴权初始化

在初始化 SDK 之后调用鉴权初始化，authBuffer 的获取参见上文实时语音鉴权信息接口 genAuthBuffer。

函数原型

```
public abstract int ApplyPTTAuthbuffer(byte[] authBuffer);
```

参数	类型	含义
authBuffer	NSData*	鉴权

示例代码

```
[[[ITMGContext GetInstance]GetPTT]ApplyPTTAuthbuffer:(NSData *)authBuffer];
```

流式语音识别

启动流式语音识别

此接口用于启动流式语音识别，同时在回调中会有实时的语音转文字返回，可以指定语言进行识别，也可以将语音中识别到的信息翻译成指定的语言返回。**停止录音调用 StopRecording**，停止之后才有回调。

函数原型

```
-(int)StartRecordingWithStreamingRecognition:(NSString *)filePath;
-(int)StartRecordingWithStreamingRecognition:(NSString *)filePath language:
(NSString*)speechLanguage translatelanguage:(NSString*)translateLanguage;
```

参数	类型	含义
filePath	String	存放的语音路径
speechLanguage	String	识别成指定文字的语言参数，参数请参见 语音转文字的语言参数参考列表
translateLanguage	String	翻译成指定文字的语言参数，参数请参见 语音转文字的语言参数参考列表（此参数暂不可用，请填写与 speechLanguage 相同的参数）

示例代码

```
recordfilePath = [docDir stringByAppendingFormat:@"/test_%d.ptt",index++];
[[[ITMGContext GetInstance] GetPTT] StartRecordingWithStreamingRecognition:recordfilePath
language:@"cmn-Hans-CN"];
```

流式语音识别的回调

启动流式语音识别后，需要在回调函数 OnEvent 中监听回调消息，事件消息分为以下两个：

- `ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_COMPLETE` 是在停止录制并完成识别后才返回文字，相当于一段话说完才会返回识别的文字。
- `ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_IS_RUNNING` 是在录音过程中就会实时返回识别到的文字，相当于边说话边返回识别到的文字。

根据需求在 OnEvent 函数中对相应事件消息进行判断。传递的参数包含以下四个信息。

消息名称	含义
result	用于判断流式语音识别是否成功的返回码
text	语音转文字识别的文本
file_path	录音存放的本地地址
file_id	录音在后台的 url 地址，录音在服务器存放 90 天。fileid 固定字段为 `http://gme-v2-`

> **注意**
> 监听 `ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_IS_RUNNING` 消息时，file_id 为空。

错误码

错误码	含义	处理方式
32775	流式语音转文本失败，但是录音成功	调用 UploadRecordedFile 接口上传录音，再调用 SpeechToText 接口进行语音转文字操作
32777	流式语音转文本失败，但是录音成功，上传成功	返回的信息中有上传成功的后台 url 地址，调用 SpeechToText 接口进行语音转文字操作
32786	流式语音转文本失败	在流式录制状态当中，请等待流式录制接口执行结果返回

示例代码

```
- (void)OnEvent:(ITMG_MAIN_EVENT_TYPE)eventType data:(NSDictionary*)data
{
    NSNumber *number = [data objectForKey:@"result"];
    switch (eventType)
    {
        case ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_COMPLETE:
        {
            if (data != NULL &&[[data objectForKey:@"result"] intValue]== 0)
            {
                donwLoadUrlPath = data[@"file_id"];

                recordfilePath = [data objectForKey:@"file_path"];
                _localFileField.text = recordfilePath;

                _donwloadUrlField.text = [data objectForKey:@"file_id"] ;

                UITextField *_audiotoTextField =(UITextField*)objc_getAssociatedObject(self,
[PTT_AUDIO_TO_TEXT UTF8String]);
                _audiotoTextField.text = [data objectForKey:@"text"] ;
            }
```

```
        }
            break;
    }
}
```

语音消息录制

录制的流程为：录音 > 停止录音 > 录音回调返回 > 启动下一次录音。

限制最大语音信息时长

限制最大语音消息的长度，最大支持58秒。

函数原型

```
-(QAVResult)SetMaxMessageLength:(int)msTime
```

参数	类型	含义
msTime	int	语音时长，单位 ms，区间为 1000 < msTime < 58000

示例代码

```
[[[ITMGContext GetInstance]GetPTT]SetMaxMessageLength:(int)msTime];
```

启动录音

此接口用于启动录音。需要将录音文件上传后才可以进行语音转文字等操作。**停止录音调用 StopRecording。**

函数原型

```
-(int)StartRecording:(NSString*)filePath;
```

参数	类型	含义
filePath	NSString	存放的语音路径

示例代码

```
recordfilePath =[docDir stringByAppendingFormat:@"/test_%d.ptt",index++];
[[[ITMGContext GetInstance]GetPTT]StartRecording:recordfilePath];
```

停止录音

此接口用于停止录音。此接口为异步接口，停止录音后会有录音完成回调，成功之后录音文件才可用。

函数原型

```
-(QAVResult)StopRecording;
```

示例代码

```
[[[ITMGContext GetInstance]GetPTT]StopRecording];
```

启动录音的回调

录音完成的回调，通过委托传递消息。
停止录音调用StopRecording。停止录音后才有启动录音的回调。
启动录音完成后的回调调用函数 OnEvent，事件消息为 ITMG_MAIN_EVNET_TYPE_PTT_RECORD_COMPLETE，在 OnEvent 函数中对事件消息进行判断。
传递的参数包含两个信息，一个是 result，另一个是 file_path。

错误码

错误码值	原因	建议方案
4097	参数为空	检查代码中接口参数是否正确
4098	初始化错误	检查设备是否被占用，或者权限是否正常，是否初始化正常
4099	正在录制中	确保在正确的时机使用 SDK 录制功能
4100	没有采集到音频数据	检查麦克风设备是否正常
4101	录音时，录制文件访问错误	确保文件存在，文件路径的合法性
4102	麦克风未授权错误	使用 SDK 需要麦克风权限，添加权限请参考对应引擎或平台的 SDK 工程配置文档
4103	录音时间太短错误	首先，限制录音时长的单位为毫秒，检查参数是否正确；其次，录音时长要1000毫秒以上才能成功录制
4104	没有启动录音操作	检查是否已经调用启动录音接口

示例代码

```
-(void)OnEvent:(ITMG_MAIN_EVENT_TYPE)eventType data:(NSDictionary *)data{
    NSLog(@"OnEvent:%lu,data:%@",(unsigned long)eventType,data);
    switch (eventType) {
        case ITMG_MAIN_EVNET_TYPE_PTT_RECORD_COMPLETE:
        {
        //录音回调
        }
            break;
    }
}
```

暂停录音

此接口用于暂停录音。如需恢复录音请调用接口 ResumeRecording。

函数原型

```
-(int)PauseRecording;
```

示例代码

```
[[[ITMGContext GetInstance]GetPTT]PauseRecording;
```

恢复录音

此接口用于恢复录音。

函数原型

```
-(int)ResumeRecording;
```

示例代码

```
[[[ITMGContext GetInstance]GetPTT]ResumeRecording;
```

取消录音

调用此接口取消录音。取消之后没有回调。

函数原型

```
-(QAVResult)CancelRecording;
```

示例代码

```
[[[ITMGContext GetInstance]GetPTT]CancelRecording];
```

获取语音消息麦克风实时音量

此接口用于获取麦克风实时音量，返回值为 int 类型，值域为0 – 200。

> **说明**
> 此接口不同于实时语音接口，此接口在 ITMGPTT 下。

函数原型

```
-(QAVResult)GetMicLevel;
```

示例代码

```
[[[ITMGContext GetInstance]GetPTT]GetMicLevel];
```

设置语音消息录制音量

此接口用于设置离线语音录制音量，值域为0 – 200。

> **说明**
> 此接口不同于实时语音接口，此接口在 ITMGPTT 下。

函数原型

```
-(QAVResult)SetMicVolume:(int) volume;
```

示例代码

```
[[[ITMGContext GetInstance]GetPTT]SetMicVolume:100];
```

获取语音消息录制音量

此接口用于获取离线语音录制音量。返回值为 int 类型，值域为0 – 200。

> **说明**
> 此接口不同于实时语音接口，此接口在 ITMGPTT 下。

函数原型

```
-(int)GetMicVolume;
```

示例代码

```
[[[ITMGContext GetInstance]GetPTT]GetMicVolume];
```

获取语音消息扬声器实时音量

此接口用于获取扬声器实时音量。返回值为 int 类型，值域为0 – 200。

> **说明**
> 此接口不同于实时语音接口，此接口在 ITMGPTT 下。

函数原型

```
-(QAVResult)GetSpeakerLevel;
```

示例代码

```
[[[ITMGContext GetInstance]GetPTT]GetSpeakerLevel];
```

设置语音消息播放音量

此接口用于设置语音消息播放音量，值域为0 – 200。

> **说明**
> 此接口不同于实时语音接口，此接口在 ITMGPTT 下。

函数原型

```
-(QAVResult)SetSpeakerVolume:(int)volume;
```

示例代码

```
[[[ITMGContext GetInstance]GetPTT]SetSpeakerVolume:100];
```

获取语音消息播放音量

此接口用于获取语音消息播放音量。返回值为 int 类型，值域为0 - 200。

> **说明**
> 此接口不同于实时语音接口，此接口在 ITMGPTT 下。

函数原型

```
-(int)GetSpeakerVolume;
```

示例代码

```
[[[ITMGContext GetInstance]GetPTT]GetSpeakerVolume];
```

语音消息播放

播放语音

此接口用于播放语音。

函数原型

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

参数	类型	含义
downloadFilePath	NSString	本地语音文件的路径
type	ITMG_VOICE_TYPE	变声类型，请参见 变声接入文档

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

示例代码

```
[[[ITMGContext GetInstance]GetPTT]PlayRecordedFile:path];
```

播放语音的回调

播放语音的回调，事件消息为 ITMG_MAIN_EVNET_TYPE_PTT_PLAY_COMPLETE， 在 OnEvent 函数中对事件消息进行判断。传递的参数包含两个信息，result 和 file_path。

错误码

错误码	原因	建议方案

值		
20481	初始化错误	检查设备是否被占用，或者权限是否正常，是否初始化正常
20482	正在播放中，试图打断并播放下一个失败了（正常是可以打断的）	检查代码逻辑是否正确
20483	参数为空	检查代码中接口参数是否正确
20484	内部错误	初始化播放器错误，解码失败等问题产生此错误码，需要结合日志定位问题

示例代码

```
-(void)OnEvent:(ITMG_MAIN_EVENT_TYPE)eventType data:(NSDictionary *)data{
    NSLog(@"OnEvent:%lu,data:%@",(unsigned long)eventType,data);
    switch (eventType) {
        case ITMG_MAIN_EVNET_TYPE_PTT_PLAY_COMPLETE:
        {
        //播放语音的回调
        }
            break;
    }
}
```

停止播放语音

此接口用于停止播放语音。停止播放语音也会有播放完成的回调。

函数原型

```
-(int)StopPlayFile;
```

示例代码

```
[[[ITMGContext GetInstance]GetPTT]StopPlayFile];
```

获取语音文件的大小

通过此接口，获取语音文件的大小。

函数原型

```
-(int)GetFileSize:(NSString*)filePath;
```

参数	类型	含义
filePath	NSString	语音文件的路径，此路径为本地路径

示例代码

```
[[[ITMGContext GetInstance]GetPTT]GetFileSize:path];
```

获取语音文件的时长

此接口用于获取语音文件的时长，单位毫秒。

函数原型

```
-(int)GetVoiceFileDuration:(NSString*)filePath;
```

参数	类型	含义
filePath	NSString	语音文件的路径，此路径为本地路径

示例代码

```
[[[ITMGContext GetInstance]GetPTT]GetVoiceFileDuration:path];
```

语音消息上传及下载

上传语音文件

此接口用于上传语音文件。

函数原型

```
-(void)UploadRecordedFile:(NSString*)filePath;
```

参数	类型	含义
filePath	NSString	上传的语音路径，此路径为本地路径

示例代码

```
[[[ITMGContext GetInstance]GetPTT]UploadRecordedFile:path];
```

上传语音完成的回调

上传语音完成后，事件消息为 ITMG_MAIN_EVNET_TYPE_PTT_UPLOAD_COMPLETE，在 OnEvent 函数中对事件消息进行判断。传递的参数包含三个信息，result，file_path 和 file_id。

错误码

错误码值	原因	建议方案
8193	上传文件时，文件访问错误	确保文件存在，文件路径的合法性
8194	签名校验失败错误	检查鉴权密钥是否正确，检查是否有初始化离线语音
8195	网络错误	检查设备网络是否可以正常访问外网环境
8196	获取上传参数过程中网络失败	检查鉴权是否正确，检查设备网络是否可以正常访问外网环境
8197	获取上传参数过程中回包数据为空	检查鉴权是否正确，检查设备网络是否可以正常访问外网环境
8198	获取上传参数过程中回包解包失败	检查鉴权是否正确，检查设备网络是否可以正常访问外网环境
8200	没有设置 appinfo	检查 apply 接口是否有调用，或者入参是否为空

示例代码

```
-(void)OnEvent:(ITMG_MAIN_EVENT_TYPE)eventType data:(NSDictionary *)data{
    NSLog(@"OnEvent:%lu,data:%@",(unsigned long)eventType,data);
    switch (eventType) {
        case ITMG_MAIN_EVNET_TYPE_PTT_UPLOAD_COMPLETE:
        {
            if (data != NULL &&[[data objectForKey:@"result"] intValue]== 0)
            {
                _donwloadUrlField.text = [data objectForKey:@"file_id"] ;
                donwLoadUrlPath = [data objectForKey:@"file_id"] ;
            }
        }
            break;
    }
}
```

下载语音文件

此接口用于下载语音文件。

函数原型

```
-(void)DownloadRecordedFile:(NSString*)fileId downloadFilePath:(NSString*)downloadFilePath
```

参数	类型	含义
fileID	NSString	文件的 url 路径
downloadFilePath	NSString	文件的本地保存路径

示例代码

```
[[[ITMGContext GetInstance]GetPTT]DownloadRecordedFile:fileIdpath downloadFilePath:path];
```

下载语音文件完成回调

下载语音完成后，事件消息为 ITMG_MAIN_EVNET_TYPE_PTT_DOWNLOAD_COMPLETE， 在 OnEvent 函数中对事件消息进行判断。
传递的参数包含三个信息，result、file_path 和 file_id。

错误码

错误码值	原因	建议方案
12289	下载文件时，文件访问错误	检查文件路径是否合法
12290	签名校验失败	检查鉴权密钥是否正确，检查是否有初始化离线语音
12291	网络存储系统异常	服务器获取语音文件失败，检查接口参数 fileid 是否正确，检查网络是否正常，检查 cos 文件存不存在
12292	服务器文件系统错误	检查设备网络是否可以正常访问外网环境，检查服务器上是否有此文件
12293	获取下载参数过程中，HTTP 网络失败	检查设备网络是否可以正常访问外网环境

12294	获取下载参数过程中，回包数据为空	检查设备网络是否可以正常访问外网环境
12295	获取下载参数过程中，回包解包失败	检查设备网络是否可以正常访问外网环境
12297	没有设置 appinfo	检查鉴权密钥是否正确，检查是否有初始化离线语音

示例代码

```
-(void)OnEvent:(ITMG_MAIN_EVENT_TYPE)eventType data:(NSDictionary *)data{
    NSLog(@"OnEvent:%lu,data:%@",(unsigned long)eventType,data);
    switch (eventType) {
        case ITMG_MAIN_EVNET_TYPE_PTT_DOWNLOAD_COMPLETE:
        {
            if (data != NULL &&[[data objectForKey:@"result"] intValue]== 0)
            {
                _audiofileToPlayField.text = [data objectForKey:@"file_path"] ;
                donwLoadLocalPath = [data objectForKey:@"file_path"];
            }
            else
            {
                donwLoadLocalPath = NULL;
            }
        }
            break;
    }
}
```

语音转文字服务

将指定的语音文件识别成文字

此接口用于将指定的语音文件识别成文字。

函数原型

```
-(void)SpeechToText:(NSString*)fileID;
```

参数	类型	含义
fileID	NSString	语音文件 url

示例代码

```
[[[ITMGContext GetInstance]GetPTT]SpeechToText:fileID];
```

将指定的语音文件翻译成文字（指定语言）

此接口可以指定语言进行识别，也可以将语音中识别到的信息翻译成指定的语言返回。

函数原型

```
-(void)SpeechToText:(NSString*)fileID (NSString*)speechLanguage (NSString*)translateLanguage;
```

参数	类型	含义
fileID	NSString*	语音文件 url，录音在服务器存放 90 天
speechLanguage	NSString*	识别出指定文字的语言参数，参数请参见 语音转文字的语言参数参考列表
translateLanguage	NSString*	翻译成指定文字的语言参数，参数请参见 语音转文字的语言参数参考列表（此参数暂时无效，填入参数应与 speechLanguage 一致）

示例代码

```
[[[ITMGContext GetInstance]GetPTT]SpeechToText:fileID speechLanguage:"cmn-Hans-CN"
translateLanguage:"cmn-Hans-CN"];
```

识别回调

将指定的语音文件识别成文字的回调，事件消息为 ITMG_MAIN_EVNET_TYPE_PTT_SPEECH2TEXT_COMPLETE，在 OnEvent 函数中对事件消息进行判断。
传递的参数包含三个信息，result、file_path 和 text，其中 text 为识别的文本。

错误码

错误码值	原因	建议方案
32769	内部错误	分析日志，获取后台返回给客户端的真正错误码，并联系后台同事协助解决。
32770	网络失败	检查设备网络是否可以正常访问外网环境
32772	回包解包失败	分析日志，获取后台返回给客户端的真正错误码，并联系后台同事协助解决。
32774	没有设置 appinfo	检查鉴权密钥是否正确，检查是否有初始化离线语音
32776	authbuffer 校验失败	检查 authbuffer 是否正确
32784	语音转文本参数错误	检查代码中接口参数 fileid 是否为空
32785	语音转文本翻译返回错误	离线语音后台错误，请分析日志，获取后台返回给客户端的真正错误码，并联系后台同事协助解决

示例代码

```
-(void)OnEvent:(ITMG_MAIN_EVENT_TYPE)eventType data:(NSDictionary *)data{
    NSLog(@"OnEvent:%lu,data:%@",(unsigned long)eventType,data);
    switch (eventType) {
        case ITMG_MAIN_EVNET_TYPE_PTT_SPEECH2TEXT_COMPLETE:
        {
            if (data != NULL &&[[data objectForKey:@"result"] intValue]== 0)
            {
                UITextField *_audiotoTextField =(UITextField*)objc_getAssociatedObject(self,
[PTT_AUDIO_TO_TEXT UTF8String]);
                _audiotoTextField.text = [data objectForKey:@"text"] ;
```

```
            }
        }
            break;
    }
}
```

高级 API

获取版本号

获取 SDK 版本号，用于分析。

函数原型

```
-(NSString*)GetSDKVersion;
```

示例代码

```
[[ITMGContext GetInstance] GetSDKVersion];
```

检查麦克风权限

返回麦克风权限状态。

函数原型

```
-(ITMG_RECORD_PERMISSION)CheckMicPermission;
```

参数含义

参数	数值	含义
ITMG_PERMISSION_GRANTED	0	麦克风已授权
ITMG_PERMISSION_Denied	1	麦克风被禁用
ITMG_PERMISSION_NotDetermined	2	尚未弹出权限框向用户申请权限
ITMG_PERMISSION_ERROR	3	接口调用错误

示例代码

```
[[ITMGContext GetInstance] CheckMicPermission];
```

设置打印日志等级

用于设置打印日志等级。建议保持默认等级。需要在 Init 之前调用。

函数原型

```
-(void)SetLogLevel:(ITMG_LOG_LEVEL)levelWrite (ITMG_LOG_LEVEL)levelPrint;
```

参数含义

参数	类型	含义
levelWrite	ITMG_LOG_LEVEL	设置写入日志的等级，TMG_LOG_LEVEL_NONE 表示不写入，默认为 TMG_LOG_LEVEL_INFO
levelPrint	ITMG_LOG_LEVEL	设置打印日志的等级，TMG_LOG_LEVEL_NONE 表示不打印，默认为 TMG_LOG_LEVEL_ERROR

ITMG_LOG_LEVEL 说明

ITMG_LOG_LEVEL	含义
TMG_LOG_LEVEL_NONE	不打印日志
TMG_LOG_LEVEL_ERROR	打印错误日志（默认）
TMG_LOG_LEVEL_INFO	打印提示日志
TMG_LOG_LEVEL_DEBUG	打印开发调试日志
TMG_LOG_LEVEL_VERBOSE	打印高频日志

示例代码

```
[[ITMGContext GetInstance] SetLogLevel:TMG_LOG_LEVEL_INFO TMG_LOG_LEVEL_INFO];
```

设置打印日志路径

。用于设置打印日志路径。默认路径为： /Users/username/Library/Containers/xxx.xxx.xxx/Data/Documents。。需要在 Init 之前调用。

函数原型

```
-(void)SetLogPath:(NSString*)logDir;
```

参数	类型	含义
logDir	NSString	路径

示例代码

```
[[ITMGContext GetInstance] SetLogPath:Path];
```

回调消息

消息列表

消息	消息代表的含义
ITMG_MAIN_EVNET_TYPE_PTT_RECORD_COMPLETE	PTT 录音完成
ITMG_MAIN_EVNET_TYPE_PTT_UPLOAD_COMPLETE	上传 PTT 完成
ITMG_MAIN_EVNET_TYPE_PTT_DOWNLOAD_COMPLETE	下载 PTT 完成
ITMG_MAIN_EVNET_TYPE_PTT_PLAY_COMPLETE	播放 PTT 完成

版权所有：腾讯云计算（北京）有限责任公司

ITMG_MAIN_EVNET_TYPE_PTT_SPEECH2TEXT_COMPLETE	语音转文字完成

Data 列表

消息	Data	例子
ITMG_MAIN_EVNET_TYPE_PTT_RECORD_COMPLETE	result; file_path	{"file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_UPLOAD_COMPLETE	result; file_path;file_id	{"file_id":"","file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_DOWNLOAD_COMPLETE	result; file_path;file_id	{"file_id":"","file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_PLAY_COMPLETE	result; file_path	{"file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_SPEECH2TEXT_COMPLETE	result; text;file_id	{"file_id":"","text":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_COMPLETE	result; file_path; text;file_id	{"file_id":"","file_path":","text":"","result":0}

H5 SDK
工程配置

最近更新时间：2023-03-20 15:36:32

为方便 H5 开发者调试和接入腾讯云游戏多媒体引擎 API，这里向您介绍适用于 H5 开发的预备工作。

H5 SDK 支持的功能

功能	H5 平台是否支持
基础实时语音功能	支持
音效与伴奏	部分支持
3D 语音	不支持
范围语音	不支持
房间管理	不支持
音频转发路由	不支持
自定义消息通道	不支持
语音消息	不支持
语音转文本	不支持

H5 SDK 支持的平台

操作系统平台	浏览器/webview	版本要求	备注
iOS	Safari	11.1.2	使用高版本的 Safari。
	Chrome	-	iOS 下暂时不支持 Chrome 浏览器
	微信浏览器	-	iOS 下暂时不支持微信浏览器
Android	TBS（微信和手机 QQ 的默认 Webview）	43600	微信和手机 QQ 默认内置的浏览器内核为 TBS
	Chrome	60+	需要支持 H264
Mac	Chrome	47+	-
	Safari	11+	-
Windows(PC)	Chrome	52+	-
	QQ 浏览器	10.2	-

准备工作

您可以通过以下步骤获取 SDK

1. 请进入 下载指引 页面。
2. 在界面中找到 H5 版本的 SDK 资源。

版权所有：腾讯云计算（北京）有限责任公司

3. 在页面中单击**下载**，完成操作。

前端工程配置步骤

1. 端口号开放

如果所在网络有防火墙，请确定有开放以下端口：

协议	端口号
TCP	8687
UDP	8000 、8800 、443

使用 CDN 引入 SDK。

2. 引入前端库文件

参考以下代码，在工程中引入 WebRTCService.min.js。

```
<head>
    <script src="../dist/WebRTCService.min.js"></script>
</head>
```

3. 添加 Audio 标签

```
<div id="gme-audio-wrap"></div>
```

服务器端部署步骤

使用 GME SDK 需要鉴权，鉴权部分涉及密钥，不适合在客户端实现，建议单独业务部署。
如暂时只需客户端实现，可参考我们提供的 Demo 工程。

1. 下载程序

请 下载 我们为您准备的 authBuffer 示例程序，该程序可以完成对指定的 SDKAppID 的鉴权信息签名。

2. 配置服务端鉴权工程

进入 signdemo 目录，修改 config.js 文件：打开 config.js 文件，先删除默认的配置，在删除代码处，调用 appidMap 函数（参数为在腾讯云后台申请的 SDKAppid 以及对应的鉴权 key）。

```
const AuthBufferConfig = function () {
    this.appidMap = {};
    this.appidMap["1400089356"] = "1cfbfd2a1a03a53e";
};
//将1400089356替换为在腾讯云后台申请的 sdkAppid，1cfbfd2a1a03a53e 替换为对应 sdkAppid 的鉴权 key
```

> **注意**
> AuthKey 必须与您的 SDKAppid 相对应。

3. 部署服务端鉴权工程

进入 authBuffer 示例程序目录，执行以下语句，以安装相关依赖：

```
npm i
```

然后执行脚本 node index.js，运行签名服务。

> ⚠ **注意**
> 由于使用到 async 语法，请确保您的 node 版本在8以上。命令行中执行 node -v 以查看版本。

4. 测试部署结果

可在命令行用以下命令测试（确保系统中有 curl 指令）：

```
//生成userSig:
curl "http://127.0.0.1:10005/" --data "sdkappid=1400089356&roomid=1234123&openid=1234567"
```

执行签名程序后，签名程序会返回鉴权信息，返回参考如下：

```
{"userSig":"AqhHE7QHLFYPfV/zfyrdRYHfuUn6eOA8g/J6GMjVy//Shr5ByJPTi8hzR2KyXMvn","errorCode":0}
```

接口文档

最近更新时间：2023-06-06 09:12:31

> **注意**
> H5 SDK 仅支持 GME 部分产品能力，支持的接口以此文档为准。请您阅读此文档并评估 H5 SDK 是否适用于您的业务场景。

接口	接口含义
Init	初始化接口
SetTMGDelegate	设置委托
EnterRoom	进入语音房间
EnableMic	开启或关闭采集设备
EnableSpeaker	开启或关闭播放设备
SetMicVolume	设置麦克风音量
ExitRoom	退出语音房间

> **说明**
> - GME 的接口调用成功后返回值为 QAVError.OK，数值为0。
> - GME 加入房间需要鉴权，请参考文档关于鉴权部分内容。
> - 设备的操作要在进房成功之后。
> - Chrome74 以后严格限制 navigator.mediaDevices 在 HTTPS 环境下访问，所以请使用 HTTPS 环境。

接入 JQ

使用 Demo 需要接入 JQ。

```
<!--Step 2: 添加Audio容器-->
<!--容器，用来承接Audio标签， 请务必留意，不能忽略-->
<div id="gme-audio-wrap"></div>
```

初始化相关接口

未初始化前，SDK 处于未初始化阶段，需要初始化鉴权后，通过初始化 SDK，才可以进房。

初始化 SDK

参数获取请参考 接入指引 。
此接口需要来自腾讯云控制台的 SDKAppID 号码作为参数，再加上 openId，这个 openId 是唯一标识一个用户，规则由 App 开发者自行制定，App 内不重复即可（目前只支持 INT64）。

> **注意**
> 初始化 SDK 之后才可以进房。

函数原型

```
WebGMEAPI.fn.Init = function (document, SdkAppId, openId) {...}
```

参数	意义
document	HTML DOM Document 对象
SdkAppId	来自腾讯云控制台的 SdkAppId 号码
openId	用户的账号，由开发者定义，必须大于10000，用于标识用户

示例代码

```
const cSdkAppId = () => document.getElementById("input-SdkAppId").value;
const cOpenID = () => document.getElementById("input-OpenID").value;
gmeAPI.Init(document, cSdkAppId(), cOpenID());
```

设置回调

接口类采用 Delegate 方法用于向应用程序发送回调通知。将回调函数注册给 SDK，用于接收回调的信息。将回调函数注册给 SDK，要在进房之前设置。

函数原型

```
WebGMEAPI.fn.SetTMGDelegate = function (delegate){...}
```

参数	意义
onEvent	SDK 回调事件

示例代码

```
gmeAPI.SetTMGDelegate(onEvent);
```

实时语音相关接口

初始化之后，SDK 调用进房后进去了房间，才可以进行实时语音通话。

加入房间

用生成的鉴权信息进房，会收到消息为 ITMG_MAIN_EVENT_TYPE_ENTER_ROOM 的回调。加入房间默认不打开麦克风及扬声器。

函数原型

```
WebGMEAPI.fn.EnterRoom = function (roomId, roomType, authBuffer) {...}
```

参数	意义
roomId	房间号，最大支持127字符
roomType	房间音频类型
authBuffer	鉴权码，获取方式请参考 工程配置

示例代码

```
function bindButtonEvents() {
        $("#start_btn").click(function () {
            console.log('start!');
            //步骤1，  获取 AuthBuffer
            var FetchSigCgi = 'http://134.175.146.244:10005/';
            $.ajax({
                type: "POST",
                url: FetchSigCgi,
                dataType: 'json',
                data: {
                    sdkappid: cSdkAppId(),
                    roomid: cRoomNum(),
                    openid: cOpenID(),
                },
                success: function (json) {
                    //步骤2，获取 AuthBuffer 成功
                    if (json && json.errorCode === 0) {
                        let userSig = json.userSig;
                        gmeAPI.Init(document, cSdkAppId(), cOpenID());
                        gmeAPI.SetTMGDelegate(onEvent);
                        gmeAPI.EnterRoom(cRoomNum(), 1, userSig);
                    } else {
                        console.error(json);
                    }
                },
                error: function (err) {
                    console.error(err);
                }
            });
        });
```

事件回调

加入房间完成后会发送信息 ITMG_MAIN_EVENT_TYPE_ENTER_ROOM，在 OnEvent 函数中进行判断。

示例代码

```
onEvent = function (eventType, result) {
         if (eventType === gmeAPI.event.ITMG_MAIN_EVENT_TYPE_ENTER_ROOM)
        {
            //进房成功
        }
        else if (eventType === gmeAPI.event.ITMG_MAIN_EVENT_TYPE_USER_UPDATE)
        {
            app._data.downStreamInfoList = result.PeerInfo;//接收的对端的信息 ，参见下表
            app._data.brSend = result.UploadBRSend;//上传语音数据的码率
            app._data.rtt = result.UploadRTT;//上传 RTT
        }
        else if (eventType === gmeAPI.event.ITMG_MAIN_EVENT_TYPE_EXIT_ROOM)
        {
            //退房成功
        }
        else if (eventType === gmeAPI.event.ITMG_MAIN_EVENT_TYPE_ROOM_DISCONNECT)
        {
```

```
            //房间断开连接
        }
    };
```

接收的对端的信息如下 downStreamInfoList：

参数	意义
brRecv	接收的码率
delay	接收的延迟
jitterBufferMs	抖动延迟
jitterReceived	接收 jitter

退出房间

通过调用此接口可以退出所在房间。这是一个异步接口，退房之后有回调，返回值为 AV_OK 的时候代表异步投递成功。

函数原型

```
WebGMEAPI.fn.ExitRoom = function (){...}
```

示例代码

```
gmeAPI.ExitRoom();
```

开启关闭麦克风

此接口用来开启关闭麦克风。加入房间默认不打开麦克风及扬声器。

函数原型

```
WebGMEAPI.fn.EnableMic = function (bEnable) {...}
```

参数	意义
isEnabled	如果需要打开麦克风，则传入的参数为 true，如果关闭麦克风，则参数为 false

示例代码

```
gmeAPI.EnableMic(false);
```

设置麦克风的音量

此接口用于设置麦克风的音量。参数 volume 用于设置麦克风的音量，当数值为0的时候表示静音，当数值为100的时候表示音量不增不减，默认数值为100。

函数原型

```
WebGMEAPI.fn.SetMicVolume = function (volume){...}
```

参数	意义
volume	设置音量，范围0到100

示例代码

```
gmeAPI.SetMicVolume(100);
```

开启关闭扬声器

此接口用于开启关闭扬声器。

函数原型

```
WebGMEAPI.fn.EnableSpeaker = function (bEnable){...}
```

参数	意义
isEnabled	如果需要关闭扬声器，则传入的参数为 false，如果打开扬声器，则参数为 true

示例代码

```
gmeAPI.EnableSpeaker(true);
```

Electron SDK
集成 SDK

最近更新时间：2023-03-20 15:36:32

为方便 Electron 开发者调试和接入腾讯云游戏多媒体引擎产品 API，本文档主要为您介绍适用于 Electron 开发的工程配置指引。

支持的平台

- Windows（PC）

导入SDK

步骤1：安装 Node.js

1. 根据 Windows 操作系统选择下载最新版本的 Node.js 安装包 Windows Installer (.msi) 64-bit。
2. 打开应用程序列表中的 Node.js command prompt，启动命令行窗口，用于输入后续步骤中的各项命令。

步骤2：安装 Electron

在命令行窗口中执行如下命令，安装 Electron，建议版本号 >= 4.0.0。

```
$ npm install electron -g
```

步骤3：安装 Electron 版的 GME SDK

1. 在您的 Electron 项目中使用 npm 命令安装 Gme SDK 包：

```
$ npm install gme-electron-sdk@latest --save
```

2. 在项目脚本里引入模块并使用：

```
const { GmeContext } = require('gme-electron-sdk');
// import gmesdk from 'gme-electron-sdk';
gmeContext = new GmeContext();
// 获取 SDK 版本号
gmeContext.GetSDKVersion();
```

步骤4：打包可执行程序

安装打包工具：推荐使用Electron Forge，您可以执行如下命令

1. 将 Electron Forge 添加到您应用的开发依赖中，并使用其"import"命令设置 Forge 的脚手架：

```
npm install --save-dev @electron-forge/cli
npx electron-forge import
```

2. 使用 Forge 的 make 命令来创建可分发的应用程序：

```
npm run make
```

实时语音

最近更新时间：2025-06-18 14:16:01

为方便 Electron 开发者调试和接入腾讯云游戏多媒体引擎客户端 API，本文为您介绍适用于 Electron 实时语音功能的开发接入技术文档。

使用 GME 重要事项

GME 提供实时语音服务、语音消息服务及转文本服务，使用 GME 服务都依赖 Init 和 Poll 等核心接口。

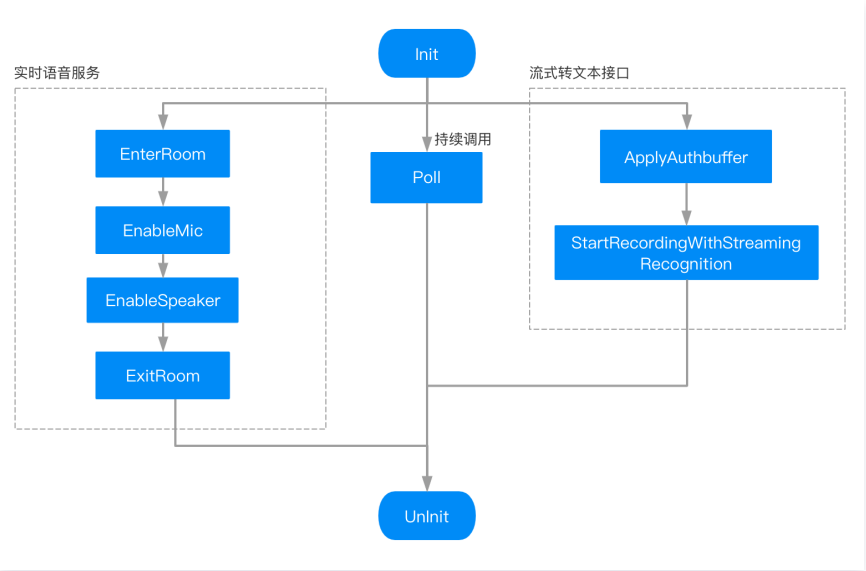
重点提示

- 已完成 GME 应用创建，并获取 SDK AppID 和 Key。请参见 服务开通指引 。
- 已开通 **GME 实时语音服务、语音消息服务以及转文本服务**。请参见 服务开通指引 。
- GME 使用前请对工程进行配置，否则 SDK 不生效。
- GME 的接口调用成功后返回值为 GmeError.AV_OK，数值为 0。
- GME 的接口调用要在同一个线程下。
- GME 需要周期性的调用 Poll 接口触发事件回调。
- 错误码详情可参见 错误码 。

接入 SDK

重要步骤

接入 SDK 重要流程如下：



1. 初始化 GME
2. 周期性调用 Poll 触发回调
3. 进入实时语音房间
4. 打开麦克风
5. 打开扬声器
6. 退出语音房间
7. 反初始化 GME

核心接口

接口	接口含义

Init	初始化 GME。
Poll	触发事件回调。
Pause	系统暂停。
Resume	系统恢复。
Uninit	反初始化 GME。

引用Gme模块

```
const { GmeContext } = require('gme-electron-sdk');
```

获取实例

在使用语音功能时，需要首先获取 GmeSDK 对象。

```
context = new GmeContext();
```

初始化 SDK

未初始化前，SDK 处于未初始化阶段，**需要通过接口 Init 初始化 SDK**，才可以使用实时语音服务、语音消息服务及转文本服务。调用 Init 接口的线程必须于其他接口在同一线程,建议都在主线程调用接口。

接口原型

```
//class GmeSDK
Init(appid: string, openid: string): number;
```

参数	类型	含义
sdkAppId	String	来自 腾讯云控制台 的 GME 服务提供的 AppID，获取请参见 语音服务开通指引 。
openID	String	openID 只支持int64 类型（转为 string 传入），规则由 App 开发者自行制定，App 内不重复即可。如需使用字符串作为 Openid 传入，可 提交工单 联系开发者。

返回值

返回值	处理
GmeError.AV_OK= 0	初始化 SDK 成功。
AV_ERR_SDK_NOT_FULL_UPDATE=7015	检查 SDK 文件是否完整，建议删除后重新导入 SDK。

⚠ 关于7015错误提示

- 7015错误码是通过 md5 进行判断，在接入过程中若出现此错误，请根据提示检查 SDK 文件是否完整、SDK 文件版本是否一致。
- 出现返回值 AV_ERR_SDK_NOT_FULL_UPDATE 时，此返回值**只有提示作用**，并不会造成初始化失败。
- 由于第三方加固、Unity 打包机制等因素会影响库文件 md5，造成误判，所以**正式发布请在逻辑中忽略此错误**，并尽量不在 UI 中提示。

示例代码

```
string SDKAPPID3RD = "14000xxxxx";
string openId="10001";
number ret = context.Init(SDKAPPID3RD, openId);
//通过返回值判断是否初始化成功
if (ret != GmeError.AV_OK)
{
        console.log("SDK初始化失败:");
        return;
}
```

设置回调

接口类采用 Delegate 方法用于向应用程序发送回调通知。将回调函数注册给 SDK，用于接收回调的信息，需要在进房之前设置。

函数原型及示例代码

设置回调，用于接收回调的信息，需要在进房之前设置。

```
SetTMGDelegate(cb: ITMGDelegate);
//在初始化 SDK 时候
context =  GmeSDK.GetInstance();
context.setTMGDelegate(function(eventId, msg){
  if (type == ITMG_MAIN_EVENT_TYPE_ENTER_ROOM)
  {
            //回调处理
  }
});
```

触发事件回调

需要周期的调用 Poll 触发事件回调。Poll 是 GME 的消息泵，GME 需要周期性的调用 Poll 接口触发事件回调。如果没有调用 Poll ，将会导致整个 SDK 服务运行异常。详情请参见 Sample Project 中的 EnginePollHelper 文件。

> **务必周期性调用 Poll 接口**
> 务必周期性调用 Poll 接口且在主线程调用，以免接口回调异常。

接口原型

```
Poll(): number;
```

示例代码

```
setInterval(function () {
      context.Poll();
    }, 50);
```

系统暂停

当系统发生 Pause 事件时，需要同时通知引擎进行 Pause。例如在应用退后台时候（OnApplicationPause, isPause=True），如果不需要后台播放房间内声音，请调用 Pause 接口暂停整个 GME 服务。

接口原型

```
Pause() :number
```

系统恢复

当系统发生 Resume 事件时，需要同时通知引擎进行 Resume。Resume 接口只恢复实时语音。

接口原型

```
Resume() :number
```

反初始化 SDK

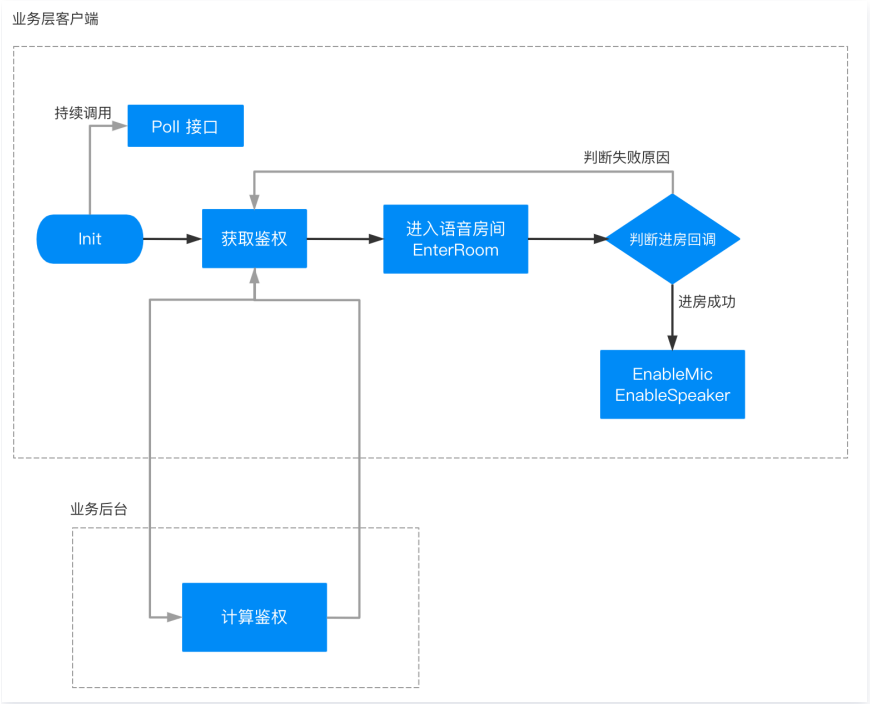
反初始化 SDK，进入未初始化状态。**如果游戏业务侧账号与 openid 是绑定的，那切换游戏账号需要反初始化 GME，再用新的 openid 初始化。**

接口原型

```
Uninit() : number;
```

实时语音房间相关接口

初始化之后，SDK 调用进房后进去了房间，才可以进行实时语音通话。
使用问题可参见 实时语音相关问题 。



接口	接口含义
GenAuthBuffer	本地鉴权计算。
EnterRoom	加入房间。
ExitRoom	退出房间。
IsRoomEntered	判断是否已经进入房间。

本地鉴权计算

生成 AuthBuffer，用于相关功能的加密和鉴权，如正式发布请使用后台部署密钥，后台部署请参见 鉴权密钥 。

接口原型

```
GenAuthBuffer(appId: string,roomId: string, openId:string, appKey: number) :string;
```

参数	类型	含义
appId	String	来自腾讯云控制台的 AppID 号码。
roomId	String	房间号，最大支持127字符。
openId	String	用户标识。与 Init 时候的 OpenID 相同。
key	Number	来自腾讯云 控制台 的权限密钥。

示例代码

```
let userSig = context.GenAuthBuffer(this.appid, this.roomId, this.userId, this.authKey)
context.EnterRoom(this.roomId, this.roomType, userSig);
```

加入房间

用生成的鉴权信息进房，加入房间默认不打开麦克风及扬声器。

> **注意**
> - 加入房间事件回调结果 result 为 0 代表进房成功，进房接口 EnterRoom 返回值为 0 不代表进房成功。
> - 房间的音频类型由第一个进房的人确定，此后房间里有成员修改房间类型，将对此房间所有成员生效。例如第一个进入房间的人使用的房间音频类型是流畅音质，第二个进房的是即使进房时候调用接口的音频类型参数是高清音质，进入房间之后也会变成流畅音质。需要有成员调用 ChangeRoomType 才会修改房间的音频类型。

接口原型

```
EnterRoom(roomid: string, roomType: number, appKey: string) :number;
```

参数	类型	含义
roomId	String	房间号，最大支持127字符。
roomType	ITMGRoomType	房间类型，游戏建议使用 ITMG_ROOM_TYPE_FLUENCY。房间音频类型请参见 音质选择 。
appKey	String	鉴权码。

示例代码

```
context.EnterRoom(roomID, ITMG_ROOM_TYPE_STANDARD, retAuthBuff);
```

加入房间事件回调

加入房间完成后会通过回调返回 ITMG_MAIN_EVENT_TYPE_ENTER_ROOM 事件类型返回进房结果，监听进房结果事件后进行处理。如果回调为成功，即此时进房成功，开始进行**计费**。

> **计费问题参考：**
> - 购买指南。
> - 计费相关问题。
> - 使用实时语音后，如果客户端掉线了，是否还会继续计费？

示例代码

```
//对事件进行监听：
 gmeContext.setTMGDelegate(function(eventId, msg){
  switch (eventId) {
      case ITMG_MAIN_EVENT_TYPE_ENTER_ROOM:
      {
      }
  }
});
```

Data 详情

消息	Data	例子
ITMG_MAIN_EVENT_TYPE_ENTER_ROOM	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_ROOM_DISCONNECT	result; error_info	{"error_info":"waiting timeout, please check your network","result":0}

如果断网，将会有断网的回调提示 `ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVENT_TYPE_ROOM_DISCONNECT`，此时 SDK 会自动进行重连，回调是 `ITMG_MAIN_EVENT_TYPE_RECONNECT_START`，当重连成功时，会有 `ITMG_MAIN_EVENT_TYPE_RECONNECT_SUCCESS` 回调。

错误码

错误码值	原因及建议方案
7006	鉴权失败原因： • AppID 不存在或者错误 • authbuff 鉴权错误 • 鉴权过期 • OpenId 不符合规范
7007	已经在其它房间。
1001	已经在进房过程中，然后又重复了此操作。建议在进房回调返回之前不要再调用进房接口。
1003	已经进房了在房间中，又调用一次进房接口。
1101	确保已经初始化 SDK，确保 OpenId 是否符合规则，或者确保在同一线程调用接口，以及确保 Poll 接口正常调用。

退出房间

通过调用此接口可以退出所在房间。这是一个异步接口，返回值为 AV_OK 的时候代表异步投递成功。如果应用中有退房后立即进房的场景，在接口调用流程上，开发者无需要等待 ExitRoom 的回调 RoomExitComplete 通知，只需直接调用接口。

接口原型

```
ExitRoom(): number;
```

示例代码

```
context.ExitRoom();
```

退出房间事件回调

退出房间完成后会有回调，消息为 ITMG_MAIN_EVENT_TYPE_EXIT_ROOM。示例代码如下：

示例代码

```
gmeContext.setTMGDelegate(function(eventId, msg){
  switch (eventId) {
      case ITMG_MAIN_EVENT_TYPE_EXIT_ROOM:
      {
              //进行处理
        break;
      }
  }
});
```

判断是否已经进入房间

通过调用此接口可以判断是否已经进入房间，返回值为 boolean 类型。请勿在进房过程中调用。

接口原型

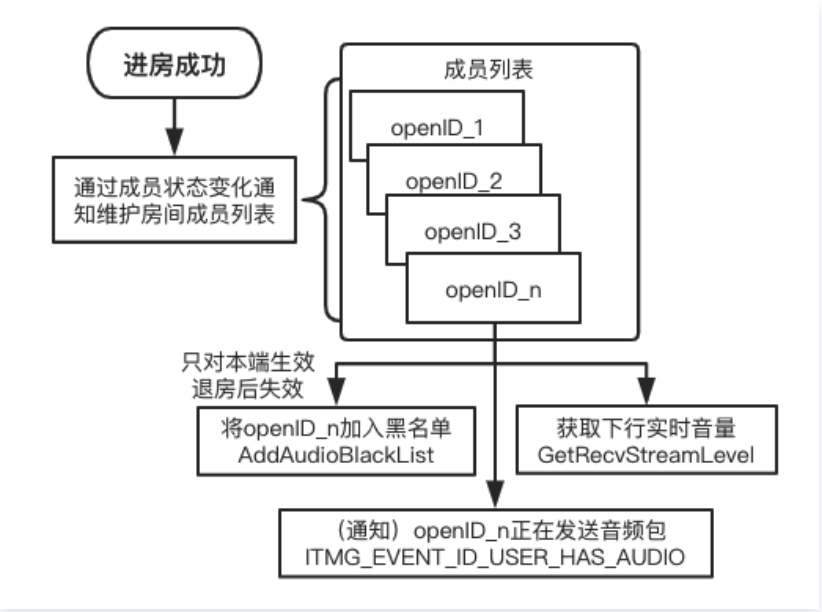
```
IsRoomEntered() :boolean
```

示例代码

```
context.IsRoomEntered();
```

房间内状态维护

此部分接口用于业务层显示说话成员、进退房成员，以及将房间内某成员禁言等功能。



接口/通知	含义
ITMG_MAIN_EVNET_TYPE_USER_UPDATE	成员状态变化通知。
AddAudioBlackList	房间中禁言某成员。
RemoveAudioBlackList	移除禁言。
IsOpenIdInAudioBlackList	查询某openid是否被禁言。

成员进房、说话状态通知事件

- 该事件适用于获取房间中说话的人并在 UI 中展示，以及有人进入、退出语音房间的一个通知。
- 该事件在状态变化才通知，状态不变化的情况下不通知。如需实时获取成员状态，请在业务层收到通知时缓存，事件消息为 ITMG_MAIN_EVNET_TYPE_USER_UPDATE，包含 event_id、count 及 openIdList，在 OnEvent 通知中对事件消息进行判断。
- 音频事件 ITMG_EVENT_ID_USER_NO_AUDIO 的通知有一个阈值，超过这个阈值才会发送通知。即本端两秒没采集到声音后， 房间其他成员才收到本端停止说话的通知。
- 音频事件只会返回成员说话状态，没有返回具体的音量。如需房间内成员具体音量可使用接口 GetVolumeById 进行获取。

event_id	含义	应用侧维护内容
ITMG_EVENT_ID_USER_ENTER	有成员进入房间，返回此时进房的 OpenID	应用侧维护成员列表
ITMG_EVENT_ID_USER_EXIT	有成员退出房间，返回此时退房的 OpenID	应用侧维护成员列表
ITMG_EVENT_ID_USER_HAS_AUDIO	有成员发送音频包，返回此时房间内说话的 OpenID，通过此事件可以判断用户是否说话，并展示声纹效果	应用侧维护通话成员列表
ITMG_EVENT_ID_USER_NO_AUDIO	有成员停止发送音频包，返回此时房间内停止说话的 OpenID	应用侧维护通话成员列表
ITMG_EVENT_ID_USER_MIC_OPENED	有成员打开了麦克风上行音频（但不一定在说话）	应用侧维护通话成员列表

版权所有：腾讯云计算（北京）有限责任公司

ITMG_EVENT_ID_USER_MIC_CLOSED	有成员关闭了麦克风上行音频	应用侧维护通话成员列表
ITMG_MAIN_EVENT_TYPE_USER_VOLUMES	回调成员的当前音量	应用侧维护通话成员列表

示例代码

```
onEvent(eventType, msg){
  if (eventType == ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVNET_TYPE_USER_UPDATE)
  {
    var eventValue = JSON.parse(msg);
    var eventID = eventValue["event_id"];
      //进行处理
    switch (eventID)
    {
    case ITMG_EVENT_ID_USER_UPDATE.ITMG_EVENT_ID_USER_ENTER:
          //有成员进入房间
          break;
    case ITMG_EVENT_ID_USER_UPDATE.ITMG_EVENT_ID_USER_EXIT:
          //有成员退出房间
        break;
    case ITMG_EVENT_ID_USER_UPDATE.ITMG_EVENT_ID_USER_HAS_AUDIO:
        //有成员发送音频包
        break;
    case ITMG_EVENT_ID_USER_UPDATE.ITMG_EVENT_ID_USER_NO_AUDIO:
        //有成员停止发送音频包
        break;
    case ITMG_EVENT_ID_USER_UPDATE.ITMG_EVENT_ID_USER_MIC_OPENED:
        //有成员打开了麦克风上行音频（但不一定在说话）
        break;
    case ITMG_EVENT_ID_USER_UPDATE.ITMG_EVENT_ID_USER_MIC_CLOSED:
        //有成员关闭了麦克风上行音频
        break;
    case ITMG_EVENT_ID_USER_UPDATE.ITMG_MAIN_EVENT_TYPE_USER_VOLUMES:
        //成员的当前音量
        break;
    default:
        break;
    }
    break;
  }
});
```

房间中禁言某成员

将某个 ID 加入音频数据黑名单，即不接受某人的语音， 只对本端生效，不会影响其他端。返回值为 0 表示调用成功。例如： A、B、C 都在同一个房间开麦说话：

- 如果 A 设置了 C 的黑名单， 则 A 只能听见 B 的声音。
- B 因为没有设置黑名单， 仍旧可以听见 A 和 C 的声音。
- C 同样因为没有设置黑名单， 可以听见 A 和 B 的声音。

此接口适用于在语音房间中将某用户禁言的场景。

接口原型

```
AddAudioBlackList(openId: string) :number
```

参数	类型	含义
openId	String	需添加黑名单的用户 OpenID

示例代码

```
context.AddAudioBlackList(openId);
```

移除禁言

将某个 Id 移除音频数据黑名单。返回值为0表示调用成功。

接口原型

```
RemoveAudioBlackList(openId: string) :number
```

参数	类型	含义
openId	String	需移除黑名单的 ID

示例代码

```
context.RemoveAudioBlackList(openId);
```

查询用户是否被禁言

查询某个 ID 是否在黑名单。返回值为true表示在黑名单，false表示不在黑名单。

接口原型

```
IsOpenIdInAudioBlackList(openId: string) :boolean
```

参数	类型	含义
openId	String	需查询是否在黑名单的 ID

示例代码

```
boolean isInBlackList = context.IsOpenIdInAudioBlackList(openId);
```

实时语音采集相关接口

- 初始化 SDK 之后进房，在房间中，才可以调用实时音频语音相关接口。
- 当用户界面单击打开/关闭麦克风/扬声器按钮时，建议采用 EnableMic 以及 EnableSpeaker 接口进行调用。
- 当用户界面按住麦克风按钮时发言，放开按钮不发言，建议采用进房时候调用 EnableAudioCaptureDevice 一次，后续按住发言调用 EnableAudioSend 来实现。

接口	接口含义

EnableMic	开关麦克风。
GetMicState	获取麦克风状态。
EnableAudioCaptureDevice	开关采集设备。
IsAudioCaptureDeviceEnabled	获取采集设备状态。
EnableAudioSend	打开关闭音频上行。
IsAudioSendEnabled	获取音频上行状态。
GetMicLevel	获取实时麦克风音量。
GetSendStreamLevel	获取音频上行实时音量。
SetMicVolume	设置麦克风音量。
GetMicVolume	获取麦克风音量。

开启或关闭麦克风

此接口用来开启关闭麦克风。加入房间默认不打开麦克风及扬声器。EnableMic = EnableAudioCaptureDevice + EnableAudioSend

接口原型

```
EnableMic(bEnable: boolean) : number
```

参数	类型	含义
isEnabled	Boolean	如果需要打开麦克风，则传入的参数为 true，如果关闭麦克风，则参数为 false。

示例代码

```
//打开麦克风
context.EnableMic(true);
```

麦克风状态获取

此接口用于获取麦克风状态，返回值0为关闭麦克风状态，返回值1为打开麦克风状态。

接口原型

```
GetMicState() :number
```

示例代码

```
context.GetMicState();
```

开启或关闭采集设备

此接口用来开启/关闭采集设备。加入房间默认不打开设备。

- 只能在进房后调用此接口，退房会自动关闭设备。
- 在移动端，打开采集设备通常会伴随权限申请，音量类型调整等操作。

接口原型

```
EnableAudioCaptureDevice(enable:boolean) :number
```

参数	类型	含义
enable	Boolean	如果需要打开采集设备，则传入的参数为 true，如果关闭采集设备，则参数为 false

示例代码

```
//打开采集设备
context.EnableAudioCaptureDevice(true);
```

采集设备状态获取

此接口用于采集设备状态获取。

接口原型

```
IsAudioCaptureDeviceEnabled():boolean
```

示例代码

```
boolean IsAudioCaptureDevice = context.IsAudioCaptureDeviceEnabled();
```

打开或关闭音频上行

此接口用于打开/关闭音频上行。如果采集设备已经打开，那么会发送采集到的音频数据。如果采集设备没有打开，那么仍旧无声。采集设备的打开关闭请参见接口 EnableAudioCaptureDevice。

接口原型

```
EnableAudioSend(bEnable: boolean) :number
```

参数	类型	含义
isEnabled	Boolean	如果需要打开音频上行，则传入的参数为 true，如果关闭音频上行，则参数为 false

示例代码

```
context.EnableAudioSend(true);
```

音频上行状态获取

此接口用于音频上行状态获取。

接口原型

```
IsAudioSendEnabled():boolean
```

示例代码

```
boolean IsAudioSend = context.IsAudioSendEnabled();
```

获取麦克风实时音量

此接口用于获取麦克风实时音量，返回值为 number 类型。建议20ms获取一次。值域为0 – 100，通过此接口可以获取到麦克风采集到的实时音量情况。

接口原型

```
GetMicLevel():number
```

示例代码

```
context.GetMicLevel();
```

获取音频上行实时音量

此接口用于获取自己音频上行实时音量，返回值为 number 类型，取值范围为0 – 100。

接口原型

```
GetSendStreamLevel() :number
```

示例代码

```
context.GetSendStreamLevel();
```

设置麦克风软件音量

此接口用于设置麦克风的音量。参数 volume 用于设置麦克风的音量，相当于对采集的声音做衰减或增益。

接口原型

```
SetMicVolume(volume:number) :number
```

参数	类型	含义
volume	Number	取值范围为 0–200，数值为0的时候表示静音，当数值为100的时候表示音量不增不减，默认数值为100。

示例代码

```
number micVol = (value * 100);
context.SetMicVolume (micVol);
```

获取麦克风软件音量

此接口用于获取麦克风的音量。返回值为一个number类型数值，返回值为101代表没调用过接口 SetMicVolume。

接口原型

```
GetMicVolume()
```

示例代码

```
context.GetMicVolume();
```

实时语音播放相关接口

接口	接口含义
EnableSpeaker	开关扬声器。
GetSpeakerState	获取扬声器状态。
EnableAudioPlayDevice	开关播放设备。
IsAudioPlayDeviceEnabled	获取播放设备状态。
EnableAudioRecv	打开关闭音频下行。
IsAudioRecvEnabled	获取音频下行状态。
GetSpeakerLevel	获取实时扬声器音量。
GetRecvStreamLevel	获取房间内其他成员下行实时音量。
SetSpeakerVolume	设置扬声器音量。
GetSpeakerVolume	获取扬声器音量。

开启或关闭扬声器

此接口用于开启关闭扬声器。EnableSpeaker = EnableAudioPlayDevice + EnableAudioRecv

接口原型

```
EnableSpeaker(bEnable: boolean) : number;
```

参数	类型	含义
bEnable	Boolean	如果需要关闭扬声器，则传入的参数为 false，如果打开扬声器，则参数为 true

示例代码

```
//打开扬声器
context.EnableSpeaker(true);
```

扬声器状态获取

此接口用于扬声器状态获取。返回值0为关闭扬声器状态，返回值1为打开扬声器状态。

接口原型

```
GetSpeakerState() :number
```

示例代码

```
context.GetSpeakerState();
```

开启或关闭播放设备

此接口用于开启关闭播放设备。

接口原型

```
EnableAudioPlayDevice(enable:boolean) :number
```

参数	类型	含义
enable	Boolean	如果需要关闭播放设备，则传入的参数为 false，如果打开播放设备，则参数为 true

示例代码

```
context.EnableAudioPlayDevice(true);
```

播放设备状态获取

此接口用于播放设备状态获取。

接口原型

```
IsAudioPlayDeviceEnabled() :boolean
```

示例代码

```
boolean enable = context.IsAudioPlayDeviceEnabled();
```

打开或关闭音频下行

此接口用于打开/关闭音频下行。如果播放设备已经打开，那么会播放房间里其他人的音频数据。如果播放设备没有打开，那么仍旧无声。播放设备的打开关闭参见接口请参见 EnableAudioPlayDevice。

接口原型

```
EnableAudioRecv(bEnable: boolean) :number
```

参数	类型	含义
isEnabled	Boolean	如果需要打开音频下行，则传入的参数为 true，如果关闭音频下行，则参数为 false

示例代码

```
context.EnableAudioRecv(true);
```

音频下行状态获取

此接口用于音频下行状态获取。

接口原型

```
IsAudioRecvEnabled():boolean
```

示例代码

```
boolean IsAudioRecv = context.IsAudioRecvEnabled();
```

获取扬声器实时音量

此接口用于获取扬声器实时音量。返回值为 number 类型数值，表示扬声器实时音量。建议20ms获取一次。

接口原型

```
GetSpeakerLevel():number
```

示例代码

```
context.GetSpeakerLevel();
```

获取房间内其他成员下行实时音量

此接口用于获取房间内其他成员下行实时音量，返回值为 number 类型，取值范围为0 - 200。

接口原型

```
GetRecvStreamLevel(openId: string) :number
```

参数	类型	含义
openId	String	房间其他成员的 OpenId

示例代码

```
number level =GetRecvStreamLevel(openId);
```

动态设置房间内某成员音量

此接口用于设置房间内某成员的说话音量大小，此设置只在本端生效。

接口原型

```
SetSpeakerVolumeByOpenID(openId: string, volume:number) :number;
```

参数	类型	含义
openId	String	需要调节音量大小的 OpenID。
volume	Number	百分比，建议[0-200]，其中100为默认值。

示例代码

```
context.SetSpeakerVolumeByOpenID(openId,  100);
```

获取设置的声音百分比

调用此接口获取 SetSpeakerVolumeByOpenID 设置的能量值

接口原型

```
GetSpeakerVolumeByOpenID(openId: string) :number;
```

参数	类型	含义
openId	String	需要调节音量大小的 OpenID。

返回值

接口返回 OpenID 设置的能量百分比， 默认返回100。

示例代码

```
context.GetSpeakerVolumeByOpenID(openId);
```

设置扬声器的音量

此接口用于设置扬声器的音量。

接口原型

```
SetSpeakerVolume(volume:number) :number
```

参数	类型	含义
volume	Number	设置音量，范围0 – 200，当数值为0时，表示静音，当数值为100时，表示音量不增不减，默认数值为100。

示例代码

```
number vol = 100;
context.SetSpeakerVolume(vol);
```

获取扬声器的音量

此接口用于获取扬声器的音量。返回值为 number 类型数值，代表扬声器的音量，返回值为101代表没调用过接口 SetSpeakerVolume。Level 是实时音量，Volume 是扬声器的音量，最终声音音量 = Level × Volume %。例如实时音量是数值是100，此时 Volume 的数值是60，那么最终发出来的声音数值也是60。

接口原型

```
GetSpeakerVolume() :number
```

示例代码

```
numbet volume = context.GetSpeakerVolume();
```

设备选择相关接口

设备选择相关接口只能在 PC 端上使用。

接口	接口含义
GetMicListCount	获取麦克风设备数量。
GetMicList	枚举麦克风设备。
GetSpeakerListCount	获取扬声器设备数量。
GetSpeakerList	枚举扬声器设备。
SelectMic	选定麦克风设备。
SelectSpeaker	选定扬声器设备。

获取麦克风设备数量

此接口用来获取麦克风设备数量。

函数原型

```
GetMicListCount() :number
```

示例代码

```
var micListCount = context.GetMicListCount();
```

枚举麦克风设备

此接口用来枚举麦克风设备。配合 GetMicListCount 接口使用。

函数原型

```
GetMicList() :GmeAudioDeviceInfo[];
```

示例代码

```
var micList = context.GetMicList();
```

选中麦克风设备

此接口用来选中麦克风设备。如果不调用或者传入 "DEVICEID_DEFAULT"，则选中系统默认设备。
GetMicList接口中返回的第0个设备 id 为默认设备，未选中设备时通话设备为默认设备，选中后业务层维护通话设备。如果该通话设备被拔出，则此时通话设备为默认设备，拔出的通话设备插入后，此时通话设备恢复为插入的通话设备。

函数原型

```
SelectMic(micId: string) :number;
```

参数	类型	含义
micId	String	麦克风设备 ID，设备 ID 来自于 GetMicList 返回列表。

示例代码

```
context.SelectMic(deviceID);
```

此接口用来获取扬声器设备数量。

函数原型

```
 GetSpeakerListCount() :number;
```

示例代码

```
context.GetSpeakerListCount();
```

枚举扬声器设备

此接口用来枚举扬声器设备。配合 GetSpeakerListCount 接口使用。

函数原型

```
GetSpeakerList(): GmeAudioDeviceInfo[]
```

示例代码

```
var speakList = GetSpeakerList();
```

选中扬声器设备

此接口用来选中播放设备。如果不调用或者传入 "DEVICEID_DEFAULT"，则选中系统默认播放设备。

函数原型

```
SelectSpeaker(speakerId: string) :number
```

参数含义

参数	类型	含义
speakerId	String	扬声器设备 ID，设备 ID 来自于 GetSpeakerList 返回列表。

示例代码

```
var ret = SelectSpeaker(deviceID);
```

房间用户音量回调事件

该接口用于获取房间用户音量值。

函数原型

```
TrackingVolume(timeS: number) : number
```

参数含义

参数	类型	含义
timeS	Number	设置用户音量回调时间间隔，单位：秒

房间用户音量回调事件

事件消息为 ITMG_MAIN_EVENT_TYPE_USER_VOLUMES，返回的json内部参数为 openId、volume，代表的信息如下：

参数	类型	含义
openId	String	用户 ID
volume	Number	当前音量

示例代码

```
context.TrackingVolume(0.5);
```

停止获取房间用户音量的回调

该接口用于停止获取房间用户音量值

接口原型

```
StopTrackingVolume() : number
```

示例代码

```
context.StopTrackingVolume();
```

获取当前选中的扬声器设备

该接口用于获取当前选中的扬声器设备

接口原型

```
GetCurrentSpeaker()  : [number, GmeAudioDeviceInfo | undefined]
```

返回值说明

返回值	说明
number	0:成功，其他:失败
GmeAudioDeviceInfo	成功时 GmeAudioDeviceInfo 有意义，具体参考 GmeAudioDeviceInfo 定义，失败时返回 undefined

GmeAudioDeviceInfo说明

```
export class GmeAudioDeviceInfo{
    deviceName:string;
    deviceId:string;
    constructor(name:string, id:string){
        this.deviceName = name;
        this.deviceId = id;
    };
}
```

示例代码

```
let [nRet, devInfo] = context.GetCurrentSpeaker();
```

获取当前选中的麦克风设备

该接口用于获取当选中的麦克风设备

接口原型

```
GetCurrentMic()  : [number, GmeAudioDeviceInfo | undefined]
```

返回值说明

返回值	说明
number	0:成功，其他:失败
GmeAudioDeviceInfo	成功时 GmeAudioDeviceInfo 有意义，具体参考 GmeAudioDeviceInfo 定义，失败时返回 undefined

示例代码

```
let [nRet, devInfo] = context.GetCurrentMic();
```

高级 API

启动耳返

此接口用于启动耳返，需要 EnableLoopBack+EnableSpeaker 才可以听到自己声音。

接口原型

```
EnableLoopBack(bEnable: boolean) :number
```

参数	类型	含义
enable	Boolean	设置是否启动

示例代码

```
context.EnableLoopBack(true);
```

获取用户房间音频类型

此接口用于获取用户房间音频类型，返回值为房间音频类型，返回值为0时代表获取用户房间音频类型发生错误，房间音频类型参考 EnterRoom 接口。

接口原型

```
GetRoomType() :number
```

示例代码

```
context.GetRoomType();
```

房间类型修改

此接口用于修改用户房间音频类型，结果参见回调事件，事件类型为 ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE。房间的音频类型由第一个进房的人确定，此后房间里有成员修改房间类型，将对此房间所有成员生效。

接口原型

```
ChangeRoomType(roomType: number) :number
```

参数	类型	含义
roomtype	Number	房间切换成的目标类型，房间音频类型参考 EnterRoom 接口

示例代码

```
context.ChangeRoomType(ITMG_ROOM_TYPE_FLUENCY);
```

回调事件

房间类型设置完成后，回调的事件消息为 ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE，返回的参数为 result、error_info 及 new_room_type，new_room_type 代表的信息如下，在 OnEvent 函数中对事件消息进行判断。

事件子类型	代表参数	含义
ITMG_ROOM_CHANGE_EVENT_ENTERROOM	1	表示在进房的过程中，自带的音频类型与房间不符合，被修改为所进入房间的音频类型
ITMG_ROOM_CHANGE_EVENT_START	2	表示已经在房间内，音频类型开始切换（例如调用 ChangeRoomType 接口后切换音频类型 ）
ITMG_ROOM_CHANGE_EVENT_COMPLETE	3	表示已经在房间，音频类型切换完成
ITMG_ROOM_CHANGE_EVENT_REQUEST	4	表示房间成员调用 ChangeRoomType 接口，请求切换房间音频类型

示例代码

```
context.setTMGDelegate(function(eventId, msg){
  if (ITMGContext.ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE == type)
```

```
        {
        //对房间类型事件进行处理
    }
});
```

房间通话质量监控事件

质量监控事件，此通知事件适用于监听网络质量，如果用户网络差的话，业务层将通过 UI 提醒用户切换网络。在进房后触发，事件2秒回调一次，事件消息为 ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_QUALITY，返回的参数为 weight、loss 及 delay，代表的信息如下：

参数	类型	含义
weight	Number	范围是1 – 50，数值为50是音质评分极好，数值为1是音质评分很差，几乎不能使用，数值为0代表初始值，无含义。一般数值在30以下就可以提醒用户网络较差，建议切换网络。
loss	Var	上行丢包率。
delay	Number	音频触达延迟时间（ms）。

获取版本号

获取 SDK 版本号，用于分析。

接口原型

```
GetSDKVersion() :string
```

示例代码

```
context.GetSDKVersion();
```

设置应用名称和版本

该接口用于设置应用名称和版本。

接口原型

```
SetAppVersion(appVersion: string) : number
```

参数含义

参数	类型	含义
appVersion	String	应用名称和版本

示例代码

```
context.SetAppVersion("gme V2.0.0");
```

设置打印日志等级

用于设置打印日志等级。建议保持默认等级。需要在 Init 之前调用。

接口原型

```
SetLogLevel(level: number) : number
```

参数含义

参数	类型	含义
level	Number	设置日志的等级，TMG_LOG_LEVEL_NONE 表示不写入，默认为 TMG_LOG_LEVEL_INFO。

level 说明如下：

level	含义
TMG_LOG_LEVEL_NONE	不打印日志。
TMG_LOG_LEVEL_ERROR	打印错误日志（默认）。
TMG_LOG_LEVEL_INFO	打印提示日志。
TMG_LOG_LEVEL_DEBUG	打印开发调试日志。
TMG_LOG_LEVEL_VERBOSE	打印高频日志。

示例代码

```
context.SetLogLevel(TMG_LOG_LEVEL_INFO);
```

设置打印日志路径

用于设置打印日志路径。默认路径如下。需要在 Init 之前调用。

平台	路径
Windows	%appdata%\GMEGLOBAL\GME\ProcessName

接口原型

```
SetLogPath(logPath: string)
```

参数	类型	含义
logPath	String	路径

示例代码

```
string logDir = ""//自行设置路径
context.SetLogPath(logDir);
```

获取诊断信息

获取音视频通话的实时通话质量的相关信息。该接口主要用来查看实时通话质量、排查问题等，业务侧可以忽略。

接口原型

```
GetQualityTips() :string
```

示例代码

```
string tips = context.GetQualityTips();
```

实时语音伴奏相关接口

开始播放伴奏

此接口用于开始播放伴奏和背景音乐，支持 m4a、wav、mp3 一共三种格式，同时开始推流

接口原型

```
StartAccompany(file: string, loopBack = true, loopCount = 1 , msTime = 0) :number
```

参数	类型	含义
file	String	背景音乐的播放文件。
loopBack	Boolean	是否混音发送，一般都设置为 true，即其他人也能听到伴奏。
loopCount	Number	循环次数，数值为-1表示无限循环。填0不播放。
msTime	Number	延迟时间（ms）。

示例代码

```
context.StartAccompany(filePath,true,-1,0);
```

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。

```
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

此接口用于停止播放伴奏，同时停止推流

接口原型

```
StopAccompany(duckerTime:number) :number
```

参数	类型	含义
duckerTime	Number	淡出时间（ms）。

示例代码

```
context.StopAccompany(0);
```

设置伴奏音量

此接口用于设置伴奏音量，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。

接口原型

```
SetAccompanyVolume(volume: number) :number
```

参数	类型	含义
volume	Number	音量数值。

示例代码

```
context.SetAccompanyVolume(100);
```

暂停播放伴奏

此接口用于暂停播放伴奏。

接口原型

```
PauseAccompany() :number
```

示例代码

```
context.PauseAccompany();
```

恢复播放伴奏

此接口用于恢复播放伴奏。

接口原型

```
ResumeAccompany() :number
```

示例代码

```
context.ResumeAccompany();
```

回调消息

消息	含义	Data	例子 |
ITMG_MAIN_EVENT_TYPE_ENTER_ROOM	进入音频房间消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_EXIT_ROOM	退出音频房间消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_ROOM_DISCONNECT	房间因为网络等原因断开消息	result; error_info	{"error_info":"waiting timeout, please check your network","result":0}
ITMG_MAIN_EVNET_TYPE_USER_UPDATE	房间成员更新消息	user_list; event_id	{"event_id":1,"user_list":["0"]}
ITMG_MAIN_EVENT_TYPE_NUMBER_OF_USERS_UPDATE	房间成员数量更新消息	AccUser;AllUser; ProxyUser	{"AccUser":2,"AllUser":2,"ProxyUser":0}
ITMG_MAIN_EVENT_TYPE_NUMBER_OF_AUDIOSTREAMS_UPDATE	房间内音频流数量更新	AudioStreams	{"AudioStreams":0}
ITMG_MAIN_EVENT_TYPE_RECONNECT_START	房间重连开始消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_RECONNECT_SUCCESS	房间重连成功消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_SWITCH_ROOM	快速切换房间消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE	房间状态切换消息	result; error_info; sub_event_type; new_room_type	{"error_info":"","new_room_type":0,"result":0}
ITMG_MAIN_EVENT_TYPE_AUDIO_DATA_EMPTY	音频数据为空消息	reason	{"reason":65536}
ITMG_MAIN_EVENT_TYPE_ROOM	开始跨房连麦消息	result;	{"result":0}

_SHARING_START			
ITMG_MAIN_EVENT_TYPE_ROOM_SHARING_STOP	跨房连麦停止消息	result;	{"result":0}
ITMG_MAIN_EVENT_TYPE_RECORD_COMPLETED	K歌录制音频文件完成消息	result;filepath;duration	{"result":0,"filepath":"","duration":0}
ITMG_MAIN_EVENT_TYPE_RECORD_PREVIEW_COMPLETED	K歌预览录制音频文件完成消息	result	{"result":0}
ITMG_MAIN_EVENT_TYPE_RECORD_MIX_COMPLETED	合成录制文件完成消息	result;filepath;mic_filepath;duration	{"result":0,"filepath":"","mic_filepath":"", "duration" : 0}
ITMG_MAIN_EVENT_TYPE_IOS_MUTE_SWITCH_RESULT	检测iOS静音开关消息	muteSwitchIsOn	{"muteSwitchIsOn":true}
ITMG_MAIN_EVENT_TYPE_SPEAKER_DEFAULT_DEVICE_CHANGED	默认扬声器设备修改消息	result; error_info	{"deviceID":"{0.0.0.00000000}.{a4f1e8be-49fa-43e2-b8cf-dd00542b47ae}","deviceName":"扬声器 (Realtek High Definition Audio)","error_info":"","isNewDevice":true,"isUsedDevice":false,"result":0}
ITMG_MAIN_EVENT_TYPE_SPEAKER_NEW_DEVICE	新增扬声器设备消息	result; error_info	{"deviceID":"{0.0.0.00000000}.{a4f1e8be-49fa-43e2-b8cf-dd00542b47ae}","deviceName":"扬声器 (Realtek High Definition Audio)","error_info":"","isNewDevice":true,"isUsedDevice":false,"result":0}
ITMG_MAIN_EVENT_TYPE_SPEAKER_LOST_DEVICE	丢失扬声器设备消息	result; error_info	{"deviceID":"{0.0.0.00000000}.{a4f1e8be-49fa-43e2-b8cf-dd00542b47ae}","deviceName":"扬声器 (Realtek High Definition Audio)","error_info":"","isNewDevice":false,"isUsedDevice":false,"result":0}
ITMG_MAIN_EVENT_TYPE_MIC_NEW_DEVICE	新增麦克风设备消息	result; error_info	{"deviceID":"{0.0.1.00000000}.{5fdf1a5b-f42d-4ab2-890a-7e454093f229}","deviceName":"麦克风 (Realtek High Definition Audio)","error_info":"","isNewDevice":true,"isUsedDevice":true,"result":0}
ITMG_MAIN_EVENT_TYPE_MIC_LOST_DEVICE	丢失麦克风设备消息	result; error_info	{"deviceID":"{0.0.1.00000000}.{5fdf1a5b-f42d-4ab2-890a-7e454093f229}","deviceName":"麦克风 (Realtek High Definition Audio)","error_info":"","isNewDevice":false,"isUsedDevice":true,"result":0}
ITMG_MAIN_EVENT_TYPE_MIC_DEFAULT_DEVICE_CHANGED	默认麦克风设备修改消息	result; error_info	{"deviceID":"{0.0.1.00000000}.{5fdf1a5b-f42d-4ab2-890a-7e454093f229}","deviceName":"麦克风 (Realtek High Definition

			Audio)","error_info":"","isNewDevice":false,"isUsedDevice":true,"result":0}
ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_QUALITY	房间质量消息	weight; loss; delay	{"weight":5,"loss":0.1,"delay":1}
ITMG_MAIN_EVENT_TYPE_AUDIO_ROUTE_CHANGED	音频路由改变消息	route_type	{"route_type":1}
ITMG_MAIN_EVENT_TYPE_ACCOMPANY_FINISH	伴奏播放完成消息	file_path;is_finished;result	{"file_path":"","is_finished":false,"result":0}
ITMG_MAIN_EVENT_TYPE_SERVER_AUDIO_ROUTE_EVENT	设置服务音频路由转发消息	ErrorCode;error_info	{"ErrorCode" : 0,"error_info" : ""}
ITMG_MAIN_EVENT_TYPE_CUSTOMDATA_UPDATE	收到发送的自定义音频文本 消息	sub_type;content;senderid	{"sub_type: 0,"content":"","senderid":""}
ITMG_MAIN_EVNET_TYPE_PTT_RECORD_COMPLETE	语音消息录制完成消息	result; file_path	{"file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_UPLOAD_COMPLETE	语音消息上传完成消息	result; file_path;file_id	{"file_id":"","file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_DOWNLOAD_COMPLETE	语音消息下载完成消息	result; file_path;file_id	{"file_id":"","file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_PLAY_COMPLETE	语音消息播放完成消息	result; file_path	{"file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_SPEECH2TEXT_COMPLETE	语音消息极速转文本完成消息	result; text;file_id	{"file_id":"","text":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_COMPLETE	语音消息流式转文本完成消息	result; file_path; text;file_id	{{"file_id":"","file_path":","text":"","result":0}}
ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECO	语音消息正在流式转文本中	result; file_path; text;file_id	{{"file_id":"","file_path":","text":"","result":0}}

GNITION_IS_RUNNING			
ITMG_MAIN_EVNET_TYPE_PTT_TEXT2SPEECH_COMPLETE	文本转语音完成消息	result; text;file_id	{{"file_id":"","text":"","result":0}}
ITMG_MAIN_EVNET_TYPE_PTT_TRANSLATE_TEXT_COMPLETE	文本翻译完成消息	result; text;file_id	{{"file_id":"","text":"","result":0}}
ITMG_MAIN_EVENT_TYPE_CHORUS_EVENT	收到合唱请求消息	error_info; result,Accompanier_Openid;sub_type	{ "error_info" : "","result" : 0, “Accompanier_Openid” : "","sub_type" : 2}
ITMG_MAIN_EVENT_TYPE_CHANGETEAMID	更改TEAMID消息	error_info;result; sub_type;teamid	{"error_info" : "","result" : 0,"sub_type" : 452,"teamid" : 321}
ITMG_MAIN_EVENT_TYPE_AGE_DETECTED	年龄识别结果消息	result; age_detected_result	{ "age_detected_result" : 0, "result" : 0}
ITMG_MAIN_EVENT_TYPE_HARDWARE_TEST_RECORD_FINISH	测试延迟录制结束消息	result; file_path;is_finished	{"file_path" : “” , "result" : 0， “is_finished” : true}
ITMG_MAIN_EVENT_TYPE_HARDWARE_TEST_PREVIEW_FINISH	测试延迟预览结束消息	result; file_path;is_finished	{"file_path" : “” , "result" : 0， “is_finished” : true}
ITMG_MAIN_EVNET_TYPE_ROOM_MANAGEMENT_OPERATOR	房间管理操作方法消息	SenderID; ReceiverID;OperateType;Result;OperateValue	{"SenderID":"","ReceiverID":"","OperateType":0, "Result": 0,"OperateValue":0}
ITMG_MAIN_EVNET_TYPE_VOICE_CHANGER_FETCH_COMPLETE	获取变声模板完成消息	result; voice_list	{"result":740000,"voice_list":null}
ITMG_MAIN_EVENT_TYPE_USER_VOLUMES	获取用户音量消息	result; voice_list	{"open_id":"","volume":10}

语音消息及转文本

最近更新时间：2024-12-17 17:39:22

为方便 Electron 开发者调试和接入腾讯云游戏多媒体引擎客户端 API，本文为您介绍适用于 Electron 语音消息服务及转文本服务的接入技术文档。

使用 GME 重要事项

GME 提供实时语音服务、语音消息服务及转文本服务，使用 GME 服务都依赖 Init 和 Poll 等核心接口。

重点提示

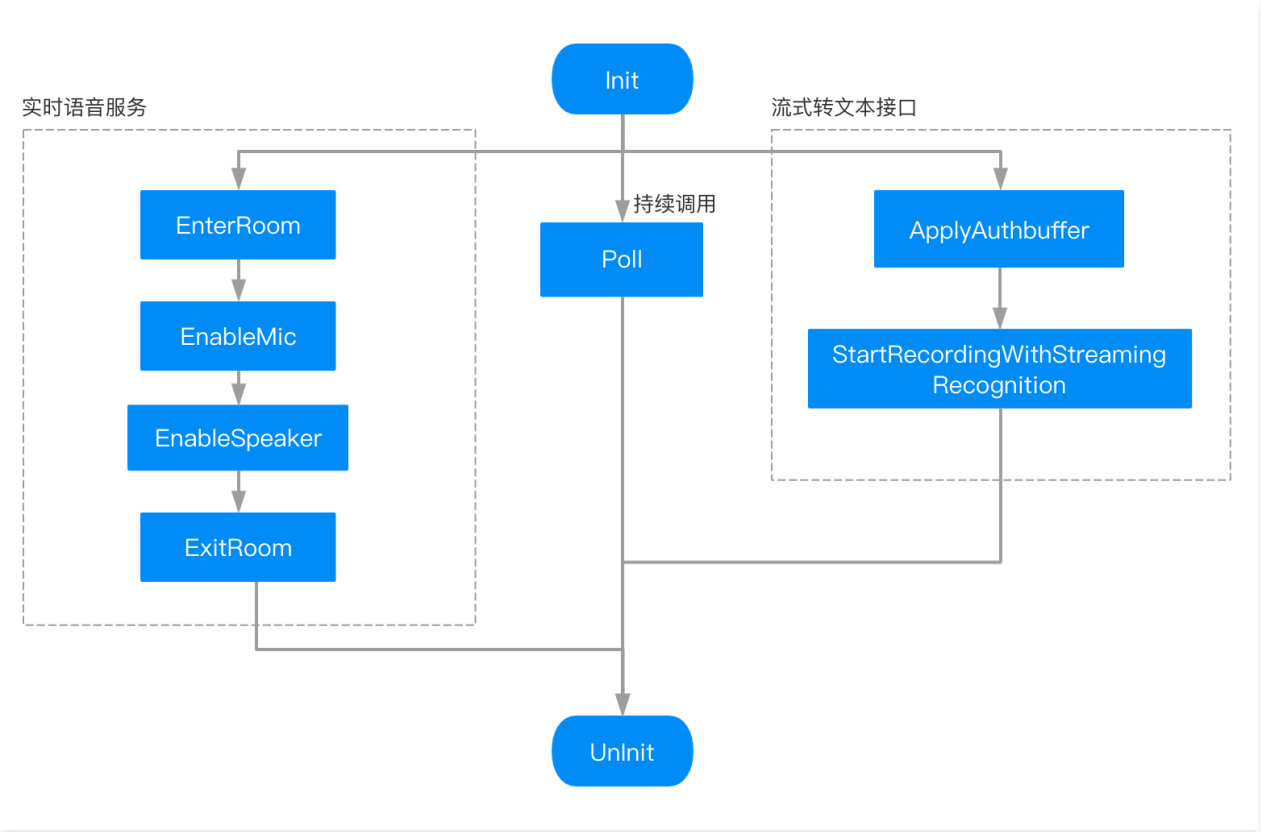
- 已完成 GME 应用创建，并获取 SDK AppID 和 Key。请参见 服务开通指引 。
- 已开通 **GME 实时语音服务、语音消息服务以及转文本服务**。请参见 服务开通指引 。
- GME 使用前请对工程进行配置，否则 SDK 不生效。
- GME 的接口调用成功后返回值为 GmeError.AV_OK，数值为 0。
- GME 的接口调用要在同一个线程下。
- GME 需要周期性的调用 Poll 接口触发事件回调。
- 错误码详情可参见 错误码 。

> ⚠ **注意**
> 语音转文本相关接口有默认频率限制，限额范围内计费方式请参见 计费文档；若需提升接口频率限额或了解超额计费方式，请联系商务或 提交工单咨询 。
> - 语音消息非流式转文本接口 **PttSpeechToText()**：默认单账号限制并发数为10路
> - 语音消息流式转文本接口 **PttStartRecordingWithStreamingRecognition()**：默认单账号限制并发数为50路

接入 SDK

重要步骤

接入 SDK 重要流程如下：



1. 初始化 GME
2. 周期性调用 Poll 触发回调
3. 鉴权初始化
4. 启动流式语音识别
5. 停止录制
6. 反初始化 GME

ts 类

```
GmeContext Gme业务实现接口
GmeError Gme 错误码定义类
```

核心接口

接口	接口含义
Init	初始化 GME
Poll	触发事件回调
Uninit	反初始化 GME

导入Gme模块

```
const { GmeContext } = require('gme-electron-sdk');
```

获取实例

var m_context = new GmeContext();

初始化 SDK

未初始化前，SDK 处于未初始化阶段，**需要通过接口 Init 初始化 SDK**，才可以使用实时语音服务、语音消息服务及转文本服务。调用 Init 接口的线程必须于其他接口在同一线程,建议都在主线程调用接口。

接口原型

```
Init(appid: string, openid: string): number;
```

参数	类型	含义
sdkAppId	string	来自 腾讯云控制台 的 GME 服务提供的 AppID，获取请参见 语音服务开通指引 。
openID	string	openID 只支持 Int64 类型（转为 string 传入），规则由 App 开发者自行制定，App 内不重复即可。如需使用字符串作为 Openid 传入，可 提交工单 联系开发者。

返回值

返回值	处理
GmeError.AV_OK= 0	初始化 SDK 成功
AV_ERR_SDK_NOT_FULL_UPDATE=7015	检查 SDK 文件是否完整，建议删除后重新导入 SDK

> ⚠ **关于7015错误提示**
> - 7015错误码是通过 md5 进行判断，在接入过程中若出现此错误，请根据提示检查 SDK 文件是否完整、SDK 文件版本是否一致。
> - 出现返回值 AV_ERR_SDK_NOT_FULL_UPDATE 时，此返回值**只有提示作用**，并不会造成初始化失败。

示例代码

```
number ret = m_context.Init(sdkAppId, openID);
//通过返回值判断是否初始化成功
if (ret != GmeError.AV_OK)
{
     console.log("SDK初始化失败:");
        return;
}
```

触发事件回调

通过在定时器调用 Poll 可以触发事件回调。Poll 是 GME 的消息泵，GME 需要周期性的调用 Poll 接口触发事件回调。如果没有调用 Poll ，将会导致整个 SDK 服务运行异常。详情请参见 Sample Project 中的 EnginePollHelper 文件。

> ⚠ **注意**
> 务必周期性调用 Poll 接口且在主线程调用，以免接口回调异常。

接口原型

```
Poll();
```

示例代码

```
setInterval(function () {
      m_context.Poll();
    }, 50);
```

反初始化 SDK

反初始化 SDK，进入未初始化状态。**如果游戏业务侧账号与 openid 是绑定的，那切换游戏账号需要反初始化 GME，再用新的 openid 初始化。**

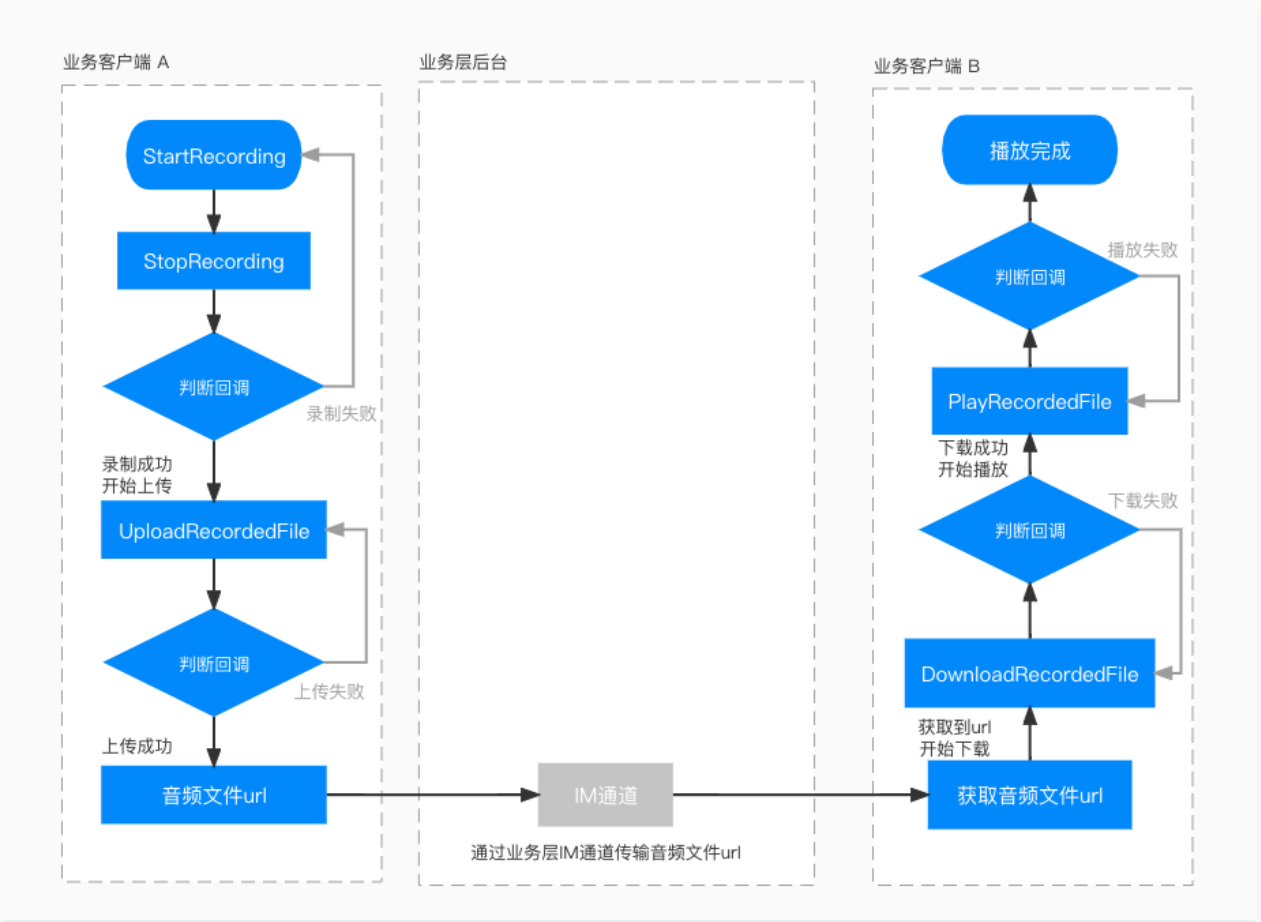
接口原型

```
Uninit() : number
```

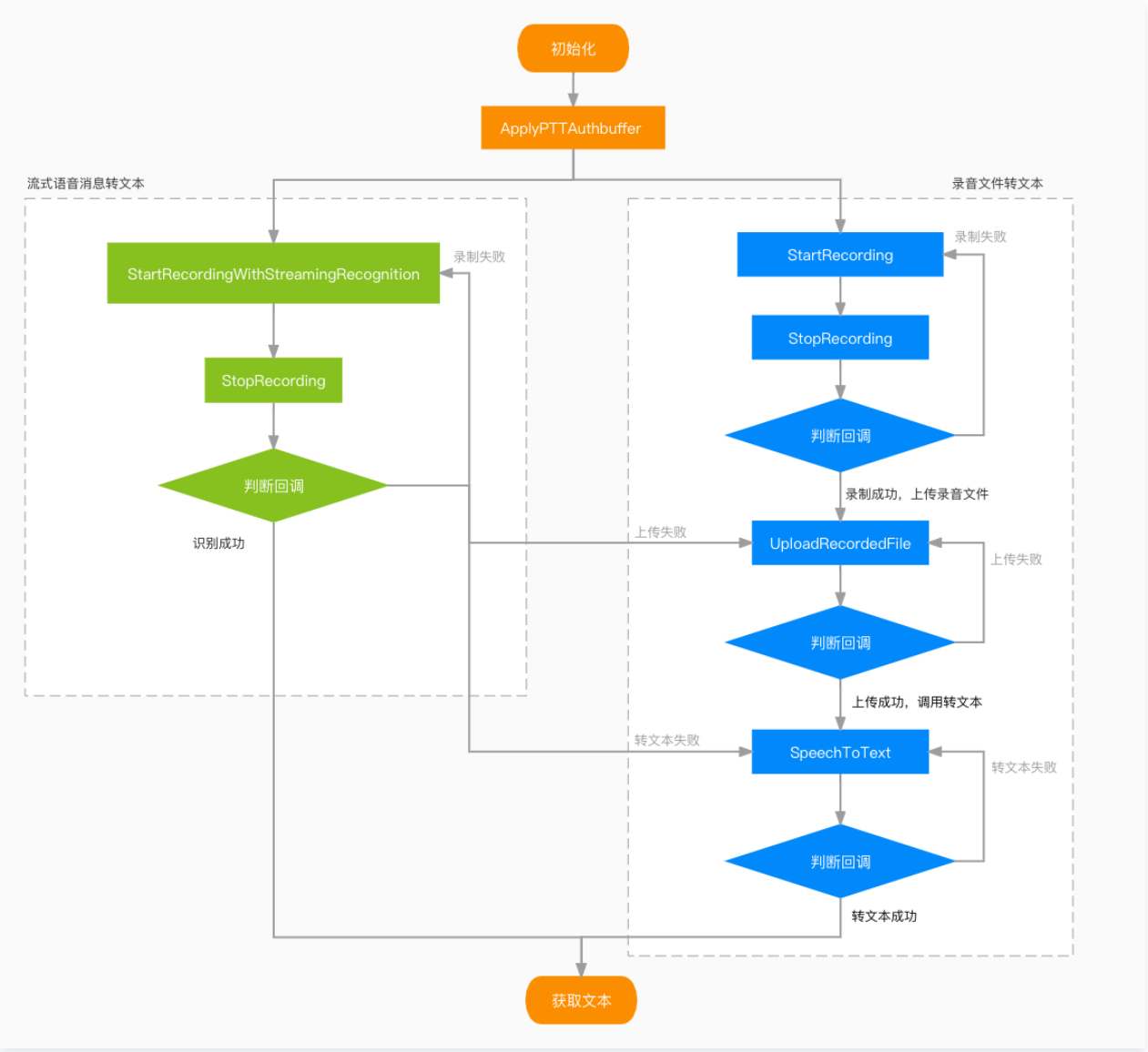
语音消息服务及转文本服务

> **说明**
> - 转文本服务分录音文件极速转文本以及语音消息流式转文本。
> - 使用语音消息服务不需要进入实时语音房间。
> - 语音消息最大录制时长默认为58秒，最短不能小于1秒。如果需要再加以限制，例如限制为最大录制时长为10秒，请在初始化之后调用 SetMaxMessageLength 接口进行设置。

语音消息服务使用流程


业务客户端 A
业务层后台
业务客户端 B
StartRecording
StopRecording
判断回调
录制失败
录制成功
开始上传
UploadRecordedFile
判断回调
上传失败
上传成功
音频文件url
IM通道
通过业务层IM通道传输音频文件url
播放完成
判断回调
播放失败
PlayRecordedFile
下载成功
开始播放
判断回调
下载失败
DownloadRecordedFile
获取到url
开始下载
获取音频文件url

转文本服务使用流程



接口	接口含义
GenAuthBuffer	获取鉴权信息
SetMaxMessageLength	限制最大语音信息时长

生成本地鉴权

生成 AuthBuffer，用于相关功能的加密和鉴权，如正式发布请使用后台部署密钥，后台部署请参见 鉴权密钥 。

接口原型

```
GenAuthBuffer(appId: string,roomId: string, openId:string, appKey: string) :string
```

参数	类型	含义
appId	string	来自腾讯云控制台的 AppId 号码。
roomId	string	填 null 或者空字符串。
openId	string	用户标识。与 Init 时候的 OpenId 相同。

key	string	来自腾讯云 控制台 的权限密钥。

应用鉴权

生成鉴权信息后，将鉴权赋值到 SDK 中。

接口原型

```
ApplyPTTAuthbuffer(authBuffer: string) :number
```

参数	类型	含义
authBuffer	string	鉴权

示例代码

```
var authBuffer = m_context.GetAuthBuffer(UserConfig.GetAppID(), UserConfig.GetUserID(),
null,UserConfig.GetAuthKey());
m_context.ApplyPTTAuthbuffer(authBuffer);
```

限制最大语音信息时长

限制最大语音消息的长度，最大支持58秒。

接口原型

```
PttSetMaxMessageLength(msTime: number) :number
```

参数	类型	含义
msTime	number	语音时长，单位 ms，区间为 1000 < msTime < = 58000

示例代码

```
m_context.PttSetMaxMessageLength(58000);
```

流式语音识别

语音消息及转文字相关接口

接口	接口含义
PttStartRecordingWithStreamingRecognition	启动流式录音
PttStopRecording	停止录音

启动流式语音识别

此接口用于启动流式语音识别，同时在回调中会有实时的语音转文字返回，可以指定语言进行识别，也可以将语音中识别到的信息翻译成指定的语言返回。**停止录音调用 停止录制 。**

接口原型

```
PttStartRecordingWithStreamingRecognition(filePath: string, speechLanguage: string,
translateLanguage: string) :number
```

参数	类型	含义
filePath	string	存放的语音路径
speechLanguage	string	识别成指定文字的语言参数，参数请参见 语言参数参考列表
translateLanguage	string	翻译成指定文字的语言参数，参数请参见 语言参数参考列表

示例代码

```
string filePath = "xx/xxx/xxx.silk"
var ret = m_context.StartRecordingWithStreamingRecognition(filePath,"cmn-Hans-CN","cmn-Hans-
CN");
if (ret == 0) {
    this.currentStatus = "开始流式录音";
} else {
    this.currentStatus = "开始流式录音失败";
}
```

> **注意**
> 翻译会收取额外费用，请参见 购买指南 。

流式语音识别的回调

启动流式语音识别后，需要在 OnEvent 通知中监听回调消息，事件消息分为以下两个：
ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_COMPLETE 是在停止录制并完成识别后才返回文字，相当于一段话说完才会返回识别的文字。
ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_IS_RUNNING 是在录音过程中就会实时返回识别到的文字，相当于边说话边返回识别到的文字。
根据需求在 回调 通知中对相应事件消息进行判断。传递的参数包含以下四个信息。

消息名称	含义
result	用于判断流式语音识别是否成功的返回码
text	语音转文字识别的文本
file_path	录音存放的本地地址
file_id	录音在后台的 url 地址，录音在服务器存放90天

> **注意**
> 监听 `ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_IS_RUNNING` 消息时，file_id 为空。

错误码

错误码	含义	处理方式
32775	流式语音转文本失败，但是录音成功	调用 UploadRecordedFile 接口上传录音，再调用 SpeechToText 接口进行语音转文字操作

32777	流式语音转文本失败，但是录音成功，上传成功	返回的信息中有上传成功的后台 url 地址，调用 SpeechToText 接口进行语音转文字操作
32786	流式语音转文本失败	在流式录制状态当中，请等待流式录制接口执行结果返回
32787	转文本成功，文本翻译服务未开通	需要在控制台开通文本翻译服务
32788	转文本成功，文本翻译语言参数不支持	重新检查传入参数

如果出现 4098 错误码，请参见 常见问题文档 进行解决。

示例代码

```
m_context.setTMGDelegate(function(eventId, msg){
   switch (eventType) {
                    case ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_COMPLETE:
                    {
                        HandleSTREAM2TEXTComplete(data,true);
                        break;
                        }
                    ...
                            case ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_IS_RUNNING:
                    {
                        HandleSTREAM2TEXTComplete(data, false);
                        break;
                    }
                }
});
```

语音消息录制

录制的流程为：录音 > 停止录音 > 录音回调返回 > 启动下一次录音。

语音消息及转文字相关接口

接口	接口含义
PttStartRecording	启动录音
PttPauseRecording	暂停录音
PttResumeRecording	恢复录音
PttStopRecording	停止录音
PttCancelRecording	取消录音

启动录音

此接口用于启动录音。

接口原型

```
PttStartRecording(filePath: string) : number;
```

参数	类型	含义
filePath	string	存放的语音路径

示例代码

```
string filepath = "xxxx/xxx.silk";
var ret = m_context.PttStartRecording(filepath);
```

停止录音

此接口用于停止录音。此接口为异步接口，停止录音后会有录音完成回调，成功之后录音文件才可用。

接口原型

```
PttStopRecording() :number;
```

示例代码

```
m_context.PttStopRecording();
```

启动录音的回调

录音完成的回调，通过委托传递消息。
停止录音调用StopRecording。停止录音后才有启动录音的回调。

参数	类型	含义
code	string	当 code 为 0 时，录制完成
filepath	string	录制的存放地址，必须是可以访问到的路径，不可将 fileid 作为路径

错误码

错误码值	原因	建议方案
4097	参数为空	检查代码中接口参数是否正确
4098	初始化错误	检查设备是否被占用，或者权限是否正常，是否初始化正常
4099	正在录制中	确保在正确的时机使用 SDK 录制功能
4100	没有采集到音频数据	检查麦克风设备是否正常
4101	录音时，录制文件访问错误	确保文件存在，文件路径的合法性
4102	麦克风未授权错误	使用 SDK 需要麦克风权限，添加权限请参考对应引擎或平台的 SDK 工程配置文档
4103	录音时间太短错误	首先，限制录音时长的单位为毫秒，检查参数是否正确；其次，录音时长要1000毫秒以上才能成

版权所有：腾讯云计算（北京）有限责任公司

		功录制
4104	没有启动录音操作	检查是否已经调用启动录音接口

示例代码

```
m_context.setTMGDelegate(function(eventId, msg){
   switch (eventType) {
                    case ITMG_MAIN_EVENT_TYPE_ENTER_ROOM:
                    {
                    //进行处理
                    break;
                        }
                    ...
                            case ITMG_MAIN_EVNET_TYPE_PTT_RECORD_COMPLETE:
                    {
                    //进行处理
                    break;
                    }
                }
});
```

暂停录音

此接口用于暂停录音。如需恢复录音请调用接口 PttResumeRecording。

接口原型

```
PttPauseRecording() : number
```

示例代码

```
number ret = m_context.PttPauseRecording();
```

恢复录音

此接口用于恢复录音。

接口原型

```
PttResumeRecording() : number;
```

示例代码

```
number ret = m_context.PttResumeRecording();
```

取消录音

调用此接口取消录音。**取消之后没有回调**。

接口原型

```
PttCancelRecording() : number
```

示例代码

```
m_context.PttCancelRecording();
```

语音消息上传、下载及播放

接口	接口含义
PttUploadRecordedFile	上传语音文件
PttDownloadRecordedFile	下载语音文件
PttPlayRecordedFile	播放语音
PttStopPlayFile	停止播放语音
PttGetFileSize	语音文件的大小
PttGetVoiceFileDuration	语音文件的时长

上传语音文件

此接口用于上传语音文件。

接口原型

```
PttUploadRecordedFile(filePath: string) : number
```

参数	类型	含义
filePath	String	上传的语音路径，此路径为本地路径

示例代码

```
var ret = m_context.PttUploadRecordedFile(filePath);
```

上传语音完成的回调

上传语音完成后，事件消息为 ITMG_MAIN_EVNET_TYPE_PTT_UPLOAD_COMPLETE，在 OnEvent 函数中对事件消息进行判断。传递的参数包含三个信息，result，file_path 和 file_id。

参数	类型	含义
result	number	当 code 为0时，录制完成
filepath	string	录制的存放地址
fileid	string	文件的 url 路径

错误码

错误码值	原因	建议方案
8193	上传文件时，文件访问错误	确保文件存在，文件路径的合法性
8194	签名校验失败错误	检查鉴权密钥是否正确，检查是否有初始化离线语音

8195	网络错误	检查设备网络是否可以正常访问外网环境
8196	获取上传参数过程中网络失败	检查鉴权是否正确，检查设备网络是否可以正常访问外网环境
8197	获取上传参数过程中回包数据为空	检查鉴权是否正确，检查设备网络是否可以正常访问外网环境
8198	获取上传参数过程中回包解包失败	检查鉴权是否正确，检查设备网络是否可以正常访问外网环境
8200	没有设置 appinfo	检查 apply 接口是否有调用，或者入参是否为空

示例代码

```
m_context.setTMGDelegate(function(eventId, msg){
 switch (eventType) {
            case ITMG_MAIN_EVENT_TYPE_ENTER_ROOM:
            {
            //进行处理
            break;
                }
            ...
                    case ITMG_MAIN_EVNET_TYPE_PTT_UPLOAD_COMPLETE:
            {
            //进行处理
            break;
            }
        }
});
```

下载语音文件

此接口用于下载语音文件。

接口原型

```
PttDownloadRecordedFile(fileId: string, filePath: string) : number
```

参数	类型	含义
fileId	string	文件的 url 路径
filePath	string	文件的本地保存路径，必须是可以访问到的路径，不可将 fileid 作为路径

示例代码

```
var ret = m_context.PttDownloadRecordedFile(fileID,filePath);
```

下载语音文件完成回调

下载语音完成后，事件消息为 ITMG_MAIN_EVNET_TYPE_PTT_DOWNLOAD_COMPLETE， 在 OnEvent 函数中对事件消息进行判断。
传递的参数包含三个信息，result、file_path 和 file_id。

参数	类型	含义
result	number	当 code 为0时，下载完成

filepath	string	录制的存放地址
fileid	string	文件的 url 路径，录音在服务器存放 90 天

错误码

错误码值	原因	建议方案
12289	下载文件时，文件访问错误	检查文件路径是否合法
12290	签名校验失败	检查鉴权密钥是否正确，检查是否有初始化离线语音
12291	网络存储系统异常	服务器获取语音文件失败，检查接口参数 fileid 是否正确，检查网络是否正常，检查 COS 文件存不存在
12292	服务器文件系统错误	检查设备网络是否可以正常访问外网环境，检查服务器上是否有此文件
12293	获取下载参数过程中，HTTP 网络失败	检查设备网络是否可以正常访问外网环境
12294	获取下载参数过程中，回包数据为空	检查设备网络是否可以正常访问外网环境
12295	获取下载参数过程中，回包解包失败	检查设备网络是否可以正常访问外网环境
12297	没有设置 appinfo	检查鉴权密钥是否正确，检查是否有初始化离线语音

示例代码

```
m_context.setTMGDelegate(function(eventId, msg){
 switch (eventType) {
           case ITMG_MAIN_EVENT_TYPE_ENTER_ROOM:
            {
            //进行处理
            break;
             }
        }
});
```

播放语音

此接口用于播放语音。

接口原型

```
PttPlayRecordedFile(filePath: string, voiceType: ITMG_VOICE_TYPE) : number
```

参数	类型	含义
filePath	string	本地语音文件的路径
voicetype	ITMG_VOICE_TYPE	变声类型，请参见 变声接入文档

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

示例代码

```
m_context.PlayRecordedFile(filePath);
```

播放语音的回调

播放语音的回调，事件消息为 ITMG_MAIN_EVNET_TYPE_PTT_PLAY_COMPLETE， 在 OnEvent 函数中对事件消息进行判断。传递的参数包含两个信息，一个是 result，另一个是 file_path。

参数	类型	含义
code	number	当 code 为0时，播放完成
filepath	string	录制的存放地址

错误码

错误码值	原因	建议方案
20481	初始化错误	检查设备是否被占用，或者权限是否正常，是否初始化正常
20482	正在播放中，试图打断并播放下一个失败了（正常是可以打断的）	检查代码逻辑是否正确
20483	参数为空	检查代码中接口参数是否正确
20484	内部错误	初始化播放器错误，解码失败等问题产生此错误码，需要结合日志定位问题

示例代码

```
m_context.setTMGDelegate(function(eventId, msg){
 switch (eventType) {
        case ITMG_MAIN_EVNET_TYPE_PTT_PLAY_COMPLETE:
        {
        //进行处理
        break;
         }
        }
});
```

停止播放语音

此接口用于停止播放语音。停止播放语音也会有播放完成的回调。

接口原型

```
PttStopPlayFile() : number
```

示例代码

```
m_context.PttStopPlayFile();
```

获取语音文件的大小

通过此接口，获取语音文件的大小。

接口原型

```
PttGetFileSize(filePath: string) : number
```

参数	类型	含义
filePath	string	语音文件的路径，此路径为本地路径

示例代码

```
m_context.PttGetFileSize(filePath);
```

获取语音文件的时长

此接口用于获取语音文件的时长，单位毫秒。

接口原型

```
PttGetVoiceFileDuration(filePath: string) : number
```

参数	类型	含义
filePath	string	语音文件的路径，此路径为本地路径

示例代码

```
number fileDuration = m_context.PttGetVoiceFileDuration(filePath);
```

录音文件极速转文

将指定的语音文件翻译成文字（指定语言）

此接口可以指定语言进行识别，也可以将语音中识别到的信息翻译成指定的语言返回。

> **注意**
> 翻译会收取额外费用，请参见 购买指南 。

接口原型

```
PttSpeechToText(fileID: string, speechLanguage: string, translateLanguage: string) : number
```

参数	类型	含义
fileID	string	语音文件 url，录音在服务器存放90天

speechLanguage	string	识别出指定文字的语言参数，参数参见 语言参数参考列表
translatelanguage	string	翻译成指定文字的语言参数，参数参见 语言参数参考列表 中的翻译语言参数

示例代码

```
m_context.PttSpeechToText(filePath,"cmn-Hans-CN","cmn-Hans-CN");
```

识别回调

将指定的语音文件识别成文字的回调，事件消息为 ITMG_MAIN_EVNET_TYPE_PTT_SPEECH2TEXT_COMPLETE，在 OnEvent 函数中对事件消息进行判断。
传递的参数包含三个信息，result、file_path 和 text，其中 text 为识别的文本。

参数	类型	含义
result	number	当 code 为0时，录制完成
fileid	string	语音文件 url，录音在服务器存放90天
text	string	转换的文本结果

错误码

错误码值	原因	建议方案
32769	内部错误	分析日志，获取后台返回给客户端的真正错误码，并联系后台同事协助解决
32770	网络失败	检查设备网络是否可以正常访问外网环境
32772	回包解包失败	分析日志，获取后台返回给客户端的真正错误码，并联系后台同事协助解决
32774	没有设置 appinfo	检查鉴权密钥是否正确，检查是否有初始化离线语音
32776	authbuffer 校验失败	检查 authbuffer 是否正确
32784	语音转文本参数错误	检查代码中接口参数 fileid 是否为空
32785	语音转文本翻译返回错误	离线语音后台错误，请分析日志，获取后台返回给客户端的真正错误码，并联系后台同事协助解决
32787	转文本成功，文本翻译服务未开通	需要在控制台开通文本翻译服务
32788	转文本成功，文本翻译语言参数不支持	重新检查传入参数

示例代码

```
m_context.setTMGDelegate(function(eventId, msg){
 switch (eventType) {
         case ITMG_MAIN_EVENT_TYPE_ENTER_ROOM:
            {
            //进行处理
            break;
                }
            ...
                    case ITMG_MAIN_EVNET_TYPE_PTT_SPEECH2TEXT_COMPLETE:
```

```
            {
            //进行处理
            break;
            }
});
```

语音消息音量相关接口

接口	接口含义
PttGetMicLevel	获取实时麦克风音量
PttSetMicVolume	设置录制音量
PttGetMicVolume	获取录制音量
PttGetSpeakerLevel	获取实时扬声器音量
PttSetSpeakerVolume	设置播放音量
PttGetSpeakerVolume	获取播放音量

获取语音消息麦克风实时音量

此接口用于获取麦克风实时音量，返回值为 number 类型，值域为0 - 200。

接口原型

```
PttGetMicLevel():number
```

示例代码

```
m_context.PttGetMicLevel();
```

设置语音消息录制音量

此接口用于设置离线语音录制音量，值域为0 - 200。

接口原型

```
PttSetMicVolume(vol:number) :number
```

参数	类型	含义
vol	number	取值范围为 0-200，数值为0的时候表示静音，当数值为100的时候表示音量不增不减，默认数值为100。

示例代码

```
m_context.PttSetMicVolume(vol);
```

获取语音消息录制音量

此接口用于获取离线语音录制音量。返回值为 number 类型，值域为0 - 200。

接口原型

```
PttGetMicVolume() : number
```

示例代码

```
m_context.PttGetMicVolume();
```

获取语音消息扬声器实时音量

此接口用于获取扬声器实时音量。返回值为 number 类型，值域为0 - 200。

接口原型

```
PttGetSpeakerLevel() : number;
```

示例代码

```
m_context.PttGetSpeakerLevel();
```

设置语音消息播放音量

此接口用于设置离线语音播放音量，值域为0 - 200。

接口原型

```
PttSetSpeakerVolume(vol: number) : number
```

示例代码

```
m_context.PttSetSpeakerVolume(100);
```

获取语音消息播放音量

此接口用于获取离线语音播放音量。返回值为 number 类型，值域为0 - 200。

接口原型

```
PttGetSpeakerVolume() : number
```

示例代码

```
m_context.PttGetSpeakerVolume();
```

高级 API

获取版本号

获取 SDK 版本号，用于分析。

接口原型

```
GetSDKVersion() :string
```

示例代码

```
string sdkVersion = m_context.GetSDKVersion();
```

设置打印日志等级

用于设置打印日志等级。建议保持默认等级。需要在 Init 之前调用。

接口原型

```
SetLogLevel(level: number) : number
```

参数含义

参数	类型	含义
level	ITMG_LOG_LEVEL	设置写入日志的等级，TMG_LOG_LEVEL_NONE 表示不写入，默认为 TMG_LOG_LEVEL_INFO

level 说明如下：

level	含义
TMG_LOG_LEVEL_NONE	不打印日志
TMG_LOG_LEVEL_ERROR	打印错误日志（默认）
TMG_LOG_LEVEL_INFO	打印提示日志
TMG_LOG_LEVEL_DEBUG	打印开发调试日志
TMG_LOG_LEVEL_VERBOSE	打印高频日志

示例代码

```
m_context.SetLogLevel(TMG_LOG_LEVEL_INFO);
```

设置打印日志路径

用于设置打印日志路径。默认路径如下。需要在 Init 之前调用。

平台	路径
Windows	%appdata%\GMEGLOBAL\GME\ProcessName

接口原型

```
SetLogPath(logPath: string)
```

参数	类型	含义
logPath	string	路径

示例代码

```
string logDir = ""//自行设置路径
m_context.SetLogPath(logDir);
```

版权所有：腾讯云计算（北京）有限责任公司

Flutter SDK 集成 SDK

最近更新时间：2023-05-04 15:12:08

为方便 Flutter 开发者调试和接入腾讯云游戏多媒体引擎产品 API，本文档主要为您介绍适用于 Flutter 开发的工程配置指引。

支持的平台

GME Flutter SDK 支持 iOS、Android 平台。

引入SDK

步骤1：下载 GME Flutter SDK

在 下载指引 中下载 SDK 文件，该版本中包括 GME Plugin，请将 SDK 文件解压到本地目录里。

步骤2：增加 Flutter 工程中 GME 插件的依赖

在您的 Flutter 项目中的 pubspec.yaml 文件中添加 GME 的依赖，注意参数中的 path 指上述 SDK 解压的路径

```
dependencies:
  flutter:
    sdk: flutter
  gme:
    path: ../
```

保存 pubspec.yaml 文件后在终端（命令行界面）中输入 flutter pub get 命令使您的项目中的 GME 插件生效（如果您的 IDE 中配置了 Flutter 的插件，保存将会自动执行该命令）。

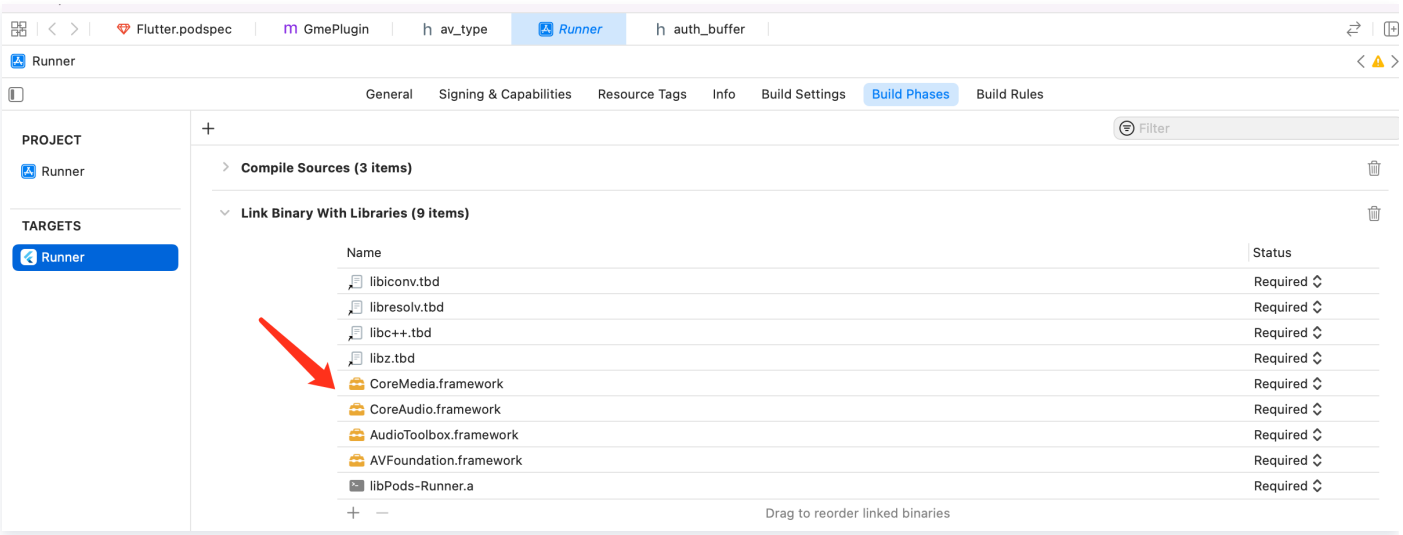
```
flutter pub get
```

iOS 工程修改

1. 在终端环境里进入到您的 Flutter 项目内 iOS 工程目录中，执行 pod install。

> ⚠ **注意：**
> 如出现Unable to determine Swift version for the following pods问题，请删除gme/ios/Classes/SwiftGmePlguin.swift文件
> 之后重新执行pod install命令

2. 在 xcode 工程中配置 GME 依赖的库（如果您的工程中本来就有，此步骤可以忽略）依赖文件如下图：

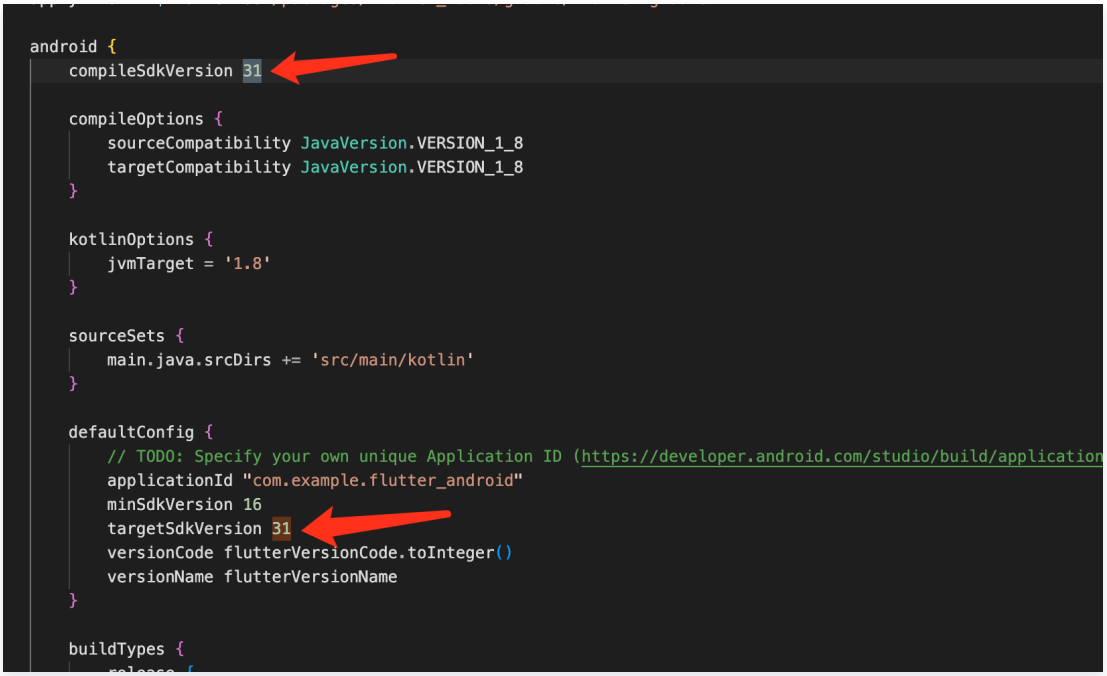


3. 游戏多媒体引擎 iOS 平台所需要的隐私权限如下：

- Required background modes：允许后台运行（可选）。
- Microphone Usage Description：允许麦克风权限。

Android 工程修改

1. 因 GME 需要获取通话等权限并使用了 flutter permission-handler 权限管理插件，所以需要使用31以上版本的 Android SDK（如在工程中已经使用请忽略），修改如图：



```
android {
    compileSdkVersion 31

    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }

    kotlinOptions {
        jvmTarget = '1.8'
    }

    sourceSets {
        main.java.srcDirs += 'src/main/kotlin'
    }

    defaultConfig {
        // TODO: Specify your own unique Application ID (https://developer.android.com/studio/build/application
        applicationId "com.example.flutter_android"
        minSdkVersion 16
        targetSdkVersion 31
        versionCode flutterVersionCode.toInteger()
        versionName flutterVersionName
    }

    buildTypes {
```

2. 在 flutter 工程文件 android/app/src/AndroidManifest.xml 中添加工程权限（如已修改请忽略）。

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

版权所有：腾讯云计算（北京）有限责任公司

实时语音

最近更新时间：2023-03-20 15:36:33

为方便 Flutter 开发者调试和接入腾讯云游戏多媒体引擎客户端 API，本文为您介绍适用于 Flutter 实时语音功能的开发接入技术文档。

使用 GME 重要事项

GME 提供实时语音服务、语音消息服务及转文本服务，使用 GME 服务都依赖 Init 和 Poll 等核心接口。

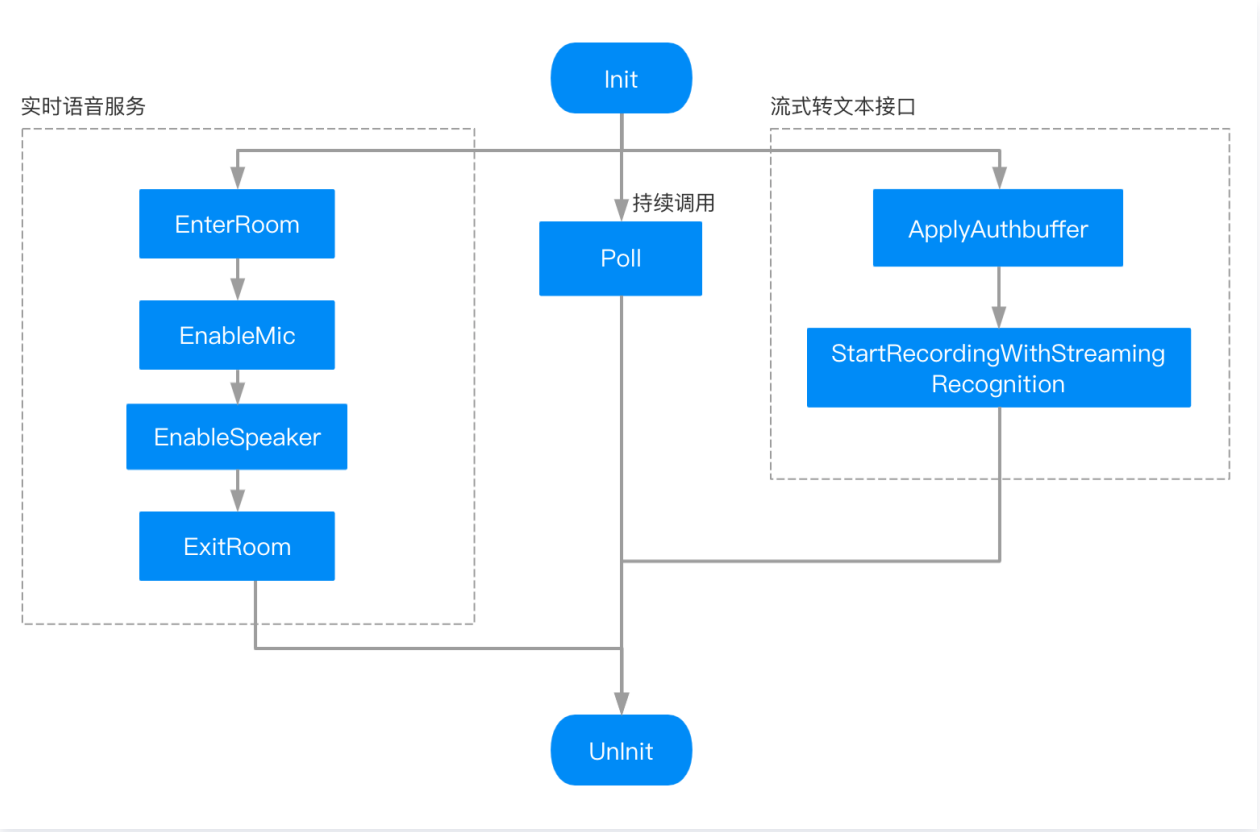
重点提示

- 已完成 GME 应用创建，并获取 SDK AppID 和 Key。请参见 服务开通指引 。
- 已开通 **GME 实时语音服务、语音消息服务以及转文本服务**。请参见 服务开通指引 。
- GME 使用前请对工程进行配置，否则 SDK 不生效。
- GME 的接口调用成功后返回值为 GmeError.AV_OK，数值为 0。
- GME 的接口调用要在同一个线程下。
- GME 需要周期性的调用 Poll 接口触发事件回调。
- 错误码详情可参见 错误码 。

接入 SDK

重要步骤

接入 SDK 重要流程如下：



1. 初始化 GME
2. 周期性调用 Poll 触发回调
3. 进入实时语音房间
4. 打开麦克风

5. 打开扬声器
6. 退出语音房间
7. 反初始化 GME

核心接口

接口	接口含义
Init	初始化 GME
Poll	触发事件回调
Pause	系统暂停
Resume	系统恢复
Uninit	反初始化 GME

引用 GME 模块

```
import 'package:gme/gme.dart';
import 'package:gme/gmeType.dart';
```

获取实例

在使用语音功能时，需要首先获取 GmeSDK 对象。

```
ITMGContext context = ITMGContext.GetInstance();
```

初始化 SDK

未初始化前，SDK 处于未初始化阶段，**需要通过接口 Init 初始化 SDK**，才可以使用实时语音服务、语音消息服务及转文本服务。调用 Init 接口的线程必须于其他接口在同一线程,建议都在主线程调用接口。

接口原型

```
//class ITMGContext
Future<int> InitSDK(String appID, String openID)
```

参数	类型	含义
sdkAppId	string	来自 腾讯云控制台 的 GME 服务提供的 AppID，获取请参见 语音服务开通指引 。
openID	string	openID 只支持int64 类型（转为 string 传入），规则由 App 开发者自行制定，App 内不重复即可。如需使用字符串作为 Openid 传入，可 提交工单 联系开发者。

返回值

返回值	处理
GmeError.AV_OK= 0	初始化 SDK 成功
AV_ERR_SDK_NOT_FULL_UPDATE=7015	检查 SDK 文件是否完整，建议删除后重新导入 SDK

> **关于7015错误提示**
> - 7015错误码是通过 md5 进行判断，在接入过程中若出现此错误，请根据提示检查 SDK 文件是否完整、SDK 文件版本是否一致。
> - 出现返回值 AV_ERR_SDK_NOT_FULL_UPDATE 时，此返回值**只有提示作用**，并不会造成初始化失败。
> - 由于第三方加固、Unity 打包机制等因素会影响库文件 md5，造成误判，所以**正式发布请在逻辑中忽略此错误**，并尽量不在 UI 中提示。

示例代码

```
string SDKAPPID3RD = "14000xxxxx";
string openId="10001";
int res = await ITMGContext.GetInstance().InitSDK(SDKAPPID3RD, openId);

if (ret != GmeError.AV_OK)
{
    print("Init SDK Error");
    return;
}
```

设置回调

接口类采用 Delegate 方法用于向应用程序发送回调通知。将回调函数注册给 SDK，用于接收回调的信息，需要在进房之前设置。

函数原型及示例代码

设置回调，用于接收回调的信息，需要在进房之前设置。

```
//在初始化 SDK 时候
ITMGContext.GetInstance().SetEvent(handleEventMsg);
//回调方法
void handleEventMsg(int eventType, String data) async {
    // enterRoom event
    print("AddDelegate3" + eventType.toString());
    switch (eventType) {
      case ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVENT_TYPE_ENTER_ROOM:
        {
        //进房回调
        }
        break;
      case ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE:
        {
         //切换房间回调
        }
        break;
    }
}
```

触发事件回调

需要周期的调用 Poll 可以触发事件回调。Poll 是 GME 的消息泵，GME 需要周期性的调用 Poll 接口触发事件回调。如果没有调用 Poll ，将会导致整个 SDK 服务运行异常。详情请参见 Sample Project 中的 EnginePollHelper 文件。

> **务必周期性调用 Poll 接口**

务必周期性调用 Poll 接口且在主线程调用，以免接口回调异常。

接口原型

```
Future<void> Poll();
```

示例代码

```
  Future<void> pollTimer() async {_pollTimer = Timer.periodic(Duration(milliseconds: 100),
(Timer timer) {
      ITMGContext.GetInstance().Poll();
    });
  }
```

系统暂停

当系统发生 Pause 事件时，需要同时通知引擎进行 Pause。例如在应用退后台时候（OnApplicationPause, isPause=True），如果不需要后台播放房间内声音，请调用 Pause 接口暂停整个 GME 服务。

接口原型

```
Future<int> Pause()
```

系统恢复

当系统发生 Resume 事件时，需要同时通知引擎进行 Resume。Resume 接口只恢复实时语音。

接口原型

```
Future<int> Resume()
```

反初始化 SDK

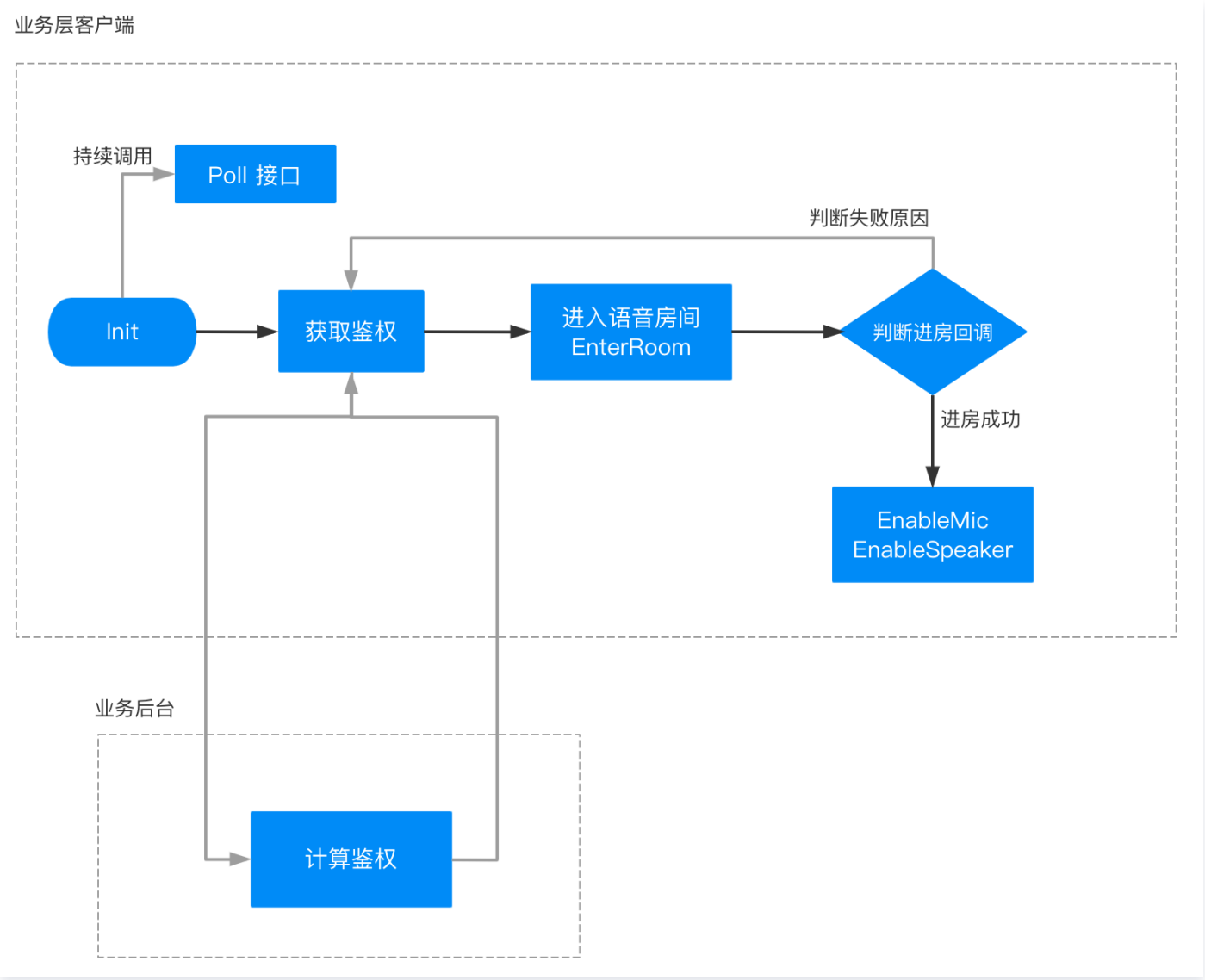
反初始化 SDK，进入未初始化状态。**如果游戏业务侧账号与 openid 是绑定的，那切换游戏账号需要反初始化 GME，再用新的 openid 初始化。**

接口原型

```
Future<int> Uninit()
```

实时语音房间相关接口

初始化之后，SDK 调用进房后进去了房间，才可以进行实时语音通话。
使用问题可参见 实时语音相关问题 。



接口	接口含义
GenAuthBuffer	本地鉴权计算
EnterRoom	加入房间
ExitRoom	退出房间
IsRoomEntered	判断是否已经进入房间

本地鉴权计算

生成 AuthBuffer，用于相关功能的加密和鉴权，如正式发布请使用后台部署密钥，后台部署请参见 鉴权密钥 。

接口原型

```
Future<Uint8List> GenAuthBuffer(String appID, String roomID, String openID, String key)
```

参数	类型	含义
appID	string	来自腾讯云控制台的 AppID 号码。

roomID	string	房间号，最大支持127字符。
openID	string	用户标识。与 Init 时候的 openID 相同。
key	string	来自腾讯云 控制台 的权限密钥。

示例代码

```
 Uint8List userSig = await ITMGContext.GetInstance().GenAuthBuffer(_editAppID.text,
_editRoomID.text, _editOpenID.text, _editKey.text);
 int res = await ITMGContext.GetInstance().EnterRoom(_editRoomID.text, 1, userSig);
```

加入房间

用生成的鉴权信息进房，加入房间默认不打开麦克风及扬声器。

注意

- 加入房间事件回调结果 result 为 0 代表进房成功，进房接口 EnterRoom 返回值为 0 不代表进房成功。
- 房间的音频类型由第一个进房的人确定，此后房间里有成员修改房间类型，将对此房间所有成员生效。例如第一个进入房间的人使用的房间音频类型是流畅音质，第二个进房的是即使进房时候调用接口的音频类型参数是高清音质，进入房间之后也会变成流畅音质。需要有成员调用 ChangeRoomType 才会修改房间的音频类型。

接口原型

```
Future<int> EnterRoom(String roomID, int roomType, Uint8List authBuffer)
```

参数	类型	含义
roomId	string	房间号，最大支持127字符
roomType	ITMGRoomType	房间类型，游戏建议使用 ITMG_ROOM_TYPE_FLUENCY。房间音频类型请参见 音质选择 。
appKey	Uint8List	鉴权码

示例代码

```
int res = await ITMGContext.GetInstance().EnterRoom(_editRoomID.text, 1, authBuffer);
```

加入房间事件回调

加入房间完成后会通过回调返回ITMG_MAIN_EVENT_TYPE_ENTER_ROOM事件类型返回进房结果，监听进房结果事件后进行处理。如果回调为成功，即此时进房成功，开始进行**计费**。

计费问题参考：

- 购买指南。
- 计费相关问题。
- 使用实时语音后，如果客户端掉线了，是否还会继续计费?

示例代码

```
//对事件进行监听：
void handleEventMsg(int eventType, String data) async {
  switch (eventType) {
      case ITMG_MAIN_EVENT_TYPE_ENTER_ROOM:
      {
        //进房后续流程处理
      }
  }
}
ITMGContext.GetInstance().SetEvent(handleEventMsg);
```

Data 详情

消息	Data	例子
ITMG_MAIN_EVENT_TYPE_ENTER_ROOM	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_ROOM_DISCONNECT	result; error_info	{"error_info":"waiting timeout, please check your network","result":0}

如果断网，将会有断网的回调提示 `ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVENT_TYPE_ROOM_DISCONNECT`，此时 SDK 会自动进行重连，回调是 `ITMG_MAIN_EVENT_TYPE_RECONNECT_START`，当重连成功时，会有 `ITMG_MAIN_EVENT_TYPE_RECONNECT_SUCCESS` 回调。

错误码

错误码值	原因及建议方案
7006	鉴权失败原因： • AppID 不存在或者错误 • authbuff 鉴权错误 • 鉴权过期 • OpenId 不符合规范
7007	已经在其它房间
1001	已经在进房过程中，然后又重复了此操作。建议在进房回调返回之前不要再调用进房接口
1003	已经进房了在房间中，又调用一次进房接口
1101	确保已经初始化 SDK，确保 OpenId 是否符合规则，或者确保在同一线程调用接口，以及确保 Poll 接口正常调用

退出房间

通过调用此接口可以退出所在房间。这是一个异步接口，返回值为 AV_OK 的时候代表异步投递成功。如果应用中有退房后立即进房的场景，在接口调用流程上，开发者无需要等待 ExitRoom 的回调 RoomExitComplete 通知，只需直接调用接口。

接口原型

```
Future<int> ExitRoom()
```

示例代码

```
ITMGContext.GetInstance().ExitRoom();
```

退出房间事件回调

退出房间完成后会有回调，消息为 ITMG_MAIN_EVENT_TYPE_EXIT_ROOM。示例代码如下：

示例代码

```
void handleEventMsg(int eventType, String data) async{
  switch (eventType) {
      case ITMG_MAIN_EVENT_TYPE_EXIT_ROOM:
      {
      //退房后续进行处理
        break;
      }
  }
}
ITMGContext.GetInstance().SetEvent(handleEventMsg);
```

判断是否已经进入房间

通过调用此接口可以判断是否已经进入房间，返回值为 bool 类型。请勿在进房过程中调用。

接口原型

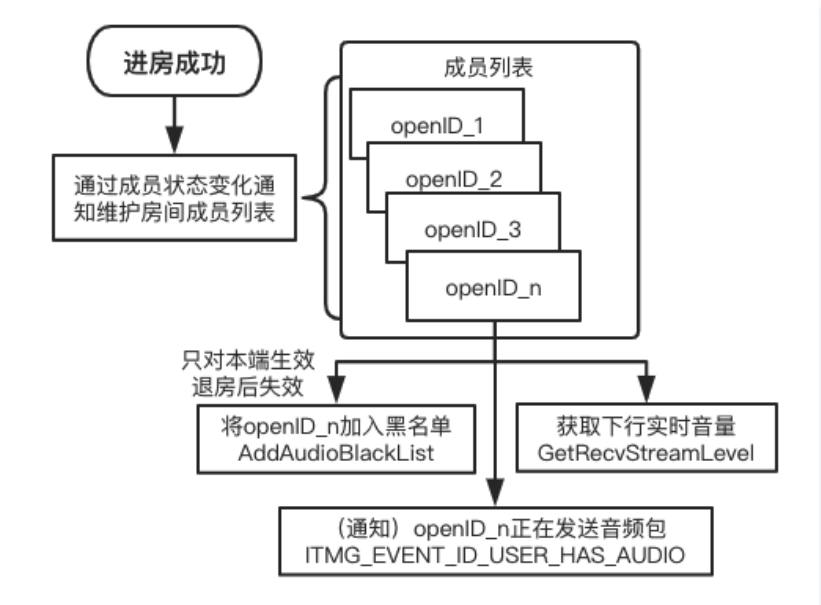
```
Future<bool> IsRoomEntered()
```

示例代码

```
bool res = await ITMGContext.GetInstance().IsRoomEntered();
```

房间内状态维护

此部分接口用于业务层显示说话成员、进退房成员，以及将房间内某成员禁言等功能。



接口/通知	含义
ITMG_MAIN_EVNET_TYPE_USER_UPDATE	成员状态变化通知

AddAudioBlackList	房间中禁言某成员
RemoveAudioBlackList	移除禁言
IsOpenIdInAudioBlackList	查询某openid是否被禁言

成员进房、说话状态通知事件

- 该事件适用于获取房间中说话的人并在 UI 中展示，以及有人进入、退出语音房间的一个通知。
- 该事件在状态变化才通知，状态不变化的情况下不通知。如需实时获取成员状态，请在业务层收到通知时缓存，事件消息为 ITMG_MAIN_EVNET_TYPE_USER_UPDATE，包含 event_id、count 及 openIdList，在 OnEvent 通知中对事件消息进行判断。
- 音频事件 EVENT_ID_ENDPOINT_NO_AUDIO 的通知有一个阈值，超过这个阈值才会发送通知。即本端两秒没采集到声音后， 房间其他成员才收到本端停止说话的通知。
- 音频事件只会返回成员说话状态，没有返回具体的音量。如需房间内成员具体音量可使用接口 GetVolumeById 进行获取。

event_id	含义	应用侧维护内容
EVENT_ID_ENDPOINT_ENTER	有成员进入房间，返回此时进房的 openid	应用侧维护成员列表
EVENT_ID_ENDPOINT_EXIT	有成员退出房间，返回此时退房的 openid	应用侧维护成员列表
EVENT_ID_ENDPOINT_HAS_AUDIO	有成员发送音频包，返回此时房间内说话的 openid，通过此事件可以判断用户是否说话，并展示声纹效果	应用侧维护通话成员列表
EVENT_ID_ENDPOINT_NO_AUDIO	有成员停止发送音频包，返回此时房间内停止说话的 openid	应用侧维护通话成员列表

示例代码

```
void handleEventMsg(int eventType, String data) async {
  if (eventType == ITMG_MAIN_EVENT_TYPE_ENTER_ROOM)
  {
              //进行处理
            switch (eventID)
             {
             case EVENT_ID_ENDPOINT_ENTER:
                  //有成员进入房间
                  break;
             case EVENT_ID_ENDPOINT_EXIT:
                  //有成员退出房间
                break;
            case EVENT_ID_ENDPOINT_HAS_AUDIO:
                //有成员发送音频包
                break;
            case EVENT_ID_ENDPOINT_NO_AUDIO:
                //有成员停止发送音频包
                break;

            default:
                break;
             }
        break;
  }
```

```
}
```

房间中禁言某成员

将某个 ID 加入音频数据黑名单，即不接受某人的语音， 只对本端生效，不会影响其他端。返回值为 0 表示调用成功。例如： A、B、C 都在同一个房间开麦说话：

- 如果 A 设置了 C 的黑名单， 则 A 只能听见 B 的声音。
- B 因为没有设置黑名单， 仍旧可以听见 A 和 C 的声音。
- C 同样因为没有设置黑名单， 可以听见 A 和 B 的声音。

此接口适用于在语音房间中将某用户禁言的场景。

接口原型

```
Future<int> AddAudioBlackList(String openID)
```

参数	类型	含义
openID	string	需添加黑名单的用户 openid

示例代码

```
res = await ITMGContext.GetInstance().GetAudioCtrl().AddAudioBlackList(_editRoomManagerID.text);
```

移除禁言

将某个 Id 移除音频数据黑名单。返回值为0表示调用成功。

接口原型

```
Future<int> RemoveAudioBlackList(String openID)
```

参数	类型	含义
openId	string	需移除黑名单的 ID

示例代码

```
res = await
ITMGContext.GetInstance().GetAudioCtrl().RemoveAudioBlackList(_editRoomManagerID.text);
```

实时语音采集相关接口

- 初始化 SDK 之后进房，在房间中，才可以调用实时音频语音相关接口。
- 当用户界面单击打开/关闭麦克风/扬声器按钮时，建议采用 EnableMic 以及 EnableSpeaker 接口进行调用。
- 当用户界面按住麦克风按钮时发言，放开按钮不发言，建议采用进房时候调用 EnableAudioCaptureDevice 一次，后续按住发言调用 EnableAudioSend 来实现。

接口	接口含义
EnableMic	开关麦克风
GetMicState	获取麦克风状态

EnableAudioCaptureDevice	开关采集设备
IsAudioCaptureDeviceEnabled	获取采集设备状态
EnableAudioSend	打开关闭音频上行
IsAudioSendEnabled	获取音频上行状态
GetMicLevel	获取实时麦克风音量
GetSendStreamLevel	获取音频上行实时音量
SetMicVolume	设置麦克风音量
GetMicVolume	获取麦克风音量

开启或关闭麦克风

此接口用来开启关闭麦克风。加入房间默认不打开麦克风及扬声器。EnableMic = EnableAudioCaptureDevice + EnableAudioSend

接口原型

```
Future<int> EnableMic(bool enable)
```

参数	类型	含义
isEnabled	bool	如果需要打开麦克风，则传入的参数为 true，如果关闭麦克风，则参数为 false

示例代码

```
//打开麦克风
int res = await ITMGContext.GetInstance().GetAudioCtrl().EnableMic(true);
```

麦克风状态获取

此接口用于获取麦克风状态，返回值0为关闭麦克风状态，返回值1为打开麦克风状态。

接口原型

```
Future<int> GetMicState()
```

示例代码

```
int micState = await ITMGContext.GetInstance().GetAudioCtrl().GetMicState();
```

开启或关闭采集设备

此接口用来开启/关闭采集设备。加入房间默认不打开设备。

- 只能在进房后调用此接口，退房会自动关闭设备。
- 在移动端，打开采集设备通常会伴随权限申请，音量类型调整等操作。

接口原型

```
Future<int> EnableAudioCaptureDevice(bool enable)
```

参数	类型	含义
enable	bool	如果需要打开采集设备，则传入的参数为 true，如果关闭采集设备，则参数为 false

示例代码

```
//打开采集设备
int res = await ITMGContext.GetInstance().GetAudioCtrl().EnableAudioCaptureDevice(true);
```

采集设备状态获取

此接口用于采集设备状态获取。

接口原型

```
Future<bool> IsAudioCaptureDeviceEnabled()
```

示例代码

```
bool res = await ITMGContext.GetInstance().GetAudioCtrl().IsAudioCaptureDeviceEnabled();
```

打开或关闭音频上行

此接口用于打开/关闭音频上行。如果采集设备已经打开，那么会发送采集到的音频数据。如果采集设备没有打开，那么仍旧无声。采集设备的打开关闭请参见接口 EnableAudioCaptureDevice。

接口原型

```
Future<int> EnableAudioSend(bool enable)
```

参数	类型	含义
isEnabled	bool	如果需要打开音频上行，则传入的参数为 true，如果关闭音频上行，则参数为 false

示例代码

```
int res = await ITMGContext.GetInstance().GetAudioCtrl().EnableAudioSend(isCheck);
```

音频上行状态获取

此接口用于音频上行状态获取。

接口原型

```
Future<bool> IsAudioSendEnabled()
```

示例代码

```
bool IsAudioSend = await ITMGContext.GetInstance().GetAudioCtrl().IsAudioSendEnabled();
```

获取麦克风实时音量

此接口用于获取麦克风实时音量，返回值为 number 类型。建议20ms获取一次。值域为0 - 100，通过此接口可以获取到麦克风采集到的实时音量情况。

接口原型

```
Future<int> GetMicLevel()
```

示例代码

```
int res = await ITMGContext.GetInstance().GetAudioCtrl().GetMicLevel();
```

获取音频上行实时音量

此接口用于获取自己音频上行实时音量，返回值为 number 类型，取值范围为0 - 100。

接口原型

```
Future<int> GetSendStreamLevel()
```

示例代码

```
int res = await ITMGContext.GetInstance().GetAudioCtrl().GetSendStreamLevel();
```

设置麦克风软件音量

此接口用于设置麦克风的音量。参数 volume 用于设置麦克风的音量，相当于对采集的声音做衰减或增益。

接口原型

```
 Future<int> SetMicVolume(int volume)
```

参数	类型	含义
volume	number	取值范围为 0-200，数值为0的时候表示静音，当数值为100的时候表示音量不增不减，默认数值为100。

示例代码

```
int volume = 100;
int res = await ITMGContext.GetInstance().GetAudioCtrl().SetMicVolume(volume);
```

获取麦克风软件音量

此接口用于获取麦克风的音量。返回值为一个number类型数值，返回值为101代表没调用过接口 SetMicVolume。

接口原型

```
Future<int> GetMicVolume()
```

示例代码

```
int res = await ITMGContext.GetInstance().GetAudioCtrl().GetMicVolume();
```

实时语音播放相关接口

接口	接口含义
EnableSpeaker	开关扬声器
GetSpeakerState	获取扬声器状态
EnableAudioPlayDevice	开关播放设备
IsAudioPlayDeviceEnabled	获取播放设备状态
EnableAudioRecv	打开关闭音频下行
IsAudioRecvEnabled	获取音频下行状态
GetSpeakerLevel	获取实时扬声器音量
GetRecvStreamLevel	获取房间内其他成员下行实时音量
SetSpeakerVolume	设置扬声器音量
GetSpeakerVolume	获取扬声器音量

开启或关闭扬声器

此接口用于开启关闭扬声器。**EnableSpeaker = EnableAudioPlayDevice + EnableAudioRecv**

接口原型

```
Future<int> EnableSpeaker(bool enable)
```

参数	类型	含义
bEnable	bool	如果需要关闭扬声器，则传入的参数为 false，如果打开扬声器，则参数为 true

示例代码

```
//打开扬声器
await ITMGContext.GetInstance().GetAudioCtrl().EnableSpeaker(isCheck);
```

扬声器状态获取

此接口用于扬声器状态获取。返回值0为关闭扬声器状态，返回值1为打开扬声器状态。

接口原型

```
Future<int> GetSpeakerState()
```

示例代码

```
int spkState = await ITMGContext.GetInstance().GetAudioCtrl().GetSpeakerState();
```

开启或关闭播放设备

此接口用于开启关闭播放设备。

接口原型

```
Future<int> EnableAudioPlayDevice(bool enable)
```

参数	类型	含义
enable	bool	如果需要关闭播放设备，则传入的参数为 false，如果打开播放设备，则参数为 true

示例代码

```
int res = await ITMGContext.GetInstance().GetAudioCtrl().EnableAudioPlayDevice(isCheck);
```

播放设备状态获取

此接口用于播放设备状态获取。

接口原型

```
Future<bool> IsAudioPlayDeviceEnabled()
```

示例代码

```
bool res = await ITMGContext.GetInstance().GetAudioCtrl().IsAudioPlayDeviceEnabled();
```

打开或关闭音频下行

此接口用于打开/关闭音频下行。如果播放设备已经打开，那么会播放房间里其他人的音频数据。如果播放设备没有打开，那么仍旧无声。播放设备的打开关闭参见接口请参见 EnableAudioPlayDevice。

接口原型

```
Future<int> EnableAudioRecv(bool enable)
```

参数	类型	含义
isEnabled	bool	如果需要打开音频下行，则传入的参数为 true，如果关闭音频下行，则参数为 false

示例代码

```
int res = await ITMGContext.GetInstance().GetAudioCtrl().EnableAudioRecv(isCheck);
```

音频下行状态获取

此接口用于音频下行状态获取。

接口原型

```
Future<bool> IsAudioRecvEnabled()
```

示例代码

```
bool res = await ITMGContext.GetInstance().GetAudioCtrl().IsAudioRecvEnabled();
```

获取扬声器实时音量

此接口用于获取扬声器实时音量。返回值为 number 类型数值，表示扬声器实时音量。建议20ms获取一次。

接口原型

```
Future<int> GetSpeakerLevel()
```

示例代码

```
bool res = await ITMGContext.GetInstance().GetAudioCtrl().GetSpeakerLevel();
```

获取房间内其他成员下行实时音量

此接口用于获取房间内其他成员下行实时音量，返回值为 number 类型，取值范围为0 - 200。

接口原型

```
Future<int> GetRecvStreamLevel(String openID)
```

参数	类型	含义
openId	string	房间其他成员的openId

示例代码

```
int res = await
ITMGContext.GetInstance().GetAudioCtrl().GetRecvStreamLevel(_editRoomManagerID.text);
```

动态设置房间内某成员音量

此接口用于设置房间内某成员的说话音量大小，此设置只在本端生效。

接口原型

```
Future<int> SetSpeakerVolumeByOpenID(String openId, int volume)
```

参数	类型	含义
openId	string	需要调节音量大小的OpenID
volume	number	百分比，建议[0-200]，其中100为默认值

示例代码

```
int res = await
ITMGContext.GetInstance().GetAudioCtrl().SetSpeakerVolumeByOpenID(_editRoomManagerID.text, 100);
```

获取设置的声音百分比

调用此接口获取 SetSpeakerVolumeByOpenID 设置的能量值

接口原型

```
Future<int> GetSpeakerVolumeByOpenID(String openId)
```

参数	类型	含义
openId	string	需要调节音量大小的OpenID

返回值

接口返回 OpenID 设置的能量百分比， 默认返回100。

示例代码

```
int res = await
ITMGContext.GetInstance().GetAudioCtrl().GetSpeakerVolumeByOpenID(_editRoomManagerID.text);
```

设置扬声器的音量

此接口用于设置扬声器的音量。

接口原型

```
Future<int> SetSpeakerVolume(int volume)
```

参数	类型	含义
volume	number	设置音量，范围0 – 200，当数值为0时，表示静音，当数值为100时，表示音量不增不减，默认数值为100。

示例代码

```
int volume = value.toInt();
int res = await ITMGContext.GetInstance().GetAudioCtrl().SetSpeakerVolume(volume);
```

获取扬声器的音量

此接口用于获取扬声器的音量。返回值为 number 类型数值，代表扬声器的音量，返回值为101代表没调用过接口 SetSpeakerVolume。Level 是实时音量，Volume 是扬声器的音量，最终声音音量 = Level × Volume %。例如实时音量是数值是100，此时 Volume 的数值是60，那么最终发出来的声音数值也是60。

接口原型

```
Future<int> GetSpeakerVolume()
```

示例代码

```
int res = await ITMGContext.GetInstance().GetAudioCtrl().GetSpeakerVolume();
```

高级 API

启动耳返

此接口用于启动耳返，需要 EnableLoopBack+EnableSpeaker 才可以听到自己声音。

接口原型

```
Future<int> EnableLoopBack(bool enable)
```

参数	类型	含义
enable	bool	设置是否启动

示例代码

```
int res = await ITMGContext.GetInstance().GetAudioCtrl().EnableLoopBack(true);
```

获取用户房间音频类型

此接口用于获取用户房间音频类型，返回值为房间音频类型，返回值为0时代表获取用户房间音频类型发生错误，房间音频类型参考 EnterRoom 接口。

接口原型

```
Future<int> GetRoomType()
```

示例代码

```
int curType = await ITMGContext.GetInstance().GetRoom().GetRoomType();
```

房间类型修改

此接口用于修改用户房间音频类型，结果参见回调事件，事件类型为 ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE。房间的音频类型由第一个进房的人确定，此后房间里有成员修改房间类型，将对此房间所有成员生效。

接口原型

```
Future<int> ChangeRoomType(int roomType)
```

参数	类型	含义
roomtype	number	房间切换成的目标类型，房间音频类型参考 EnterRoom 接口

示例代码

```
int res = await ITMGContext.GetInstance().GetRoom().ChangeRoomType(1);
```

回调事件

房间类型设置完成后，回调的事件消息为 ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE，返回的参数为 result、error_info 及 new_room_type，new_room_type 代表的信息如下，在 OnEvent 函数中对事件消息进行判断。

事件子类型	代表参数	含义
ITMG_ROOM_CHANGE_EVENT_ENTERROOM	1	表示在进房的过程中，自带的音频类型与房间不符合，被修改为所进入房间的音频类型
ITMG_ROOM_CHANGE_EVENT_START	2	表示已经在房间内，音频类型开始切换（例如调用 ChangeRoomType 接口后切换音频类型 ）
ITMG_ROOM_CHANGE_EVENT_COMPLETE	3	表示已经在房间，音频类型切换完成
ITMG_ROOM_CHANGE_EVENT_REQUEST	4	表示房间成员调用 ChangeRoomType 接口，请求切换房间音频类型

示例代码

```
case ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE:
  {
        //对房间类型事件进行处理
  }
  break;
```

房间通话质量监控事件

质量监控事件，此通知事件适用于监听网络质量，如果用户网络差的话，业务层将通过 UI 提醒用户切换网络。在进房后触发，事件2秒回调一次，事件消息为 ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_QUALITY，返回的参数为 weight、loss 及 delay，代表的信息如下：

参数	类型	含义
weight	number	范围是1 – 50，数值为50是音质评分极好，数值为1是音质评分很差，几乎不能使用，数值为0代表初始值，无含义。一般数值在30以下就可以提醒用户网络较差，建议切换网络。
loss	var	上行丢包率。
delay	number	音频触达延迟时间（ms）。

获取版本号

获取 SDK 版本号，用于分析。

接口原型

```
Future<String> GetSDKVersion()
```

示例代码

```
_sdkVersions = await ITMGContext.GetInstance().GetSDKVersion();
```

设置应用名称和版本

该接口用于设置应用名称和版本

接口原型

```
Future<void> SetAppVersion(String appVersion)
```

参数含义

参数	类型	含义
appVersion	string	应用名称和版本

示例代码

```
await ITMGContext.GetInstance().SetAppVersion("gme V2.0.0");
```

设置打印日志等级

用于设置打印日志等级。建议保持默认等级。需要在 Init 之前调用。

接口原型

```
Future<int> SetLogLevel(int levelWrite, int levelPrint)
```

参数含义

参数	类型	含义
level	number	设置日志的等级，TMG_LOG_LEVEL_NONE 表示不写入，默认为 TMG_LOG_LEVEL_INFO

level 说明如下：

level	含义
TMG_LOG_LEVEL_NONE	不打印日志
TMG_LOG_LEVEL_ERROR	打印错误日志（默认）
TMG_LOG_LEVEL_INFO	打印提示日志
TMG_LOG_LEVEL_DEBUG	打印开发调试日志
TMG_LOG_LEVEL_VERBOSE	打印高频日志

示例代码

```
ITMGContext.GetInstance().SetLogLevel(ITMG_LOG_LEVEL.TMG_LOG_LEVEL_ERROR,
ITMG_LOG_LEVEL.TMG_LOG_LEVEL_ERROR);
```

设置打印日志路径

用于设置打印日志路径。默认路径如下。需要在 Init 之前调用。

接口原型

```
Future<int> SetLogPath(String logDir)
```

参数	类型	含义
logPath	string	路径

示例代码

```
String curPath = ""//自行设置路径
ITMGContext.GetInstance().SetLogPath(curPath);
```

获取诊断信息

获取音视频通话的实时通话质量的相关信息。该接口主要用来查看实时通话质量、排查问题等，业务侧可以忽略。

接口原型

```
Future<String> GetQualityTips()
```

示例代码

```
String curQualityTips = await ITMGContext.GetInstance().GetRoom().GetQualityTips();
```

回调消息

消息	含义	Data	例子 |
ITMG_MAIN_EVENT_TYPE_ENTER_ROOM	进入音频房间消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_EXIT_ROOM	退出音频房间消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_ROOM_DISCONNECT	房间因为网络等原因断开消息	result; error_info	{"error_info":"waiting timeout, please check your network","result":0}
ITMG_MAIN_EVNET_TYPE_USER_UPDATE	房间成员更新消息	user_list; event_id	{"event_id":1,"user_list":["0"]}
ITMG_MAIN_EVENT_TYPE_RECONNECT_START	房间重连开始消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_RECONNECT_SUCCESS	房间重连成功消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_SWITCH_ROOM	快速切换房间消息	result; error_info	{"error_info":"","result":0}
ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_TYPE	房间状态切换消息	result; error_info; sub_event_type;	{"error_info":"","new_room_type":0,"result":0}

		new_room_type	
ITMG_MAIN_EVENT_TYPE_ROOM_SHARING_START	开始跨房连麦消息	result;	{"result":0}
ITMG_MAIN_EVENT_TYPE_ROOM_SHARING_STOP	跨房连麦停止消息	result;	{"result":0}
ITMG_MAIN_EVENT_TYPE_SPEAKER_DEFAULT_DEVICE_CHANGED	默认扬声器设备修改消息	result; error_info	{"deviceID":"{0.0.0.00000000}.{a4f1e8be-49fa-43e2-b8cf-dd00542b47ae}","deviceName":"扬声器 (Realtek High Definition Audio)","error_info":"","isNewDevice":true,"isUsedDevice":false,"result":0}
ITMG_MAIN_EVENT_TYPE_SPEAKER_NEW_DEVICE	新增扬声器设备消息	result; error_info	{"deviceID":"{0.0.0.00000000}.{a4f1e8be-49fa-43e2-b8cf-dd00542b47ae}","deviceName":"扬声器 (Realtek High Definition Audio)","error_info":"","isNewDevice":true,"isUsedDevice":false,"result":0}
ITMG_MAIN_EVENT_TYPE_SPEAKER_LOST_DEVICE	丢失扬声器设备消息	result; error_info	{"deviceID":"{0.0.0.00000000}.{a4f1e8be-49fa-43e2-b8cf-dd00542b47ae}","deviceName":"扬声器 (Realtek High Definition Audio)","error_info":"","isNewDevice":false,"isUsedDevice":false,"result":0}
ITMG_MAIN_EVENT_TYPE_MIC_NEW_DEVICE	新增麦克风设备消息	result; error_info	{"deviceID":"{0.0.1.00000000}.{5fdf1a5b-f42d-4ab2-890a-7e454093f229}","deviceName":"麦克风 (Realtek High Definition Audio)","error_info":"","isNewDevice":true,"isUsedDevice":true,"result":0}
ITMG_MAIN_EVENT_TYPE_MIC_LOST_DEVICE	丢失麦克风设备消息	result; error_info	{"deviceID":"{0.0.1.00000000}.{5fdf1a5b-f42d-4ab2-890a-7e454093f229}","deviceName":"麦克风 (Realtek High Definition Audio)","error_info":"","isNewDevice":false,"isUsedDevice":true,"result":0}
ITMG_MAIN_EVENT_TYPE_MIC_DEFAULT_DEVICE_CHANGED	默认麦克风设备修改消息	result; error_info	{"deviceID":"{0.0.1.00000000}.{5fdf1a5b-f42d-4ab2-890a-7e454093f229}","deviceName":"麦克风 (Realtek High Definition Audio)","error_info":"","isNewDevice":false,"isUsedDevice":true,"result":0}
ITMG_MAIN_EVENT_TYPE_CHANGE_ROOM_QUALITY	房间质量消息	weight; loss; delay	{"weight":5,"loss":0.1,"delay":1}
ITMG_MAIN_EVNET_TYPE_PTT_RECORD_COMPLETE	语音消息录制完成消息	result; file_path	{"file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_UPLOAD_COMPLETE	语音消息上传完成消息	result; file_path;file_id	{"file_id":"","file_path":"","result":0}

ITMG_MAIN_EVNET_TYPE_PTT_DOWNLOAD_COMPLETE	语音消息下载完成消息	result; file_path;file_id	{"file_id":"","file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_PLAY_COMPLETE	语音消息播放完成消息	result; file_path	{"file_path":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_SPEECH2TEXT_COMPLETE	语音消息极速转文本完成消息	result; text;file_id	{"file_id":"","text":"","result":0}
ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_COMPLETE	语音消息流式转文本完成消息	result; file_path; text;file_id	{{"file_id":"","file_path":","text":"","result":0}}
ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_IS_RUNNING	语音消息正在流式转文本中	result; file_path; text;file_id	{{"file_id":"","file_path":","text":"","result":0}}
ITMG_MAIN_EVNET_TYPE_PTT_TEXT2SPEECH_COMPLETE	文本转语音完成消息	result; text;file_id	{{"file_id":"","text":"","result":0}}
ITMG_MAIN_EVNET_TYPE_PTT_TRANSLATE_TEXT_COMPLETE	文本翻译完成消息	result; text;file_id	{{"file_id":"","text":"","result":0}}

语音消息及转文本

最近更新时间：2024-12-17 17:39:22

为方便 Flutter 开发者调试和接入腾讯云游戏多媒体引擎客户端 API，本文为您介绍适用于 Flutter 语音消息服务及转文本服务的接入技术文档。

使用 GME 重要事项

GME 提供实时语音服务、语音消息服务及转文本服务，使用 GME 服务都依赖 Init 和 Poll 等核心接口。

重点提示

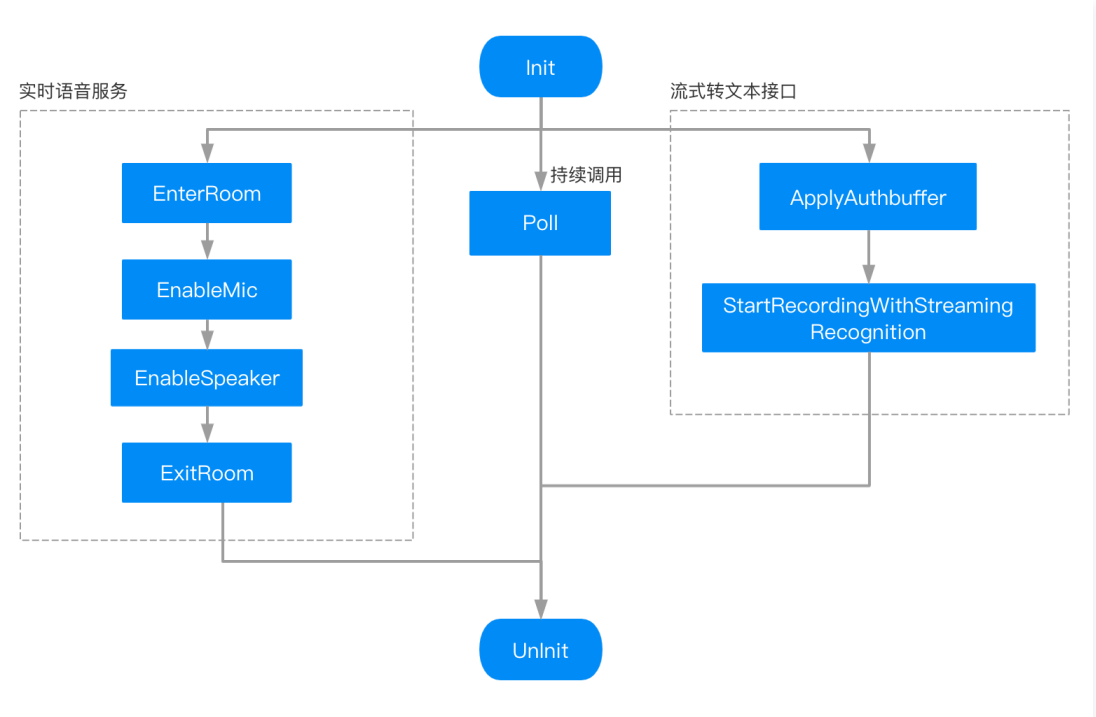
- 已完成 GME 应用创建，并获取 SDK AppID 和 Key。请参见 服务开通指引 。
- 已开通 **GME 实时语音服务、语音消息服务以及转文本服务**。请参见 服务开通指引 。
- GME 使用前请对工程进行配置，否则 SDK 不生效。
- GME 的接口调用成功后返回值为 GmeError.AV_OK，数值为 0。
- GME 的接口调用要在同一个线程下。
- GME 需要周期性的调用 Poll 接口触发事件回调。
- 错误码详情可参见 错误码 。

> ⚠ **注意**
> 语音转文本相关接口有默认频率限制，限额范围内计费方式请参见 计费文档；若需提升接口频率限额或了解超额计费方式，请联系商务或 提交工单咨询 。
> - 语音消息非流式转文本接口 ***SpeechToText()***：默认单账号限制并发数为10路
> - 语音消息流式转文本接口 ***StartRecordingWithStreamingRecognition()***：默认单账号限制并发数为50路

接入 SDK

重要步骤

接入 SDK 重要流程如下：



1. 初始化 GME

2. 周期性调用 Poll 触发回调
3. 鉴权初始化
4. 启动流式语音识别
5. 停止录制
6. 反初始化 GME

dart 文件

```
gme.dart       GME 业务实现接口
gmeType.dart   GME 类型定义文件
gmeError.dart GME 错误类型定义文件
```

核心接口

接口	接口含义
InitSDK	初始化 GME
Poll	触发事件回调
Uninit	反初始化 GME

导入Gme模块

```
import 'package:gme/gme.dart';
import 'package:gme/gmeType.dart';
```

获取实例

var m_context = await ITMGContext.GetInstance();

初始化 SDK

未初始化前，SDK 处于未初始化阶段，需要通过接口 Init 初始化 SDK，才可以使用实时语音服务、语音消息服务及转文本服务。调用 Init 接口的线程必须于其他接口在同一线程,建议都在主线程调用接口。

接口原型

```
Future<int> InitSDK(String appID, String openID)
```

参数	类型	含义
sdkAppId	string	来自 腾讯云控制台 的 GME 服务提供的 AppID，获取请参见 语音服务开通指引 。
openID	string	openID 只支持 Int64 类型（转为 string 传入），规则由 App 开发者自行制定，App 内不重复即可。如需使用字符串作为 Openid 传入，可 提交工单 联系开发者。

返回值

返回值	处理
GmeError.AV_OK= 0	初始化 SDK 成功

AV_ERR_SDK_NOT_FULL_UPDATE=7015	检查 SDK 文件是否完整，建议删除后重新导入 SDK

关于7015错误提示

- 7015错误码是通过 md5 进行判断，在接入过程中若出现此错误，请根据提示检查 SDK 文件是否完整、SDK 文件版本是否一致。
- 出现返回值 AV_ERR_SDK_NOT_FULL_UPDATE 时，此返回值**只有提示作用**，并不会造成初始化失败。

示例代码

```
int res = await ITMGContext.GetInstance().InitSDK(_editAppID.text,_editOpenID.text);
//通过返回值判断是否初始化成功
if (ret != GmeError.AV_OK)
{
    print("SDK初始化失败:");
    return;
}
```

触发事件回调

通过在定时器调用 Poll 可以触发事件回调。Poll 是 GME 的消息泵，GME 需要周期性的调用 Poll 接口触发事件回调。如果没有调用 Poll ，将会导致整个 SDK 服务运行异常。详情请参见 Sample Project 中的 EnginePollHelper 文件。

注意

务必周期性调用 Poll 接口且在主线程调用，以免接口回调异常。

接口原型

```
Future<void> Poll()
```

示例代码

```
Future<void> pollTimer() async {
  _pollTimer = Timer.periodic(Duration(milliseconds: 100), (Timer timer) {
  ITMGContext.GetInstance().Poll();
  });
}
```

反初始化 SDK

反初始化 SDK，进入未初始化状态。**如果游戏业务侧账号与 openid 是绑定的，那切换游戏账号需要反初始化 GME，再用新的 openid 初始化。**

接口原型

```
Future<int> Uninit()
```

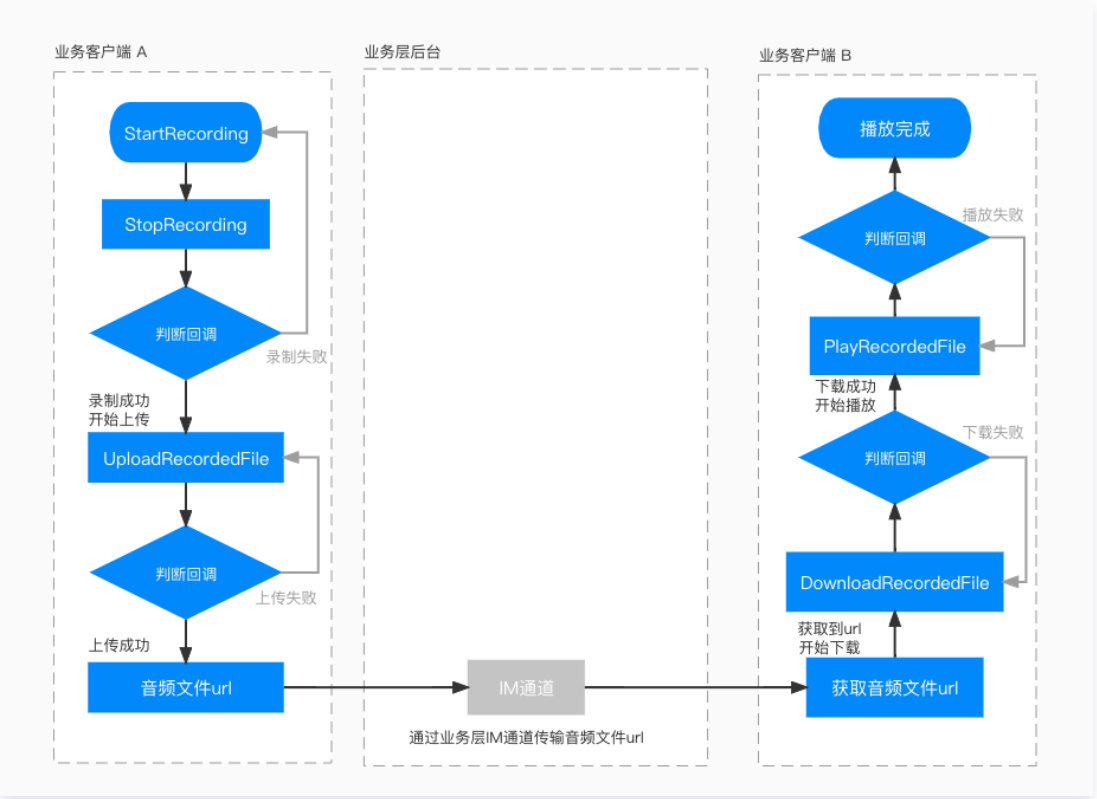
语音消息服务及转文本服务

说明

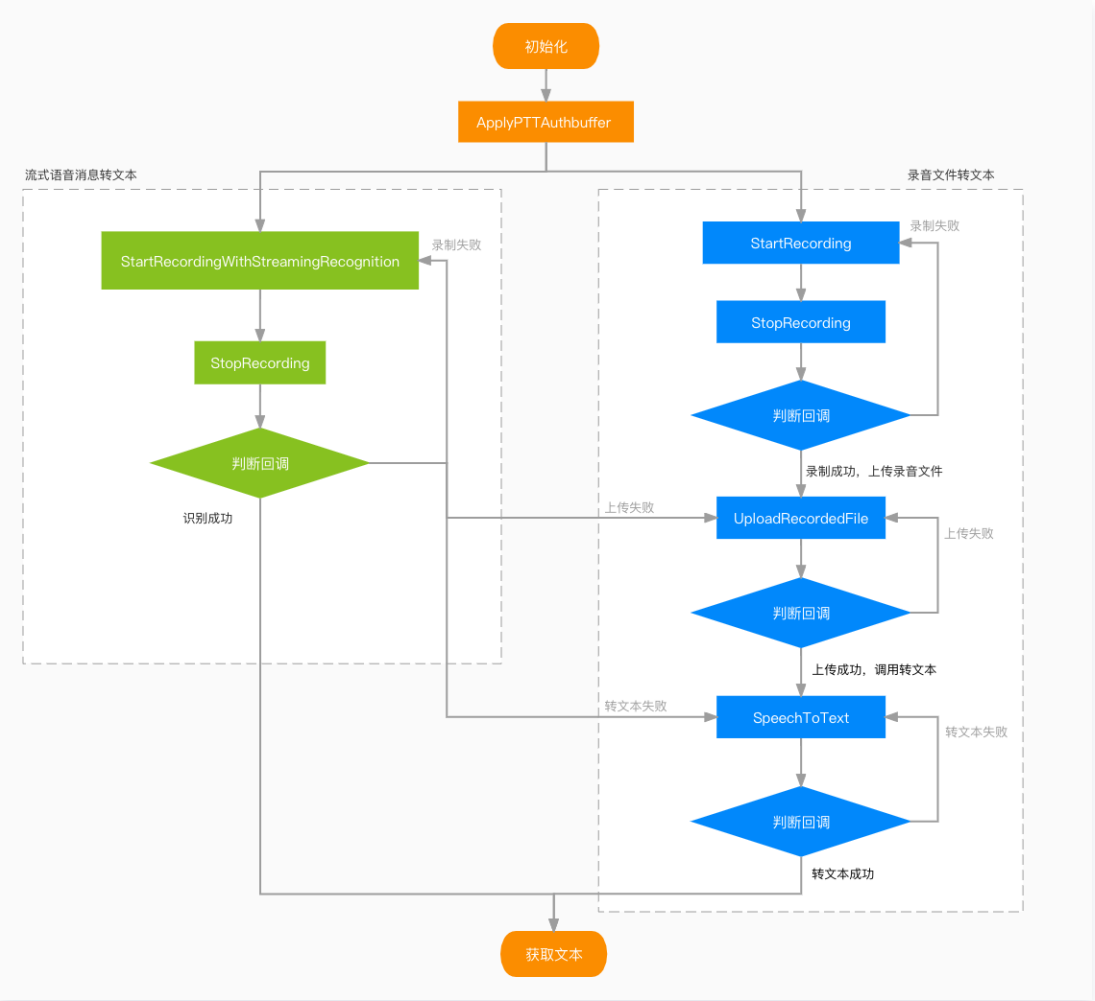
- 转文本服务分录音文件极速转文本以及语音消息流式转文本。
- 使用语音消息服务不需要进入实时语音房间。

- 语音消息最大录制时长默认为58秒，最短不能小于1秒。如果需要再加以限制，例如限制为最大录制时长为10秒，请在初始化之后调用 SetMaxMessageLength 接口进行设置。

语音消息服务使用流程



转文本服务使用流程



接口	接口含义
GenAuthBuffer	获取鉴权信息
SetMaxMessageLength	限制最大语音信息时长

生成本地鉴权

生成 AuthBuffer，用于相关功能的加密和鉴权，如正式发布请使用后台部署密钥，后台部署请参见 鉴权密钥 。

接口原型

```
Future<Uint8List> GenAuthBuffer(String appID, String roomID, String openID, String key)
```

参数	类型	含义
appId	string	来自腾讯云控制台的 AppId 号码。
roomId	string	填 null 或者空字符串。
openId	string	用户标识。与 Init 时候的 OpenId 相同。
key	string	来自腾讯云 控制台 的权限密钥。

应用鉴权

生成鉴权信息后，将鉴权赋值到 SDK 中。

接口原型

```
Future<int> ApplyPTTAuthbuffer(Uint8List authBuffer)
```

示例代码

```
Uint8List authBuffer = await ITMGContext.GetInstance().GenAuthBuffer(_editAppID.text,
_editRoomID.text, _editOpenID.text, _editKey.text);
m_context.ApplyPTTAuthbuffer(authBuffer);
```

限制最大语音信息时长

限制最大语音消息的长度，最大支持58秒。

接口原型

```
Future<int> SetMaxMessageLength(int msTime)
```

参数	类型	含义
msTime	number	语音时长，单位 ms，区间为 1000 < msTime < = 58000

示例代码

```
ITMGContext.GetInstance().GetPTT().SetMaxMessageLength(fileLen);
```

流式语音识别

语音消息及转文字相关接口

接口	接口含义
StartRecordingWithStreamingRecognition	启动流式录音
StopRecording	停止录音

启动流式语音识别

此接口用于启动流式语音识别，同时在回调中会有实时的语音转文字返回，可以指定语言进行识别，也可以将语音中识别到的信息翻译成指定的语言返回。停止录音调用 停止录制 。

接口原型

```
Future<int> StartRecordingWithStreamingRecognition(String filePath, String speechLanguage,
String translateLanguage)
```

参数	类型	含义
filePath	string	存放的语音路径
speechLanguage	string	识别成指定文字的语言参数，参数请参见 语言参数参考列表

translateLanguage	string	翻译成指定文字的语言参数，参数请参见 语言参数参考列表

示例代码

```
string filePath = "xx/xxx/xxx.silk"
int res = await
ITMGContext.GetInstance().GetPTT().StartRecordingWithStreamingRecognition(filePath,
strCurLanguage, strCurLanguage);
if (ret == 0) {
    this.currentStatus = "开始流式录音";
} else {
    this.currentStatus = "开始流式录音失败";
}
```

> **注意**
> 翻译会收取额外费用，请参见 购买指南 。

流式语音识别的回调

启动流式语音识别后，需要在 OnEvent 通知中监听回调消息，事件消息分为以下两个：
ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_COMPLETE 是在停止录制并完成识别后才返回文字，相当于一段话说完才会返回识别的文字。
ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_IS_RUNNING 是在录音过程中就会实时返回识别到的文字，相当于边说话边返回识别到的文字。
根据需求在 回调 通知中对相应事件消息进行判断。传递的参数包含以下四个信息。

消息名称	含义
result	用于判断流式语音识别是否成功的返回码
text	语音转文字识别的文本
file_path	录音存放的本地地址
file_id	录音在后台的 url 地址，录音在服务器存放90天

> **注意**
> 监听 `ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_IS_RUNNING` 消息时，file_id 为空。

错误码

错误码	含义	处理方式
32775	流式语音转文本失败，但是录音成功	调用 UploadRecordedFile 接口上传录音，再调用 SpeechToText 接口进行语音转文字操作
32777	流式语音转文本失败，但是录音成功，上传成功	返回的信息中有上传成功的后台 url 地址，调用 SpeechToText 接口进行语音转文字操作
32786	流式语音转文本失败	在流式录制状态当中，请等待流式录制接口执行结果返回
32787	转文本成功，文本翻译服务未开通	需要在控制台开通文本翻译服务

32788	转文本成功，文本翻译语言参数不支持	重新检查传入参数

如果出现 4098 错误码，请参见 常见问题文档 进行解决。

示例代码

```
void handleEventMsg(int eventType, String data){
   switch (eventType) {
        case ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_COMPLETE:
        {
            HandleSTREAM2TEXTComplete(data,true);
            break;
        }
        ...
        case ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVNET_TYPE_PTT_STREAMINGRECOGNITION_IS_RUNNING:
        {
            HandleSTREAM2TEXTComplete(data, false);
            break;
        }
    }
}
ITMGContext.GetInstance().SetEvent(handleEventMsg);
```

语音消息录制

录制的流程为：录音 > 停止录音 > 录音回调返回 > 启动下一次录音。

语音消息及转文字相关接口

接口	接口含义
StartRecording	启动录音
PauseRecording	暂停录音
ResumeRecording	恢复录音
StopRecording	停止录音
CancelRecording	取消录音

启动录音

此接口用于启动录音。

接口原型

```
Future<int> StartRecording(String filePath)
```

参数	类型	含义
filePath	string	存放的语音路径

示例代码

```
string filepath = "xxxx/xxx.silk";
int res = await ITMGContext.GetInstance().GetPTT().StartRecording(filepath);
```

停止录音

此接口用于停止录音。此接口为异步接口，停止录音后会有录音完成回调，成功之后录音文件才可用。

接口原型

```
Future<int> StopRecording()
```

示例代码

```
ITMGContext.GetInstance().GetPTT().StopRecording();
```

启动录音的回调

录音完成的回调，通过委托传递消息。
停止录音调用StopRecording。停止录音后才有启动录音的回调。

参数	类型	含义
code	string	当 code 为 0 时，录制完成
filepath	string	录制的存放地址，必须是可以访问到的路径，不可将 fileid 作为路径

错误码

错误码值	原因	建议方案
4097	参数为空	检查代码中接口参数是否正确
4098	初始化错误	检查设备是否被占用，或者权限是否正常，是否初始化正常
4099	正在录制中	确保在正确的时机使用 SDK 录制功能
4100	没有采集到音频数据	检查麦克风设备是否正常
4101	录音时，录制文件访问错误	确保文件存在，文件路径的合法性
4102	麦克风未授权错误	使用 SDK 需要麦克风权限，添加权限请参考对应引擎或平台的 SDK 工程配置文档
4103	录音时间太短错误	首先，限制录音时长的单位为毫秒，检查参数是否正确；其次，录音时长要1000毫秒以上才能成功录制
4104	没有启动录音操作	检查是否已经调用启动录音接口

示例代码

```
void handleEventMsg(int eventType, String data){
   switch (eventType) {
        case ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVENT_TYPE_ENTER_ROOM:
```

```
        {
          //进行处理
          break;
        }
        ...
        case ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVNET_TYPE_PTT_RECORD_COMPLETE:
        {
          //进行处理
          break;
        }
    }
}
ITMGContext.GetInstance().SetEvent(handleEventMsg);
```

暂停录音

此接口用于暂停录音。如需恢复录音请调用接口 ResumeRecording。

接口原型

```
Future<int> PauseRecording()
```

示例代码

```
ITMGContext.GetInstance().GetPTT().PauseRecording();
```

恢复录音

此接口用于恢复录音。

接口原型

```
Future<int> ResumeRecording()
```

示例代码

```
ITMGContext.GetInstance().GetPTT().ResumeRecording();
```

取消录音

调用此接口取消录音。**取消之后没有回调**。

接口原型

```
Future<int> CancelRecording()
```

示例代码

```
ITMGContext.GetInstance().GetPTT().CancelRecording();
```

语音消息上传、下载及播放

接口	接口含义
UploadRecordedFile	上传语音文件
DownloadRecordedFile	下载语音文件
PlayRecordedFile	播放语音
StopPlayFile	停止播放语音
GetFileSize	语音文件的大小
GetVoiceFileDuration	语音文件的时长

上传语音文件

此接口用于上传语音文件。

接口原型

```
Future<int> UploadRecordedFile(String filePath)
```

参数	类型	含义
filePath	String	上传的语音路径，此路径为本地路径

示例代码

```
ITMGContext.GetInstance().GetPTT().UploadRecordedFile(_filePath);
```

上传语音完成的回调

上传语音完成后，事件消息为 ITMG_MAIN_EVNET_TYPE_PTT_UPLOAD_COMPLETE，在 OnEvent 函数中对事件消息进行判断。传递的参数包含三个信息，result，file_path 和 file_id。

参数	类型	含义
result	number	当 code 为0时，录制完成
filepath	string	录制的存放地址
fileid	string	文件的 url 路径

错误码

错误码值	原因	建议方案
8193	上传文件时，文件访问错误	确保文件存在，文件路径的合法性
8194	签名校验失败错误	检查鉴权密钥是否正确，检查是否有初始化离线语音
8195	网络错误	检查设备网络是否可以正常访问外网环境
8196	获取上传参数过程中网络失败	检查鉴权是否正确，检查设备网络是否可以正常访问外网环境
8197	获取上传参数过程中回包数据为空	检查鉴权是否正确，检查设备网络是否可以正常访问外网环境
8198	获取上传参数过程中回包解包失败	检查鉴权是否正确，检查设备网络是否可以正常访问外网环境

版权所有：腾讯云计算（北京）有限责任公司

8200	没有设置 appinfo	检查 apply 接口是否有调用，或者入参是否为空

示例代码

```
void handleEventMsg(int eventType, String data){
 switch (eventType) {
        ...
        case ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVNET_TYPE_PTT_UPLOAD_COMPLETE:
        {
            //进行处理
            break;
        }
    }
}
ITMGContext.GetInstance().SetEvent(handleEventMsg);
```

下载语音文件

此接口用于下载语音文件。

接口原型

```
Future<int> DownloadRecordedFile(String fileId, String filePath)
```

参数	类型	含义
fileId	string	文件的 url 路径
filePath	string	文件的本地保存路径，必须是可以访问到的路径，不可将 fileid 作为路径

示例代码

```
ITMGContext.GetInstance().GetPTT().DownloadRecordedFile(_fileId, _filePath);
```

下载语音文件完成回调

下载语音完成后，事件消息为 ITMG_MAIN_EVNET_TYPE_PTT_DOWNLOAD_COMPLETE， 在 OnEvent 函数中对事件消息进行判断。
传递的参数包含三个信息，result、file_path 和 file_id。

参数	类型	含义
result	number	当 code 为0时，下载完成
filepath	string	录制的存放地址
fileid	string	文件的 url 路径，录音在服务器存放 90 天

错误码

错误码值	原因	建议方案
12289	下载文件时，文件访问错误	检查文件路径是否合法

12290	签名校验失败	检查鉴权密钥是否正确，检查是否有初始化离线语音
12291	网络存储系统异常	服务器获取语音文件失败，检查接口参数 fileid 是否正确，检查网络是否正常，检查 COS 文件存不存在
12292	服务器文件系统错误	检查设备网络是否可以正常访问外网环境，检查服务器上是否有此文件
12293	获取下载参数过程中，HTTP 网络失败	检查设备网络是否可以正常访问外网环境
12294	获取下载参数过程中，回包数据为空	检查设备网络是否可以正常访问外网环境
12295	获取下载参数过程中，回包解包失败	检查设备网络是否可以正常访问外网环境
12297	没有设置 appinfo	检查鉴权密钥是否正确，检查是否有初始化离线语音

示例代码

```
void handleEventMsg(int eventType, String data){
    switch (eventType) {
        ...
        case ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVENT_TYPE_ENTER_ROOM:
        {
          //进行处理
          break;
        }
    }
}
ITMGContext.GetInstance().SetEvent(handleEventMsg);
```

播放语音

此接口用于播放语音。

接口原型

```
Future<int> PlayRecordedFile(String filePath, int voiceType)
```

参数	类型	含义
filePath	string	本地语音文件的路径
voicetype	ITMG_VOICE_TYPE	变声类型，请参见 变声接入文档

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

示例代码

```
int res = await ITMGContext.GetInstance().GetPTT().PlayRecordedFile(_filePath, _nVoiceType);
```

播放语音的回调

播放语音的回调，事件消息为 ITMG_MAIN_EVNET_TYPE_PTT_PLAY_COMPLETE， 在 OnEvent 函数中对事件消息进行判断。传递的参数包含两个信息，一个是 result，另一个是 file_path。

参数	类型	含义
code	number	当 code 为0时，播放完成
filepath	string	录制的存放地址

错误码

错误码值	原因	建议方案
20481	初始化错误	检查设备是否被占用，或者权限是否正常，是否初始化正常
20482	正在播放中，试图打断并播放下一个失败了（正常是可以打断的）	检查代码逻辑是否正确
20483	参数为空	检查代码中接口参数是否正确
20484	内部错误	初始化播放器错误，解码失败等问题产生此错误码，需要结合日志定位问题

示例代码

```
void handleEventMsg(int eventType, String data){
 switch (eventType) {
    case ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVNET_TYPE_PTT_PLAY_COMPLETE:
    {
        //进行处理
        break;
    }
   }
}
```

停止播放语音

此接口用于停止播放语音。停止播放语音也会有播放完成的回调。

接口原型

```
Future<int> StopPlayFile()
```

示例代码

```
ITMGContext.GetInstance().GetPTT().StopPlayFile();
```

获取语音文件的大小

通过此接口，获取语音文件的大小。

接口原型

```
Future<int> GetFileSize(String filePath)
```

参数	类型	含义
filePath	string	语音文件的路径，此路径为本地路径

示例代码

```
final int res = await ITMGContext.GetInstance().GetPTT().GetFileSize(_filePath);
```

获取语音文件的时长

此接口用于获取语音文件的时长，单位毫秒。

接口原型

```
Future<int> GetVoiceFileDuration(String filePath)
```

参数	类型	含义
filePath	string	语音文件的路径，此路径为本地路径

示例代码

```
final int res = await ITMGContext.GetInstance().GetPTT().GetVoiceFileDuration(_filePath);
```

录音文件极速转文

将指定的语音文件翻译成文字（指定语言）

此接口可以指定语言进行识别，也可以将语音中识别到的信息翻译成指定的语言返回。

> ⚠ **注意**
> 翻译会收取额外费用，请参见 购买指南 。

接口原型

```
Future<int> SpeechToText(String fileId, String speechLanguage, String translateLanguage)
```

参数	类型	含义
fileID	string	语音文件 url，录音在服务器存放90天
speechLanguage	string	识别出指定文字的语言参数，参数参见 语言参数参考列表
translatelanguage	string	翻译成指定文字的语言参数，参数参见 语言参数参考列表 中的翻译语言参数

示例代码

```
ITMGContext.GetInstance().GetPTT().SpeechToText(_fileId, "cmn-Hans-CN", "cmn-Hans-CN");
```

识别回调

将指定的语音文件识别成文字的回调，事件消息为 ITMG_MAIN_EVNET_TYPE_PTT_SPEECH2TEXT_COMPLETE， 在 OnEvent 函数中对事件消息进行判断。
传递的参数包含三个信息，result、file_path 和 text，其中 text 为识别的文本。

参数	类型	含义
result	number	当 code 为0时，录制完成
fileid	string	语音文件 url，录音在服务器存放90天
text	string	转换的文本结果

错误码

错误码值	原因	建议方案
32769	内部错误	分析日志，获取后台返回给客户端的真正错误码，并联系后台同事协助解决
32770	网络失败	检查设备网络是否可以正常访问外网环境
32772	回包解包失败	分析日志，获取后台返回给客户端的真正错误码，并联系后台同事协助解决
32774	没有设置 appinfo	检查鉴权密钥是否正确，检查是否有初始化离线语音
32776	authbuffer 校验失败	检查 authbuffer 是否正确
32784	语音转文本参数错误	检查代码中接口参数 fileid 是否为空
32785	语音转文本翻译返回错误	离线语音后台错误，请分析日志，获取后台返回给客户端的真正错误码，并联系后台同事协助解决
32787	转文本成功，文本翻译服务未开通	需要在控制台开通文本翻译服务
32788	转文本成功，文本翻译语言参数不支持	重新检查传入参数

示例代码

```
void handleEventMsg(int eventType, String data){
 switch (eventType) {
    case ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVENT_TYPE_ENTER_ROOM:
       {
         //进行处理
         break;
       }
       ...
       case ITMG_MAIN_EVENT_TYPE.ITMG_MAIN_EVNET_TYPE_PTT_SPEECH2TEXT_COMPLETE:
       {
         //进行处理
         break;
       }
    }
}
```

语音消息音量相关接口

接口	接口含义
GetMicLevel	获取实时麦克风音量
SetMicVolume	设置录制音量
GetMicVolume	获取录制音量
GetSpeakerLevel	获取实时扬声器音量
SetSpeakerVolume	设置播放音量
GetSpeakerVolume	获取播放音量

获取语音消息麦克风实时音量

此接口用于获取麦克风实时音量，返回值为 number 类型，值域为0 – 200。

接口原型

```
Future<int> GetMicLevel()
```

示例代码

```
final int res = await ITMGContext.GetInstance().GetPTT().GetMicLevel();
```

设置语音消息录制音量

此接口用于设置离线语音录制音量，值域为0 – 200。

接口原型

```
Future<int> SetMicVolume(int volume)
```

参数	类型	含义
vol	number	取值范围为 0–200，数值为0的时候表示静音，当数值为100的时候表示音量不增不减，默认数值为100。

示例代码

```
final int res = await ITMGContext.GetInstance().GetPTT().SetMicVolume(100);
```

获取语音消息录制音量

此接口用于获取离线语音录制音量。返回值为 number 类型，值域为0 – 200。

接口原型

```
Future<int> GetMicVolume()
```

示例代码

```
final int res = await ITMGContext.GetInstance().GetPTT().GetMicVolume();
```

获取语音消息扬声器实时音量

此接口用于获取扬声器实时音量。返回值为 number 类型，值域为0 - 200。

接口原型

```
Future<int> GetSpeakerLevel()
```

示例代码

```
final int res = await ITMGContext.GetInstance().GetPTT().GetSpeakerLevel();
```

设置语音消息播放音量

此接口用于设置离线语音播放音量，值域为0 - 200。

接口原型

```
Future<int> SetSpeakerVolume(int volume)
```

示例代码

```
final int res = await ITMGContext.GetInstance().GetPTT().SetSpeakerVolume(100);
```

获取语音消息播放音量

此接口用于获取离线语音播放音量。返回值为 number 类型，值域为0 - 200。

接口原型

```
Future<int> GetSpeakerVolume()
```

示例代码

```
final int res = await ITMGContext.GetInstance().GetPTT().GetSpeakerVolume();
```

高级 API

获取版本号

获取 SDK 版本号，用于分析。

接口原型

```
Future<String> GetSDKVersion()
```

示例代码

```
_sdkVersions = await ITMGContext.GetInstance().GetSDKVersion();
```

设置打印日志等级

用于设置打印日志等级。建议保持默认等级。需要在 Init 之前调用。

接口原型

```
Future<int> SetLogLevel(int levelWrite, int levelPrint)
```

参数含义

参数	类型	含义
level	ITMG_LOG_LEVEL	设置写入日志的等级，TMG_LOG_LEVEL_NONE 表示不写入，默认为 TMG_LOG_LEVEL_INFO

level 说明如下：

level	含义
TMG_LOG_LEVEL_NONE	不打印日志
TMG_LOG_LEVEL_ERROR	打印错误日志（默认）
TMG_LOG_LEVEL_INFO	打印提示日志
TMG_LOG_LEVEL_DEBUG	打印开发调试日志
TMG_LOG_LEVEL_VERBOSE	打印高频日志

示例代码

```
ITMGContext.GetInstance().SetLogLevel(ITMG_LOG_LEVEL.TMG_LOG_LEVEL_ERROR,ITMG_LOG_LEVEL.TMG_LOG_
LEVEL_ERROR);
```

设置打印日志路径

用于设置打印日志路径。默认路径如下。需要在 Init 之前调用。

接口原型

```
Future<int> SetLogPath(String logDir)
```

参数	类型	含义
logPath	string	路径

示例代码

```
String logDir = ""//自行设置路径
ITMGContext.GetInstance().SetLogPath(curPath);
```

SDK 版本升级指引

最近更新时间：2023-03-20 15:36:33

本文主要介绍适用于开发的接口升级技术文档。方便您调试和接入腾讯云游戏多媒体引擎 API。

前提条件

- 已经开通并使用 GME 服务，需要升级到最新版本。
- 如无法判断是否需要升级到最新版本，可以 提交工单 联系开发者。

GME 2.x 升级 GME 2.9

SDK 更新动态

- 动态库拆分。
- 修改 Android 包名。

GME SDK 更新后，除了 libgmesdk 外，新增以下库文件。

库文件对应功能

新版本的GME拆解了动态库，尽可能降低包体大小。可以根据自己的需求只引入相应的库文件。例如只需要变声功能，则只需引入 libgmesoundtouch。

库文件	对应功能
libgmefdkaac	1. 用于进入标准、高清音质房间 2. 用于播放 acc 格式伴奏文件
libgmefaad2	用于播放 mp4 格式伴奏文件
libgmeogg	用于播放 ogg 格式伴奏文件
libgmelamemp3	用于播放 mp3 格式伴奏文件
libgmesoundtouch	用于变声、变调功能

升级须知

iOS 端升级，需要参考 iOS 工程升级指引 进行相关配置。
Android 端升级，需要修改包名，将 Tencent 改为 GME，混淆配置也需要修改，参考 Android工程导出文档 。
Unity 端升级，如果使用了标准音质或者高清音质，或者使用了伴奏，需要参考 Unity使用高清音质文档 进行解决。

GME 2.2 升级 GME 2.3.5

SDK 更新动态

新增功能

- 支持实时语音过程中使用离线语音。
- 支持实时语音过滤，可识别可能令人反感、不安全或不适宜内容。
- 支持 H5 实时语音，实现全平台实时语音互通。
- 新增 Android v8a 架构支持。
- Android 低延时采集播放适配。

优化能力

- 优化 SDK 的范围语音功能接口，降低接入门槛。
- 语音降噪效果优化。
- 大幅降低 SDK 内存消耗。

主要接口变更

EnterRoom

进房操作由同步改为异步调用，返回值为0时的，表示异步投递成功，等待回调函数进行处理，若返回值不为0时，则表示异步投递失败。

```
public abstract int EnterRoom();
```

ExitRoom

退房操作由同步改为异步调用，参照 RoomExitComplete 回调函数处理，返回值为 AV_OK 时，表示异步投递成功。

> **注意**
> 如果应用中有退房后立即进房的场景，在接口调用流程上，开发者无需要等待 ExitRoom 的回调 RoomExitComplete 通知，只需直接调用接口。

```
public abstract int ExitRoom();
```

错误码变更

- 如需对所有错误码统一处理，请使用 !AV_OK。
- 如需单独处理每一类错误，请关注接口返回的错误类型。

> **说明**
> 错误码“1”没有明确含义，且2.3.5以后版本不再返回，故删除。

其他接口变更

PauseAudio/ResumeAudio

```
public int PauseAudio()
public int ResumeAudio()
```

在2.3之前版本的 SDK 接口调用过程中，如有调用 ITMGAudioCtrl::PauseAudio/ResumeAudio 两个接口，可参照下表进行版本对比。

2.3 之前版本	升级 2.3 版本
使用目的为与其他模块互斥	将 PauseAudio 改为 Pause，将 ResumeAudio 改为 Resume
使用目的为在实时语音中使用离线语音	请将 PauseAudio 及 ResumeAudio 删除

SetLogLevel 接口参数变更

原接口

```
ITMGContext virtual void SetLogLevel(int logLevel, bool enableWrite, bool enablePrint)
```

新接口

```
ITMGContext virtual void SetLogLevel(ITMG_LOG_LEVEL levelWrite, ITMG_LOG_LEVEL levelPrint)
```

参数说明

参数	类型	说明
levelWrite	ITMG_LOG_LEVEL	设置写入日志的等级，TMG_LOG_LEVEL_NONE 表示不写入
levelPrint	ITMG_LOG_LEVEL	设置打印日志的等级，TMG_LOG_LEVEL_NONE 表示不打印

ITMG_LOG_LEVEL 类型

ITMG_LOG_LEVEL	说明
TMG_LOG_LEVEL_NONE=0	不打印日志
TMG_LOG_LEVEL_ERROR=1	打印错误日志（默认）
TMG_LOG_LEVEL_INFO=2	打印提示日志
TMG_LOG_LEVEL_DEBUG=3	打印开发调试日志
TMG_LOG_LEVEL_VERBOSE=4	打印高频日志

GME 2.3.5 升级 GME 2.5.1

增加接口

GetSendStreamLevel

此接口用于获取音频上行实时音量，返回值为 int 类型，取值范围为0到100。

```
ITMGContextGetInstance()->GetAudioCtrl()->GetSendStreamLevel();
```

GetRecvStreamLevel

此接口用于获取房间内其他成员下行实时音量，返回值为 int 类型，取值范围为0到100。

```
iter->second.level = ITMGContextGetInstance()->GetAudioCtrl()->GetRecvStreamLevel(iter-
>second.openid.c_str());
```

接口变更

语音消息及转文字接口变更返回值

以下接口返回值修改为 int 类型。

```
StartRecording
UploadRecordedFile
DownloadRecordedFile
PlayRecordedFile
SpeechToText
```

GME 2.5 升级 GME 2.7

增加接口

PlayRecordedFile(const char* filePath, ITMG_VOICE_TYPE voiceType)

此接口用于播放带有变声效果的语音消息。

SetAccompanyKey(int nKey)

此接口用于设置实时语音伴奏升降调。

编译工具链文档

最近更新时间：2023-11-27 11:11:21

本文主要介绍适用于开发的编译工具链文档，方便您调试和接入腾讯云游戏多媒体引擎 API。

2.9.10 版本

平台	说明
Win32	• Windows Software Development Kit (Visual Studio 2015): 10.0.14393.0 • Platform Toolset: v140_xp
macOS	• MacOS Minimum Deployment Target: 10.10 • macOS SDK: 12.0 • Xcode®: 13.1 • Target Architectures: x86_64
iOS	• iOS Minimum Deployment Target: 9.0 • iOS SDK: 15.0 • Xcode®: 13.1 • Target Architectures: arm64, armv7, arm64(simulator), x86_64(simulator)
Android	• Android SDK: Minimum API 16 • Android SDK Tools: 30.0.3 • Android NDK: r23
PS4	SDK: SDK 6.000
PS5	SDK: SDK 3.000
Switch	SDK: DevEnv8.3.0
XboxOne	Microsoft® XDK: 10.0.17134.5055 (July 2018 qfe7)

错误码

最近更新时间：2023-03-17 14:16:30

为方便开发者调试和接入腾讯云游戏多媒体引擎产品 API，这里向您介绍适用于游戏多媒体引擎开发的错误码文档。

通用错误码

错误码名称	错误码值	原因及建议方案
AV_ERR_NET_REQUEST_FALLED	7004	网络请求失败，一般由网络状态不稳定引起，可参见 实时语音房间问题 进行排查
AV_ERR_CHARGE_OVERDUE	7005	账号欠费导致失败，需要在腾讯云控制台上查看是否欠费
AV_ERR_AUTH_FIALD	7006	鉴权失败 有以下几个原因： 1. AppID 不存在或者错误。 2. authbuff 鉴权错误。 3. 鉴权过期。
AV_ERR_REPEATED_OPERATION	1001	已经在进行某种操作，再次去做同样的操作，则会产生这个错误。操作类别主要有：AVContext 类、房间类、设备类、成员类。AVContext 类型的操作：StartContext/StopContext。房间类型的操作：EnterRoom/ExitRoom。设备类型的操作：打开/关闭某个设备。建议等待上一个操作完成后再进行下一个操作
AV_ERR_EXCLUSIVE_OPERATION	1002	已经在进行某种操作，再次去做同类型的其他操作，则会产生这个错误。请等待上一个操作完成后再进行下一个操作
AV_ERR_HAS_IN_THE_STATE	1003	对象已经处于某种状态，再去做使得它进入这种状态的操作时，则会产生这个错误。如已经在房间中，再去做进入房间的操作，就会产生这个错误，由于已经处于所要状态，可以认为该操作已经成功，当作成功来处理
AV_ERR_INVALID_ARGUMENT	1004	调用 SDK 接口时，传入错误的参数，则会产生这个错误。如进入房间时，传入的房间类型不等于 AVRoom::ROOM_TYPE_PAIR 或 AVRoom::ROOM_TYPE_MULTI，就会产生这个错误。请详细阅读 API 文档，获取每个接口的每个参数的有效取值范围，保证传入参数的正确性并进行相应的预防处理。
AV_ERR_TIMEOUT	1005	进行某个操作，在规定的时间内，还未返回操作结果，则会产生这个错误。多数情况下，涉及到信令传输的、且网络出问题的情况下，才容易产生这个错误。如执行进入房间操作时，30s后还没有返回进入房间操作完成的结果的话，就会产生这个错误。请确认网络是否有问题，是否能连接到外网环境，并尝试重试。
AV_ERR_NOT_IMPLEMENTED	1006	调用 SDK 接口时，如果相应的功能还未支持，则会产生这个错误。
AV_ERR_NOT_IN_MAIN_THREAD	1007	SDK 对外接口要求在主线程执行，如果业务侧调用 SDK 接口时，没有在主线程调用，则会产生这个错误。请修改业务侧逻辑，确保在主线程调用 SDK 接口。
AV_ERR_CONTEXT	1101	当 AVContext 对象未处于 CONTEXT_STATE_STARTED 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。修改业务侧逻辑，确保调用 SDK 接口时机的正确性。

版权所有：腾讯云计算（北京）有限责任公司

_NOT_EXIST		
AV_ERR_CONTEXT_NOT_STOPPED	1102	当 AVContext 对象未处于 CONTEXT_STATE_STOPPED 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。如不在这种状态下，去调用 AVContext::DestroyContext 时，就会产生这个错误。修改业务侧逻辑，确保调用 SDK 接口的时机的正确性。

实时语音功能错误码

错误码名称	错误码值	原因及建议方案
AV_ERR_IN_OTHER_ROOM	7007	已经在其它房间，需要退房后成功后再进行操作。
AV_ERR_ROOM_NOT_EXIST	1201	未进房的状态下，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。请确保调用 SDK 接口的时机的正确性。
AV_ERR_ROOM_NOT_EXITED	1202	未退房的状态下，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。如不在这种状态下，去调用 AVContext::StopContext 时，就会产生这个错误。请确保调用 SDK 接口的时机的正确性。
AV_ERR_DEVICE_NOT_EXIST	1301	当设备不存在或者设备初始化未完成时，去使用设备，则会产生这个错误。请确认设备是否真的存在，确保设备 ID 填写的正确性，确保在设备初始化成功后再去使用设备。
AV_ERR_SERVER_FAILED	10001	进房出现未知错误： 1. 请查看并确认进房 API 中的参数，如 AppID， UIN， AuthBuffer 的合法性（参照文档）。 2. 请查看控制台上的相关参数是否与本地的一致。 3. 请查看控制台是否欠费。 4. 检查开发者测试设备网络环境，是在开发者内网环境还是外网环境。
AV_ERR_SERVER_NO_PERMISSION	10003	被剔除出实时语音房间
AV_ERR_SERVER_REQUEST_ROOM_ADDRESS_FAIL	10004	进入实时语音房间失败，建议参见 开发问题 收集日志，联系 GME 开发者协助进行分析错误，解决问题。
AV_ERR_SERVER_CONNECT_ROOM_FAIL	10005	进入实时语音房间失败，建议参见 开发问题 收集日志，联系 GME 开发者协助进行分析错误，解决问题。

语音消息错误码

录制音频相关错误码

错误码名称	错误码值	原因及建议方案
VOICE_RECORDER_PARAM_NULL	4097	参数为空，请检查代码中接口参数是否正确
VOICE_RECORDER_INIT_ERROR	4098	初始化错误： 1. 请检查设备是否被占用。 2. 相关权限是否正常。 3. 是否初始化正常。

VOICE_RECORDER_RECORDING_ERROR	4099	调用录制接口时，正在录制状态中。请确保在正确的时机使用 SDK 录制功能。
VOICE_RECORDER_NO_AUDIO_DATA_WARN	4100	没有采集到音频数据，请检查麦克风设备是否正常。
VOICE_RECORDER_OPENFILE_ERROR	4101	录音时，录制文件访问错误， 1. 请确保文件存在。 2. 请检查文件路径的合法性。
VOICE_RECORDER_MIC_PERMISSION_ERROR	4102	麦克风未授权错误。使用 SDK 需要麦克风权限，添加权限请参考对应引擎或平台的 SDK 工程配置文档。
VOICE_RECORD_AUDIO_TOO_SHORT	4103	录音时间太短： 1. 限制录音时长的单位为毫秒，请检查参数是否正确。 2. 录音时长要1000毫秒以上才能成功录制。
VOICE_RECORD_NOT_START	4104	没有启动录音操作，需要检查是否已经调用启动录音接口。

上传音频相关错误码

错误码名称	错误码值	原因及建议方案
VOICE_UPLOAD_FILE_ACCESSERROR	8193	上传文件时，文件访问错误，1、请确保文件存在，2、请检查文件路径的合法性。
VOICE_UPLOAD_SIGN_CHECK_FAIL	8194	签名校验失败错误： 1. 请检查鉴权密钥是否正确。 2. 请检查是否有初始化离线语音。
VOICE_UPLOAD_COS_INTERNAL_FAIL	8195	网络错误导致上传cos失败，检查设备网络是否可以正常访问外网环境，参见 如何检查网络 。
VOICE_UPLOAD_GET_TOKEN_NETWORK_FAIL	8196	上传文件过程中失败： 1. 请检查鉴权是否正确。 2. 请检查设备网络是否可以正常访问外网环境，参见 如何检查网络。
VOICE_UPLOAD_SYSTEM_INNER_ERROR	8197	上传文件过程中失败： 1. 请检查鉴权是否正确。 2. 请检查设备网络是否可以正常访问外网环境，参见 如何检查网络。
VOICE_UPLOAD_RSP_DATA_DECODE_FAIL	8198	上传文件过程中失败： 1. 请检查鉴权是否正确。 2. 请检查设备网络是否可以正常访问外网环境，参见 如何检查网络。
VOICE_UPLOAD_APPINFO_UNSET	8200	鉴权未设置： 1. 请检查 applyAuthBuffer 接口是否有调用。 2. 请检查 applyAuthBuffer 接口入参是否为空

下载音频相关错误码

错误码名称	错误码值	原因及建议方案
VOICE_DOWNLOAD_FILE_ACCESS_ERRO	1228	下载文件时，路径访问出错，请检查文件路径的合法性。

R	9	
VOICE_DOWNLOAD_SIGN_CHECK_FAIL	12290	签名校验失败： 1. 请检查鉴权密钥是否正确。 2. 请检查是否有初始化离线语音。
VOICE_DOWNLOAD_COS_INTERNAL_FAIL	12291	获取语音文件失败： 1. 请检查接口参数 fileid 是否正确。 2. 请检查网络是否正常。
VOICE_DOWNLOAD_REMOTEFILE_ACCESS_ERROR	12292	获取语音文件失败： 1. 请检查接口参数 fileid 是否正确。 2. 请检查网络是否正常。
VOICE_DOWNLOAD_GET_SIGN_NETWORK_FAIL	12293	网络错误导致获取语音文件，检查设备网络是否可以正常访问外网环境，参见 如何检查网络 。
VOICE_DOWNLOAD_SYSTEM_INNER_ERROR	12294	网络错误导致获取语音文件，检查设备网络是否可以正常访问外网环境，参见 如何检查网络 。
VOICE_DOWNLOAD_GET_SIGN_RSP_DATA_DECODE_FAIL	12295	网络错误导致获取语音文件，检查设备网络是否可以正常访问外网环境，参见 如何检查网络 。
VOICE_DOWNLOAD_APPINFO_UNSET	12297	鉴权未设置： 1. 请检查 applyAuthBuffer 接口是否有调用。 2. 请检查 applyAuthBuffer 接口入参是否为空。

播放音频相关错误码

错误码名称	错误码值	原因及建议方案
VOICE_PLAY_INIT_ERROR	20481	初始化错误： 1. 请检查设备是否被占用。 2. 相关权限是否正常。 3. 是否初始化正常。
VOICE_PLAY_PLAYING_ERROR	20482	正在播放中，试图打断并播放下一个失败了，可以先暂停播放后再继续播放音频文件。
VOICE_PLAY_PARAM_NULL	20483	参数为空导致出错，请检查代码中接口参数是否正确。
VOICE_PLAY_OPEN_FILE_ERROR	20484	打开音频文件失败： 1. 请确保文件存在。 2. 请检查文件路径的合法性。
VOICE_PLAY_NOT_START	20485	音频文件无法开始播放： 1. 请确保文件存在。 2. 请检查文件路径的合法性。
VOICE_PLAYER_SILKFILE_NULL	20486	音频播放文件内容为空，请确保文件内容正常
VOICE_PLAYER_SILKFILE_READ_ERROR	20487	读取音频文件失败，请确保文件格式正确
VOICE_PLAYER_INIT_DEVICE_ERROR	20488	设备初始化失败，请检查扬声器设备是否正常。

VOICE_PLAYER_ERROR	20489	播放失败，系统内部错误，例如线程创建，内存申请申请释放出错，建议重新启动播放音频文件

语音转文字错误码

错误码名称	错误码值	原因及建议方案
VOICE_ERR_VOICE_S2T_SYSTEM_INTERNAL_ERROR	32769	系统问题导致转文本失败，建议参见 开发问题 收集日志，联系 GME 开发者协助进行分析错误，解决问题。
VOICE_ERR_VOICE_S2T_NETWORK_FAIL	32770	网络问题导致转文本失败，检查设备网络是否可以正常访问外网环境，参见 如何检查网络 。
VOICE_ERR_VOICE_S2T_RSP_DATA_DECODE_FAIL	32772	网络错误导致转文本失败，检查设备网络是否可以正常访问外网环境，参见 如何检查网络 。
VOICE_ERR_VOICE_S2T_APPINFO_UNSET	32774	鉴权未设置： 1. 请检查 applyAuthBuffer 接口是否有调用。 2. 请检查 applyAuthBuffer 接口入参是否为空。
VOICE_ERR_VOICE_STREAMIN_RECORD_SUC_REC_FAIL	32775	流式语音转文本失败，但是录音成功了，请检查鉴权密钥是否正确
VOICE_ERR_VOICE_S2T_SIGN_CHECK_FAIL	32776	签名校验失败： 1. 请检查鉴权密钥是否正确。 2. 请检查是否有初始化离线语音。
VOICE_ERR_VOICE_STREAMIN_UPLOADANDRECORD_SUC_REC_FAIL	32777	流式语音转文本失败，但是录音成功了，上传也成功了，请检查网络是否正确，并重新调用接口将文件进行转文本操作
VOICE_ERR_VOICE_S2T_PARAM_NULL	32784	语音转文本参数错误，请检查代码中接口参数 fileid 是否为空。
VOICE_ERR_VOICE_S2T_AUTO_SPEECH_REC_ERROR	32785	语音转文本翻译返回错误，建议参见 开发问题 收集日志，联系 GME 开发者协助进行分析错误，解决问题。
VOICE_ERR_VOICE_STREAMIN_RUNING_ERROR	32786	调用接口时在流式语音消息转文本状态当中，请等待接口执行结果返回。
VOICE_ERR_VOICE_S2T_TRNSLATE_SERVICE_NOT_AVALIABLE	32787	翻译功能不支持。建议参见 开发问题 收集日志，联系 GME 开发者协助进行分析错误，解决问题。
VOICE_ERR_VOICE_S2T_TRNSLATE_LANGUAGE_NOT_SUPPORTED	32788	调用翻译的语言参数不支持，请重新检查接口传入参数
VOICE_ERR_VOICE_STREAMING_ASR_ERROR	32767698	请求流式语音消息转文本服务时出现错误，请重新上传录制文件（UploadRecordedFile），然后调用录音文件转文本接口（SpeechToText）。

伴奏功能错误码

错误码名称	错误码值	原因及建议方案
AV_ERR_ACC_OPENFILE_FAILED	4001	打开文件失败，检查文件是否存在
AV_ERR_ACC_FILE_FORAMT_NOTSUPPORT	4002	不支持的文件格式，查看文件格式是否在支持范围

		内
AV_ERR_ACC_DECODER_FAILED	4003	解码失败，查看文件格式是否在支持范围内
AV_ERR_ACC_BAD_PARAM	4004	参数错误，需要重新检查参数
AV_ERR_ACC_MEMORY_ALLOC_FAILED	4005	内存不足导致内存分配失败，建议清理设备内存
AV_ERR_ACC_CREATE_THREAD_FAILED	4006	创建线程失败，建议清理设备内存
AV_ERR_ACC_STATE_ILLIGAL	4007	状态非法，建议清理设备内存
AV_ERR_START_ACC_FIRST	4008	设备采集播放延迟录制开始前,要先开启伴奏
AV_ERR_START_ACC_IS_STARTED	4009	设备采集播放延迟预览开始前,要先停止伴奏
AV_ERR_HARDWARE_TEST_RECORD_IS_STARTED	4010	设备采集播放延迟预览前,要先停止录制
AV_ERR_HARDWARE_TEST_PREVIEW_IS_STARTED	4011	设备采集播放延迟录制开始前,要先停止预览
AV_ERR_HARDWARE_TEST_PREVIEW_DATA_IS_EMPTY	4012	设备采集播放延迟录制开始前,要先停止预览

音效功能错误码

错误码名称	错误码值	原因及建议方案
AV_ERR_EFFECT_OPENFILE_FAILED	4051	打开文件失败，建议检查文件是否存在，文件路径是否正确
AV_ERR_EFFECT_FILE_FORAMT_NOTSUPPORT	4052	不支持的文件格式，建议重新检查文件格式
AV_ERR_EFFECT_DECODER_FAILED	4053	解码失败，建议重新检查文件格式
AV_ERR_EFFECT_BAD_PARAM	4054	参数错误，建议检查接口传入参数是否正常
AV_ERR_EFFECT_MEMORY_ALLOC_FAILED	4055	内存分配失败，建议清理设备内存
AV_ERR_EFFECT_CREATE_THREAD_FAILED	4056	创建线程失败，建议清理设备内存
AV_ERR_EFFECT_STATE_ILLIGAL	4057	状态非法，建议清理设备内存

实时语音录制错误码

错误码名称	错误码值	原因及建议方案
AV_ERR_RECORD_OPENFILE_FAILED	5001	打开文件失败，建议检查文件是否存在，文件路径是否正确
AV_ERR_RECORD_FILE_FORAMT_NOTSUPPORT	5002	不支持的文件格式，建议重新检查文件格式
AV_ERR_RECORD_DECODER_FAILED	5003	解码失败，建议重新检查文件格式
AV_ERR_RECORD_BAD_PARAM	5004	参数错误，建议检查接口传入参数是否正常
AV_ERR_RECORD_MEMORY_ALLOC_FAILED	5005	内存分配失败，建议清理设备内存
AV_ERR_RECORD_CREATE_THREAD_FAILED	5006	创建线程失败，建议清理设备内存
AV_ERR_RECORD_STATE_ILLIGAL	5007	状态非法，建议清理设备内存

版权所有：腾讯云计算（北京）有限责任公司

其他错误码

错误码名称	错误码值	原因及建议方案
AV_ERR_3DVOICE_ERR_FILE_DAMAGED	7002	3D 音效文件未加载成功
AV_ERR_3DVOICE_ERR_NOT_INITED	7003	使用 3D 音效功能，需要先调用 InitSpatializer 接口
AV_ERR_NO_PERMISSION	7009	要进行某个操作的时候，没有权限，检查是否已经申请权限
AV_ERR_FILE_CANNOT_ACCESS	7010	无法访问文件，检查文件是否存在，路径是否正确
AV_ERR_FILE_DAMAGED	7011	文件被损坏，请检查文件是否可用
AV_ERR_SERVICE_NOT_OPENED	7012	没有在控制台开通实时语音功能，请在控制台上开通，参见 服务开通指引
AV_ERR_USER_CANCELED	7013	用户主动取消此操作，例如没进房成功就进行退房操作
AV_ERR_LOAD_LIB_FAILED	7014	库文件没有正常加载，请检查库文件是否缺失，以及所使用的功能，是否有对应的库文件存在，参见 库文件对应功能
AV_ERR_SDK_NOT_FULL_UPDATE	7015	升级 SDK 时，没有升级所有的文件，导致某些模块不匹配，请全量升级 SDK，参见 SDK 版本升级指引
AV_ERR_MUTESWITCH_DECTECT_ERR	7018	iOS 静音开关检测错误，查看是否是在 iOS 机型上调用
AV_ERR_SERVICE_NOT_OPENED	7022	没有在控制台开通实时语音消息功能或者转文本功能，请在控制台上开通，参见 服务开通指引