即时通讯项目

**即时通讯客户端集成用户手册**

版本<1.0>

修订文档历史记录

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | 日期 | 文档版本 | 作者 | 编写/修订说明 |
| 1 | 2017.2.15 | V1.0 | 孙翔 | 初稿 |
| 2 | 2017.3.29 | V1.1 | 孙翔 | 初稿，增加了音频的录制和播放接口 |

目录

[1. 简介 5](#_Toc489953487)

[1.1 功能 5](#_Toc489953488)

[1.2 范围 5](#_Toc489953489)

[1.3 定义、首字母缩写词和缩略语 5](#_Toc489953490)

[1.4 主要需求 5](#_Toc489953491)

[2. SDK功能说明 6](#_Toc489953492)

[2.1 框架设计 6](#_Toc489953493)

[2.2 主要功能 6](#_Toc489953494)

[3. 开发前准备 7](#_Toc489953495)

[4. 客户端集成说明 7](#_Toc489953496)

[4.1 使用 SDK开发应用 7](#_Toc489953497)

[4.1.1 添加 SDK 到APP工程 7](#_Toc489953498)

[4.1.2 调用API 8](#_Toc489953499)

[5. 客户端API接口说明 8](#_Toc489953500)

[5.1 类说明 8](#_Toc489953501)

[5.2 接口说明 8](#_Toc489953502)

[5.2.1 实例化IMClient（此处的Context参数最好是Application的） 8](#_Toc489953503)

[5.2.2 获取IMClient的实例 9](#_Toc489953504)

[5.2.3 用户登录接口 9](#_Toc489953505)

[5.2.4 用户下线接口 9](#_Toc489953506)

[5.2.5 注册消息监听器 9](#_Toc489953507)

[5.2.6 构建文本消息 9](#_Toc489953508)

[5.2.7 构建语音消息 10](#_Toc489953509)

[5.2.8 发送消息 10](#_Toc489953510)

[5.2.9 获取当前登录的用户 11](#_Toc489953511)

[5.2.10 获取当前登录的用户 11](#_Toc489953512)

[5.2.11 获取离线消息 11](#_Toc489953513)

[5.2.12 获取所有会话 11](#_Toc489953514)

[5.2.13 根据联系人获取对应会话 11](#_Toc489953515)

[5.2.14 删除指定会话 12](#_Toc489953516)

[5.2.15 删除指定会话 12](#_Toc489953517)

[5.2.16 创建群组 12](#_Toc489953518)

[5.2.17 添加群组成员 12](#_Toc489953519)

[5.2.18 加入群组或者讨论组 13](#_Toc489953520)

[5.2.19 退出群组或者讨论组 13](#_Toc489953521)

[5.2.20 获取群组信息 13](#_Toc489953522)

[5.2.21 获取群组列表 13](#_Toc489953523)

[5.2.22 查询所有未读消息 14](#_Toc489953524)

[5.2.23 查询所有消息 14](#_Toc489953525)

[5.2.24 查询指定条数的消息 14](#_Toc489953526)

[5.2.25 更新消息发送状态 14](#_Toc489953527)

[5.2.26 搜索群组 15](#_Toc489953528)

[5.2.27 下载文件 15](#_Toc489953529)

[5.2.28 发送已读回执 15](#_Toc489953530)

[5.2.29 获取所有的群组和讨论组 15](#_Toc489953531)

[5.2.30 获取所有的讨论组 16](#_Toc489953532)

[5.2.31 根据gid获取讨论组或群组 16](#_Toc489953533)

[5.2.32 获取群组或者讨论组名称 16](#_Toc489953534)

[5.2.33 解析文本消息（json字符串） 16](#_Toc489953535)

[5.2.34 解析语音消息 16](#_Toc489953536)

[5.2.35 解析后处理（转成文字消息)json 字符串 17](#_Toc489953537)

[5.2.36 解析后处理消息（文字转语音） 17](#_Toc489953538)

[5.2.37 设置录制语音的目录 17](#_Toc489953539)

[5.2.38 开始录音 17](#_Toc489953540)

[5.2.39 停止录制 18](#_Toc489953541)

[5.2.40 开始播放 18](#_Toc489953542)

[5.2.41 停止播放 18](#_Toc489953543)

[5.2.42 暂停播放 18](#_Toc489953544)

[5.2.43 初始化播放器 18](#_Toc489953545)

[5.2.44 设置是否打印日志 19](#_Toc489953546)

[5.3 接口调用实例 19](#_Toc489953547)

[5.3.1 登录接口 19](#_Toc489953548)

[5.3.1 登出接口 20](#_Toc489953549)

[5.3.2 消息构建接口 21](#_Toc489953550)

[5.3.3 消息发送接口 22](#_Toc489953551)

[5.3.4 音频录制接口 23](#_Toc489953552)

[5.3.5 录音播放接口 24](#_Toc489953553)

[5.3.6 其他接口 25](#_Toc489953554)

消息推送客户端用户手册

# 简介

## 功能

IGIM基础功能是满足APP用户互动需求的基本功能。

从聊天方式上包括单聊、群聊；从信息类型上包括文字、图片、语音、视频、通知消息、文字转语音、语音转文字等。

通过IGIM，不仅可以实现常见功能：即时聊天、消息通知、创建群组、一般群组功能等，还能完整打造易信、微信级大用户量级社交产品。

## 范围

本文档面向所有使用该 SDK的Android开发人员、测试人员、合作伙伴以及对此感兴趣的其他用户。

## 定义、首字母缩写词和缩略语

* **IGIM** 即时通讯软件开发包
* **Android** 谷歌公司开发的移动终端操作系统
* **SDK** 软件开发包 (Software Development Kit)
* **API** 应用编程接口 (Application Programming Interface)

## 主要需求

1. App与UID登录并互相绑定。
2. 客户端与服务端保持长连接。
3. 客户端之间能够通过服务端互相发送聊天消息。
4. 聊天消息包括文字、图片、语音、视频、通知消息、文字转语音、语音转文字。
5. 客户端能够建立群组，群组中部分用户有足够的权限管理群组等。

# SDK功能说明

## 框架设计

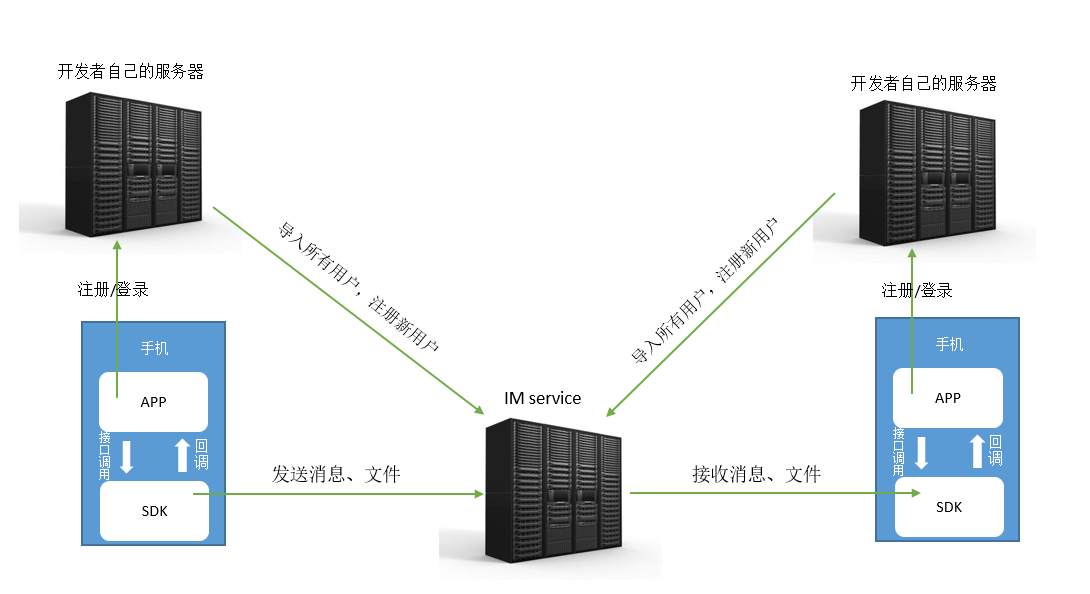


图1 IGIM 框架设计图

如上所示，上图为粗略的客户端与服务器的交互图。客户端A通过调用SDK（A）接口，将消息通过长连接或短连接发送至服务端，服务端根据消息的具体内容，按照规定的协议发送给SDK（B），SDK（B）将消息返回到客户端B。开发者可以使用自己的用户服务器，只需要将自己服务器中的用户数据导入即可。

## 主要功能

IM Android SDK提供了一系列的接口供开发者调用。开发者利用这些接口，可以开发之间的社交APP。SDK提供以下功能：

* 网络连接管理
* 登录认证服务
* 消息传输服务
* 消息存储服务
* 群组管理

# 开发前准备

⑴ 开发者需到语音云服务器注册开发者账户，获取应用appid和token。

⑵ 开发者需下载IGIM.jar,

⑶ 将下载的jar文件导入到项目中。

# 客户端集成说明

## 使用 SDK开发应用

### 添加 SDK 到APP工程

1. 创建一个Android Project
2. 在该工程下创建一个libs文件夹
3. 将开发前准备时下载的jar文件（包含IGIM和Sunflower）拷贝至刚刚创建的libs目录下
4. 将上述jar包添加到工程的java build path
5. AndroidManifest.xml声明permission

|  |
| --- |
| <!—IM\_SDK 运行需要的权限 --/>  <uses-permission android:name="android.permission.READ\_PHONE\_STATE" /> <uses-permission android:name="android.permission.INTERNET" /> <uses-permission android:name="android.permission.ACCESS\_WIFI\_STATE" /> <uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE" /> <uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE" /> <uses-permission android:name="android.permission.MOUNT\_UNMOUNT\_FILESYSTEMS" />  <uses-permission android:name="android.permission.GET\_TASKS" />  <uses-permission android:name="android.permission.READ\_EXTERNAL\_STORAGE" />  <uses-permission android:name="android.permission.RECEIVE\_BOOT\_COMPLETED" /> <uses-permission android:name="android.permission.WRITE\_SETTINGS" />  <uses-permission android:name="android.permission.ACCESS\_DOWNLOAD\_MANAGER" /> <uses-permission android:name="android.permission.DOWNLOAD\_WITHOUT\_NOTIFICATION" /> <uses-permission android:name="android.permission.DISABLE\_KEYGUARD" /> <uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION" /> <uses-permission android:name="android.permission.CHANGE\_NETWORK\_STATE" /> <uses-permission android:name="android.permission.MOUNT\_UNMOUNT\_FILESYSTEMS" /> <uses-permission android:name="andorid.permission.CHANGE\_CONFIGURATION" /> <uses-permission android:name="android.permission.SYSTEM\_ALERT\_WINDOW" /> |

1. AndroidManifest.xml增加 appID配置

|  |
| --- |
| <!-- 网络申请的appID --> <meta-data  android:name="IFLYTEK\_APPKEY"  android:value="" /> |

### 调用API

下面介绍如何调用 SDK中已封装的API完成各项操作：

1. 在主Activiy的OnCreate方法中，调用接口 createInstance

|  |
| --- |
| IMClient.createInstance(context); //context 最佳选择是Application的context |

1. 调用IMClient封装的各个接口

调用方式就是用“IMClient.getInstance().”加上接口名。如下所示调用登录接口。

|  |
| --- |
| IMClient.getInstance().login(uid,isForce,token,new resultCallBack <String> () {}); |

# 客户端API接口说明

## 类说明

客户端SDK对外提供的类如下：

|  |  |
| --- | --- |
| 类名 | 描述 |
| IMClient | 调用SDK接口的封装类，提供静态方法供外部调用 |
| IMClientError | SDK中可能返回的错误码。 |
| Group | 群组类的封装 |
| Conversation | 会话类的封装 |
| 其他消息类 | 其他消息类，在com.iflytek.cloud.im.entity.msg包中 |

## 接口说明

### 实例化IMClient（此处的Context参数最好是Application的）

|  |
| --- |
| public static void createInstance(Context context) |

功能： 创建IMClient的实例

参数： context 上下文

### 获取IMClient的实例

|  |
| --- |
| public synchronized static IMClient getInstance()获取 |

功能： 获取IMClient的实例

### 用户登录接口

|  |
| --- |
| public void login(User user, boolean isForce, String token, ResultCallback<String> listener) |

功能： 开发者调用该接口，用户登录。

参数： user 用户实例   
 isForce 是否选择强制登录  
 token 每一个用户对应的令牌  
 listener 登录状态监听器  
 结果返回：无

### 用户下线接口

|  |
| --- |
| public void logout(ResultCallback<String> listener, boolean needPush) |

功能： 用户下线，根据参数needPush，服务端确定是否继续推送消息。

参数： listener 下线操作回调  
 needPush 下线后是否继续推送消息

结果返回：无

### 注册消息监听器

|  |
| --- |
| public void regMsgListener (MsgListener msgListener); |

功能： 设置消息监听器，当SDK收到来自服务端的普通消息会从该监听器返回到开发者的APP中。

参数： msgListener 下行消息监听器

结果返回：无

### 构建文本消息

|  |
| --- |
| public CommonMsgContent buildTextMsg(String receiver, String content, boolean isGroup, int postType, String ext) |

功能： 构建文本消息

参数： receiver 消息接收人的ID

content 消息内容（文字）

isGroup 是否是群组消息

postType 后处理类型 文字转语音 1

ext 扩展字段

结果返回：构建成功的文本消息

### 构建语音消息

此接口既有返回值又有回调。返回值是为了给你做页面展示，而回调则是真正构建成功，在构建成功回调中发送消息。否则对端无法下载该消息中的文件。修改界面的展示可根据消息的唯一性标志MsgId进行查找。除构建文本消息只有返回值以外，其他消息构建接口均与此相似。

|  |
| --- |
| public CommonMsgContent buildAudioMsg(String receiver, String audioPath, boolean isGroup, int postType, String ext, BuildMsgResultCallback<CommonMsgContent> listener) |

功能： 构建语音消息

参数： receiver 消息接收人的ID

imgPath 语音在本地的路径

isGroup 是否是群组消息

postType 后处理类型 语音转文字2

ext 扩展字段

listener 构建消息回调

结果返回：构建成功的语音消息

### 发送消息

|  |
| --- |
| public void sendMessage(MessageContent content, SendMessageCallback<String> listener) |

功能： *发送消息*

参数： content 消息内容

listener 消息发送状态监听器

结果返回：无

### 获取当前登录的用户

|  |
| --- |
| public String getCurrentUser() |

功能： 获取当前登录的用户

参数： 无

结果返回：用户ID

### 获取当前登录的用户

|  |
| --- |
| public void isOnline(ResultCallback<Boolean> callback) |

功能： 查询当前用户的在线情况

参数： 结果回调

结果返回：无

### 获取离线消息

|  |
| --- |
| public void syncAllMsgs(SyncListener listener) |

功能： 获取离线消息

参数： listener 获取离线消息的回调，回调成功的参数是离线拉取的所有消息。

结果返回：无

### 获取所有会话

|  |
| --- |
| public List<Conversation> getAllConv() |

功能： 获取所有会话

参数： 无

结果返回：所有会话的集合

### 根据联系人获取对应会话

|  |
| --- |
| public Conversation getConvByContact (String contact) |

功能： 根据联系人获取对应的会话

参数： contact 联系人ID

结果返回：会话

### 删除指定会话

|  |
| --- |
| public boolean delConvSynch(Conversation conversation) |

功能： 删除指定会话

参数： conversation 要删除的会话

结果返回：是否正确删除会话

### 删除指定会话

|  |
| --- |
| public boolean delConvById(String chatID) |

功能： 删除指定会话

参数： chatID 会话的聊天对象ID

结果返回：是否正确删除

### 创建群组

|  |
| --- |
| public void createGroup(Map<String, String> createGroupParams, ResultCallback<String> listener) |

功能： 创建群组

参数： createGroupParams 创建群组的参数集合：gname,type（0是讨论组，1是群组）

listener 创建群组回调

结果返回：无

### 添加群组成员

|  |
| --- |
| public void addMemToGroup(Map<String, String> addMemberToGroupParams, ResultCallback<String> listener) |

功能： 添加群组成员

参数： addMemberToGroupParams 添加群成员的参数集合，gid,members（JsonArray格式的）,type（0是讨论组，1是群组）,msg（添加群成员的说明）

listener 添加群组成员回调

结果返回：无

### 加入群组或者讨论组

|  |
| --- |
| public void joinToGroup(Map<String, String> joinToGroupParams, ResultCallback<String> listener) { |

功能： 加入群组或者讨论组

参数： joinToGroupParams 加入群组的参数集合

listener 加入群组的回调

结果返回：无

### 退出群组或者讨论组

|  |
| --- |
| public void exitFromGroup(Map<String, String> exitFromGroupParams, ResultCallback<String> listener) |

功能： 退出群组或者讨论组

参数： exitFromGroupParams 退出群组的参数集合 gid

listener 退出群组的回调

结果返回：无

### 获取群组信息

|  |
| --- |
| public void getGroupInfo(Map<String, String> getGroupInfoParams, ResultCallback<Group> listener) |

功能： 获取群组的详细信息

参数： getGroupInfoParams 获取群组详细消息的参数集合

listener 获取群组详细消息的回调

结果返回：无

### 获取群组列表

|  |
| --- |
| public void getGroupList(ResultCallback<List<Group>> listener) |

功能： 获取群组列表

参数： listener 获取群组列表的回调

结果返回：无

### 查询所有未读消息

|  |
| --- |
| public ArrayList<MessageContent> queryUnreadMsgByChatId(String chatId) |

功能： 根据联系人ID，查询所有未读消息

参数： chatId 联系人ID

结果返回：该联系人所有未读消息集合

### 查询所有消息

|  |
| --- |
| public ArrayList<MessageContent> queryMsgByChatId(String chatId) |

功能： 根据联系人ID，查询所有消息

参数： chatId 联系人ID

结果返回：该联系人所有消息集合

### 查询指定条数的消息

|  |
| --- |
| public ArrayList<MessageContent> pagedQueryMsgByChatId(String chatId, int count, MessageContent msg) |

功能： 根据联系人ID，查询指定条数的消息

参数： chatId 联系人ID

count 指定消息条数

msg 最旧的那条消息

结果返回：该联系人指定条数消息集合

### 更新消息发送状态

|  |
| --- |
| public boolean updateMsgStateById(String msgId, int state) |

功能： 更新消息发送状态

参数： msgID 群组通知消息的唯一标识

state 需要更新的目标状态

结果返回：是否正确更新

### 搜索群组

|  |
| --- |
| public void searchGroup(String keyWords, ResultCallback<List<Group>> listener) |

功能： 搜索群组 在服务端搜索群组，供加群使用

参数： keyWords 搜索关键词

listener 搜索群组回调

结果返回：无

### 下载文件

|  |
| --- |
| public void downloadFile(CommonMsgContent commonMsgContent, boolean isVideo, ResultCallback resultCallback) { |

功能： 下载文件

参数： commonMsgContent 需要下载文件的那个消息

isVideo 是不是视频文件

resultCallback 下载回调

结果返回：无

### 发送已读回执

|  |
| --- |
| public void sendReadReceipt(MessageContent msg) |

功能： 发送已读回执，不同客户端同账号根据此回执更新消息已读，聊天对端根据此回执判断对端是否已读所有消息。

参数： msg 发送已读回执时的最新消息

结果返回：无

### 获取所有的群组和讨论组

|  |
| --- |
| public List<Group> getAllGroups() |

功能： 获取所有的群组和讨论组

参数： 无

结果返回：所有群组和讨论组的集合

### 获取所有的讨论组

|  |
| --- |
| public List<Group> getDiscussionGroups() |

功能： 获取所有的讨论组

参数： 无

结果返回：所有讨论组的集合

### 根据gid获取讨论组或群组

|  |
| --- |
| public Group getGroupByGid(String gid) |

功能： 根据gid获取讨论组或群组

参数： gid 群组和讨论组的唯一标识

结果返回：群组或者讨论组

### 获取群组或者讨论组名称

|  |
| --- |
| public String getGroupNameByGid(String gid) |

功能： 根据gid查询群组或者讨论组名称

参数： gid 群组和讨论组的唯一标识

结果返回：群组或讨论组的名称

### 解析文本消息（json字符串）

|  |
| --- |
| public static TextMsg parse2TextMsg(CommonMsgContent msg) |

功能： CommonMsgContent需要通过该方法转换为普通文本消息

参数： msg 包含文本消息的普通消息

结果返回：文本消息

### 解析语音消息

|  |
| --- |
| public AudioMsg parse2AudioMsg(CommonMsgContent msg) |

功能： CommonMsgContent需要通过该方法转换为语音消息

参数： msg 包含语音消息的普通消息

结果返回：语音消息

### 解析后处理（转成文字消息)json 字符串

|  |
| --- |
| public static PostRltText parse2PostTextRlt(CommonMsgContent msg ) |

功能： CommonMsgContent需要通过该方法转换为后处理消息，语音转文字消息

参数： msg 包含后处理消息的普通消息

结果返回：后处理消息

### 解析后处理消息（文字转语音）

|  |
| --- |
| public PostRltVoice parse2PostVoiceRlt(CommonMsgContent msg) |

功能： CommonMsgContent需要通过该方法转换为后处理消息，文字转语音消息

参数： msg 包含后处理消息的普通消息

结果返回：后处理消息

### 设置录制语音的目录

|  |
| --- |
| public void setAudioPath(String audioPath) |

功能： 设置录制语音的目录

参数： audioPath 录制语音的目录，某个文件夹的目录

结果返回：无

### 开始录音

|  |
| --- |
| public void startRecording(String fileName,PcmRecordListener listener) |

功能： 开始录音

参数： fileName 最后生成的文件名

listener 录制语音的回调

结果返回：无

### 停止录制

|  |
| --- |
| public void stopRecord() |

功能： 停止录音

参数： 无

结果返回：无

### 开始播放

|  |
| --- |
| public void startPlay() |

功能： 开始播放

参数： 无

结果返回：无

### 停止播放

|  |
| --- |
| public void stopPlay() |

功能： 停止播放

参数： 无

结果返回：无

### 暂停播放

|  |
| --- |
| public void pausePlay() |

功能： 暂停播放

参数： 无

结果返回：无

### 初始化播放器

|  |
| --- |
| public void initPlayer(String filePath,AudioPlayer.PlayerListener listener) |

功能： 初始化播放器

参数： filePath 播放文件路径

listener 播放文件的回调

结果返回：无

### 设置是否打印日志

|  |
| --- |
| public void setDebugAble(boolean debugable) |

功能： 是否输出SDK的日志

参数： debugable 是否打印

true 打印

false 不打印

## 接口调用实例

简单集成，可以只用下面的登录，构建消息，发送消息的接口。若开发者也可以自己开发录音接口，但提供的录音接口中录制的是本公司特定的格式，转写结果更为精确。

### 登录接口

实例如下

|  |
| --- |
| private void login() {  //调用登录接口，对回调接口实例化  IMClient.getInstance().login(mUser, true, mToken, new ResultCallback<String>() {  @Override  public void onSuccess(String data) {  //登录成功，保存密码，修改界面  loginSuccess();  }  @Override  public void onError(int errorCode) {  //登录失败，对返回的错误码分析  if (errorCode == IMClientError.ERROR\_LOGIN\_EXCEPTION) {  //普通登录错误，进行详细分析，并修改界面  parseLoginErrorCode(OTHER\_ERROR);  } else if (errorCode == IMClientError.ERROR\_NETWORK\_EXCEPTION) {  //网络错误，修改界面  parseLoginErrorCode(NETWORK\_ERROR);  } else if (errorCode == ERROR\_ONLINE) {  //该用户已在线错误，提示或者强行登录  parseLoginErrorCode(LOGIN\_SOMENE\_ONLINE);  }    }});  } |

### 登出接口

|  |
| --- |
| private void logout() {  //登出接口调用  IMClient.getInstance().logout(new ResultCallback<String>() {  @Override  public void onError(int errorCode) {  //登出错误  if(errorCode == ERROR\_NETWORK\_EXCEPTION) {  //如果是网络中断造成错误，默认登出成功，切换至登出成功界面  logoutSuccess();  }else {  //其他错误，按照需求进行处理，这里只是显示了错误  ToastUtil.showText("error");  }  }  @Override  public void onSuccess(String datas) {  //登出成功  logoutSuccess();  }  }, false);  } |

### 消息构建接口

1. 构建文本消息

|  |
| --- |
| //构建文本消息，具体参数参照5.2.6。返回的msg可以先做显示使用。  CommonMsgContent msg = IMClient.getInstance().buildTextMsg(mConnectPersonID, text, isGroup, txtPostType, ext); |

1. 构建音频消息

|  |
| --- |
| //构建音频消息，参数参照5.2.23  //返回的commonMsgContent可以先做界面显示使用，如消息先展示，并显示正在发送。  //发送至服务器之后才会有回调  CommonMsgContent commonMsgContent = IMClient.getInstance().buildAudioMsg(mConnectPersonID, audioFilePath, isGroup, postType, ext, new BuildMsgResultCallback<CommonMsgContent>() {  @Override  public void onSuccess(final CommonMsgContent msg) {  //消息构建成功，发送消息。  sendMsg(msg);  }  @Override  public void onError(Object msgID, int errorCode) {  //构建失败，构建音频消息，除了参数可能有误造成构建失败。  //构建音频消息需要上传音频，所以构建消息失败也可能是上传失败  String messageId = (String) msgID;  //构建音频消息失败，界面显示失败  notifyAdapterDataChanged(messageId, MessageConstant.MESSAGE\_CONTENT\_SEND\_UN);  }  }); |

### 消息发送接口

|  |
| --- |
| //消息发送接口  IMClient.getInstance().sendMessage(msg, new SendMessageCallback<String>() {  @Override  public void onFaile(String messageId, int errorCode) {  //发送失败，更新发送失败界面  notifyAdapterDataChanged(messageId, MessageConstant.MESSAGE\_CONTENT\_SEND\_UN);  }  @Override  public void onSuccess(String messageId, long seqID) {  //发送成功，更新发送成功界面  notifyAdapterDataChanged(messageId, MessageConstant.MESSAGE\_CONTENT\_SEND\_DONE);  }  }); |

### 音频录制接口

|  |
| --- |
| //音频录制接口  //设置文件保存路径   dir是string类型，到达需要存储的文件夹的名称。     IMClient.getInstance().setAudioPath(dir);     //开始录音     IMClient.getInstance().startRecording(mCurrentFileName, new PcmRecordListener() {             @Override             public void onRecordBuffer(int length, double volume) {                 //length 录制文件 单位：byte                 //volume 音量大小 单位：分贝值    }             @Override             public void onError(int error) {                 //录制错误，返回错误码             }              @Override             public void onRecordStarted(boolean success) {                 //开始录制的回调             }              @Override             public void onRecordFinished(String filePath) {                 //录制结束，返回文件路径             }     }); |

### 录音播放接口

|  |
| --- |
| //录音播放接口  //初始化播放接口，filePath--播放文件接口，  IMClient.getInstance().initPlayer(filePath, new AudioPlayer.PlayerListener() {  @Override  public void onPause() {  //暂停回调  }  @Override  public void onStart() {  //开始回调  }  @Override  public void onStop() {  //停止回调  }  @Override  public void onCompleted() {  //完成回调  }  @Override  public void onError(int errorCode) {  //错误回调  }  });  //开始播放  IMClient.getInstance().startPlay();  //停止播放  IMClient.getInstance().stopPlay();  //暂停播放  IMClient.getInstance().pausePlay(); |

### 其他接口

其他接口请参照Demo中的调用。