小西即时消息接口文档(安卓)

V1.0.1

一、文档编写背景

本文档是针对融合通信自研即时消息 Android 客户端 SDK 对第三方进行能力开放而编写的接口详细说明，以满足开发人员开发、测试人员进行接口测试。

二、SDK 适用范围及平台

SDK 适用范围：需要接入即时消息通信功能的所有 Android 应用；

SDK 适用平台：Android 2.3 及以上版本；

三、SDK 说明

3.1 能力说明

SDK 登录前初始化：提供给第三方应用使用本 SDK 进行即时消息应用开发时

SDK 的初始化、路径的配置、广播的监听等工作；

用户注册：提供给第三方应用对互联网用户进行注册账号的能力；

用户登录：提供给已经注册了第三方应用账号的用户进行登录鉴权；

连接监听：提供给已登录第三方应用的账号连接状态监听能力，如中间与服务器断开、

连接；账号冲突；

账户退出：提供给已登录第三方应用的账号退出应用、释放 SDK 的能力；

联系人变化监听以及接收消息监听：用于添加联系人变化监听以及接收消息监听。

获取好友列表：获取自己的所有好友列表

搜索联系人：按关键字模糊查找好友

添加好友：发送添加对方为好友请求，等待好友验证

删除好友：将联系人从自己的好友列表中移除

处理好友请求：同意或者拒绝某用户发来的添加好友请求

发送消息：用户登录了之后可以给他人发送文本、语音、表情、图片和视频的消息

获取群列表：获取自己所在的所有群

创建群：用户可以选择 2 个以上的好友发起群聊

添加群成员：用户可在现有的群中添加新的成员

退出群：群组内的任何成员都可以退出该群聊

根据群 ID 获取群名称: 用户可根据一个群 ID 获取该群的群名称

根据群 ID 获取群成员: 用户可根据一个群 ID 获取该群的群名称和群内所有的成员

3.2 使用说明

A）下载小西 sdk 后解压至本地目录。将其中的 jar 包，名字一般为 littlec-版本号.jar 中的内容拷贝到您的工程的 libs 目录下，如果您的工程没有 libs 文件夹，需先创建一个 libs 文件夹。

在 Eclipse 中（其他开发工具参阅其自带说明）会自动将 libs 目录作为工程的库引用进去。

B）AndroidManifest.xml 文件中权限的配置

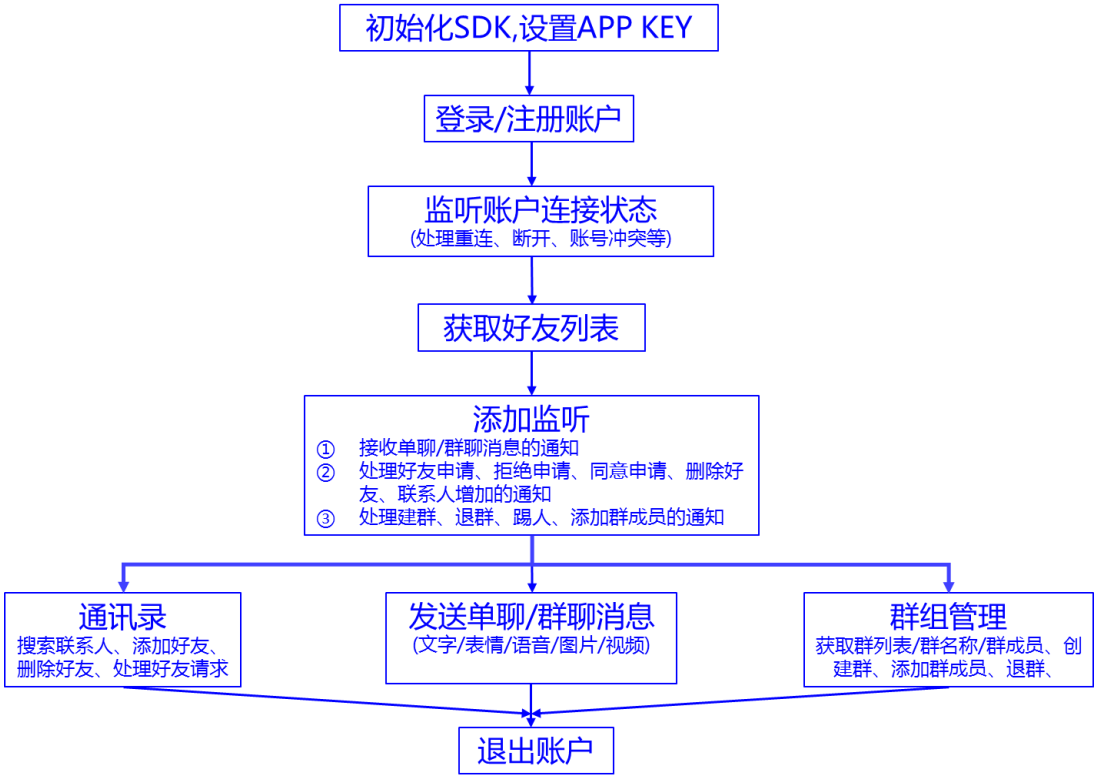
```
INTERNET:允许程序联网和发送聊天数据的权限
RECORD_AUDIO:允许程序进行语音录制的权限
ACCESS_NETWORK_STATE:允许查看当前网络状态
ACCESS_WIFI_STATE: 允许程序访问 Wi-Fi 网络状态信息
WRITE_EXTERNAL_STORAGE:允许程序进行读写 sdcard
```

示例代码：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ……>
    <uses-sdk
          android:minSdkVersion="9" android:targetSdkVersion="19"/>
     <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.RECORD_AUDIO" />
     <application ……>
          <activity ……></activity>
          </activity>
     </application>
</manifest>
```

四、流程图

4.1 流程图



五、接口说明

5.1 公共模块

5.1.1 初始化

a) 所属类

com.littlec.sdk.manager. CMAccountManager

b) 接口说明

提供给第三方应用使用本 SDK 进行即时消息应用开发时，用于 SDK 的初始化、路径的配置、广播的监听等，只需要初始化一次，在 application 的 onCreate 函数中调用。

- **方法原型**

public void init(Context appContext, String appKey)

- **入参说明**

appKey: 表示被哪个应用调用，用于处理不同应用的账户分隔，也就是每个应用都会有自己单独的 appkey，开发者可以在官网上注册自己的开发者账号得到这样的 appkey，共 8 位，6 位数字+2 位字母。

appContext：继承于 Aplication 的 Context。

- **出参说明**

无返回值，用于 SDK 的初始化、路径的配置、广播的监听等。

- **基本异常处理**

当 appContext 为 null 时，方法直接返回。

c) 接口调用场景说明

一定要在 android 应用中继承于 Application 的子类在回调方法 OnCreate 中调用该接口。

d) 接口调用实例

//该接口一般在自定的 application 的 onCreate 函数中调用

//appkey 为开发者自己在官网申请到的

```
CMIMHelper.getCmAccountManager().init(getApplicationContext(), "您在管理后台申请的 APP KEY");
```

5.1.2 用户注册

a) 所属类

com.littlec.sdk.manager. CMAccountManager；

b) 接口说明

提供给第三方应用用以对互联网用户进行注册账号的数据有效性验证及账号注册能力。

- **方法原型**

共有两种注册方法：

第一种：注册信息中带有手机号信息，需获取短信验证码才能注册成功。

分为 2 步：

1）请求获取验证码：

public void getConfigurationForm

(HashMap<String,String> registerInfo, OnCMListener registerListener)。

2）输入 4 位的验证码，传入注册表单：

根据上一个接口，请求成功后，调用下面这个接口注册用户。

public void createAccountWithCode (String code, OnCMListener registerListener)

第二种：注册信息中不带有手机号信息。

public void createAccount(HashMap<String, String> registerInfo, OnCMListener mListener) {
}

➢ 入参说明

registerInfo： 存储注册的键值对，包含键值如下：

CMSdkContants.CM_USER_NAME：注册的用户名，账号规则：1-50 个字符，字母、数字、下划线，不支持空格，不区分大小写

CMSdkContants.CM_NICK_NAME :用户昵称 ,昵称规则: 1-50 个字 ,可包含字母、数字、下划线、特殊字符或表情，可包含空格（但不能以空格开头或结尾），区分大小写，（可选）

CMSdkContants.CM_PHONE：注册账户对应的电话号码，需要为那个的电话号码;

CMSdkContants.CM_PASSWORD：注册的密码，6-32 个字符，字母、数字、特殊符号(`~!@#$%^&*()-=_+[]{}|\;:'",./<>?)，不能包含空格，区分大小写.

CMSdkContants.CM_CONFIRM_PASSWORD：确认密码，与 `passWord` 参数要一致；

registerListener：结果回调接口，不能为 `null`，用来处理注册成功或失败；

code：验证码，4 位

- **出参说明**

无返回值，该方法将进行注册数据有效性验证及用户注册，将注册结果返回给客户端。

- **基本异常处理**

当 registerInfo 中存储的参数 userName、passWord、comfirmPassWord 为 null 或规则不合法通过回调接口通知客户端注册失败，当 registerListener 为 null，抛出空指针异常。

c) 接口调用场景说明

当第三方应用需要注册新用户时调用该接口。

e) 接口调用实例

第一种：注册信息中带有手机号信息，需获取短信验证码才能注册成功。

```
//添加注册信息的hashmap表
    HashMap<String,String> registerInfo = new HashMap<String, String>();
    registerInfo.put(CMSdkContants.CM_USER_NAME,userName);
    registerInfo.put(CMSdkContants.CM_NICK_NAME,nickName);
    registerInfo.put(CMSdkContants.CM_PHONE,phone);
    registerInfo.put(CMSdkContants.CM_PASSWORD ",passWord);
    registerInfo.put(CMSdkContants.CM_CONFIRM_PASSWORD, confirmPassWord);
    //第一步，调用获取注册表单接口
    CMIMHelper.getCmAccountManager().getConfigurationForm (registerInfo,
new OnCMListener () {
            @Override
            public void onSuccess() {
            //成功返回的回调，
            //跳转到输入验证码的界面， }
            @Override
            public void onFailed(String msg) {
            //失败的回调}
});
输入验证码后，
            //第二步，设置验证码，注册用户
            String code=界面输入的验证码；
CMIMHelper.getCmAccountManager().createAccountWithCode (code, new OnCMListener()
{
                public void onSuccess() {}
                public void onFailed(String msg) {}
                });
```

第二种：注册信息中不带有手机号信息。

```
    HashMap<String,String> registerInfo = new HashMap<String, String>();
    registerInfo.put(CMSdkContants.CM_USER_NAME,userName);
```

5.1.3 用户登录

a) 所属类

com.littlec.sdk.manager. CMAccountManager；

b) 接口说明

提供给已经注册了第三方应用账号的用户登录时数据的有效性验证及用户登录功能。

- **方法原型**

public void doLogin(String userName, String passWord, OnCMListener loginListener)。

- **入参说明**

userName：注册时的账号，账号规则：1-50 个字符，由字母、数字、下划线组成，不支持空格，区分大小写

passWord：注册时的密码，注册的密码，6-32 个字符，字母、数字、特殊符号(`~!@#$%^&*()-=_+[]{}|\;:'",./<>?)，不能包含空格，区分大小写.

loginListener：登录结果回调接口，不能为 null，用来处理登录成功或失败；

- **出参说明**

无返回值，登录成功后则配置一些临时路径，设置数据库，发送消息的对象初始化等，然后通知客户端。

- **基本异常处理**

当参数 userName、 passWord 为 null 时通过回调接口通知客户端登录失败，当 loginListener 为 null，抛出空指针异常。

c) 接口调用场景说明

当第三方应用需要用户登录时，调用该接口。

f) 接口调用实例

```
CMIMHelper.getCmAccountManager().doLogin(userName,passWord,new OnCMListener () {
        @Override
        public void onSuccess() {
        //登录成功的处理代码
        }

        @Override
        public void onFailed(String errorMsg) {
        //登录失败的处理代码，参数 errorMsg 为失败的原因的描述字符串
        }
    });
```

5.1.4 连接状态监听

a) 所属类

com.littlec.sdk.manager. CMAccountManager

b) 接口说明

提供给已登录第三方应用的账号连接状态监听能力，如中间与服务器断开、连接；账号冲突。

➢ 方法原型

public void addConnectionListener(OnConnectionListener listener);

➢ 入参说明

listener：(可选)监听连接状态及账号冲突通知客户端回调接口。

➢ 出参说明

无返回值 ,当与服务器连接断开、重新连接成功或已登录的账号再次在另一客户端登录，通知客户端。

➢ 基本异常处理

当 listener 为 null 时，方法直接返回。

c) 接口调用场景说明

在登录成功之后，调用该接口。

g) 接口调用实例

```
CMIMHelper.getCMAccountManager().addConnectionListener(new OnConnectionListener(){
        @Override
        public void onReConnected() {
        //账号与服务器断开连接后自动进行重连的处理函数

            System.out.print("与服务器重新连接成功!");
        }
        @Override
        public void onDisConnected() {
        //账号与服务器异常断开连接的处理代码

            System.out.print("与服务器连接断开!");
        }
        @Override
        public void onAccountConflict() {
        //账号冲突的回调处理代码：同一个账号在其它终端上又登陆了


        }
    @Override
    public void onAccountDestroyed() {
        // TODO Auto-generated method stub
    //账号被后台删除的回调处理代码


    }
});
```

5.1.5 账户退出

a) 所属类

com.littlec.sdk.manager. CMAccountManager；

b) 接口说明

提供给已登录第三方应用的账号退出应用前账号退出、释放 SDK 的能力。

➢ 方法原型

public void doLogOut()；

➢ 入参说明

无入参

➢ 出参说明

无返回值，该操作后用户与服务器断开，释放ＳＤＫ。

➢ 基本异常处理

无

c) 接口调用场景说明

当用户在第三方应用的点击退出按钮或退出应用时调用该接口

h) 接口调用实例

//退出登录，和服务器断开连接

```
CMIMHelper.getCmAccountManager().doLogOut();
```

5.2 联系人模块

5.2.1 获取好友列表

a) 所属类

com.littlec.sdk.manager.CMContactManager

b) 接口说明

提供给第三方应用使用本 SDK 进行即时消息应用开发。在登录成功后初始化该类（只需初始化一次）。调用此接口获取所有好友列表。

- **方法原型**

public void loadAllContacts(CMChatListener.OnContactListener loadContactListener);

- **入参说明**

loadContactListener：OnContactListener 的对象，监听获取所有好友的一个回调，实现获取成功失败两个方法。不能为 null。

- **出参说明**

无返回值，用于获取所有好友列表，结果位于回调成功的方法中。

- **基本异常处理**

第三方实现回调方法，获取成功则调用 onSuccess 方法，参数为返回值。获取失败则调用 onFailed 的方法，参数为失败信息。

c) 接口调用场景说明

要在 CMContactManager 初始化后使用。

i) 接口调用实例

```
CMIMHelper.getCmContactManager().loadAllContacts(new OnContactListener() {
        @Override
        public void onSuccess(List<CMContact> contacts) {
        //好友通讯录列表成功加载完毕的处理代码
        }
        @Override
        public void onFailed(String failedMsg) {
        //好友通讯录列表加载中失败的处理代码，//failMsg 为失败原因
        }
});
```

5.2.2 搜索联系人

a) 所属类

com.littlec.sdk.manager.CMContactManager

b) 接口说明

提供给第三方应用使用本 SDK 进行即时消息应用开发，调用此接口按输入关键字模糊查找联系人。

- **方法原型**

public void searchContactsFromServer(String key,CMChatListener.OnContactListener onSearchContactListener)

- **入参说明**

key：模糊查找的关键字

onSearchContactListener：OnContactListener 的对象，监听搜索联系人的一个回调，实现搜索成功失败两个方法。不能为 null。

- **出参说明**

无返回值，用于按关键字模糊查询联系人，结果位于回调成功的方法中。

- **基本异常处理**

第三方实现回调方法，获取成功则调用 onSuccess 方法，参数为返回值。获取失败则调用 onFailed 的方法，参数为失败信息。

c) 接口调用场景说明

在需要按关键字搜索联系人的时候调用。

j) 接口调用实例

```
CMIMHelper.getCmContactManager().searchContactsFromServer(key,new OnContactListener() {
        @Override
        public void onSuccess(List<CMContact> contacts) {
        //联系人搜索完成时候的回调，contacts 为返回的结果列表
        }
        @Override
        public void onFailed(String failedMsg) {
        //联系人搜索失败的回调，failedMsg 为返回的结果列表
        }
});
```

5.2.3 添加好友

a) 所属类

com.littlec.sdk.manager.CMContactManager

b) 接口说明

提供给第三方应用使用本 SDK 进行即时消息应用开发。调用此接口添加好友到自己的好友列表中。

- **方法原型**

public void addContact(String userName, String reason, OnCMListener listener)

- **入参说明**

userName：String 类型，将要添加的用户名

reason：String 类型，添加信息（可以为任意字符串，比如加个好友呗）

listener：OnCMListener 的对象，监听添加好友的一个回调，实现添加成功失败两个方法。不能为 null。

- **出参说明**

无返回值，用于添加好友到自己的联系人列表中。

- **基本异常处理**

第三方实现回调方法，添加成功则调用 onSuccess 方法，无参数。添加失败则调用 onFailed 的方法，参数为失败信息。

c) 接口调用场景说明

在需要添加好友到联系人列表时调用。

k) 接口调用实例

```
CMIMHelper.getCmContactManager().addContact(contact.getUserId(), "加个好友吧",
        new OnCMListener () {
        @Override
        public void onSuccess() {
        // TODO Auto-generated method stub
        //添加请求发送成功
        }
        @Override
        public void onFailed(String failMsg) {
        // TODO Auto-generated method stub
        //添加请求发送失败
        }
});
```

5.2.4 删除好友

a) 所属类

com.littlec.sdk.manager.CMContactManager

b) 接口说明

提供给第三方应用使用本 SDK 进行即时消息应用开发。调用此接口从自己的联系人列表中删除好友。

- **方法原型**

public void deleteContact(String username ,String reason, OnCMListener listener)

- **入参说明**

userName：String 类型，将要删除的用户名

reason：String 类型，删除信息（可以为任意字符串，比如将你从好友列表中移除）

listener：OnCMListener 的对象 监听删除好友的一个回调 实现删除成功失败两个方法。不能为 null。

- **出参说明**

无返回值，用于从自己的联系人列表中删除好友。

- **基本异常处理**

第三方实现回调方法，删除成功则调用 onSuccess 方法，参数为被删除用户 Id。删除

失败则调用 onFailed 的方法，参数为失败信息。

c) 接口调用场景说明

在需要从联系人列表中删除好友时调用。

l) 接口调用实例

```
CMIMHelper.getCmContactManager().deleteContact(deleteUserId, "删除好友",
    new OnCMListener () {
    @Override
    public void onSuccess() {
    // TODO Auto-generated method stub
    //删除好友成功
    }
    @Override
    public void onFailed(String failMsg) {
    // TODO Auto-generated method stub
    //删除好友失败
    }
```

5.2.5 处理好友请求

a) 所属类

com.littlec.sdk.manager.CMContactManager

b) 接口说明

提供给第三方应用使用本 SDK 进行即时消息应用开发。在登录成功后初始化该类（只需初始化一次）。调用此接口处理收到的添加好友请求。

- **方法原型**

public void dealWithInvitation(String username , Boolean acceptOrReject, OnCMListener listener)

- **入参说明**

username：String 类型，将要处理的好友请求发送者的用户名

acceptOrReject：boolean 类型，true 表示接受好友请求，false 表示拒绝好友请求

listener：OnCMListener 的对象，监听好友请求处理的一个回调，实现成功失败两个方法。不能为 null。

- **出参说明**

无返回值，用于从处理收到的好友请求。

- **基本异常处理**

第三方实现回调方法，处理成功则调用 onSuccess 方法，无参数。处理失败则调用 onFailed 的方法，参数为失败信息。

c) 接口调用场景说明

在需要同意或者拒绝添加好友请求时时调用。

m) 接口调用实例

```
CMIMHelper.getCmContactManager().dealWithInvitation(
                                        msg.getFrom(), acceptOrReject,
                                        new OnCMListener() {
                                            @Override
```

```
                public void onSuccess() {

}

                @Override
            public void onFailed(String failMsg) {
                    // TODO Auto-generated method stub

                }
            });
```

5.2.6 根据用户名获取联系人信息

a) 所属类

com.littlec.sdk.manager.CMContactManager

b) 接口说明

提供给第三方应用使用本 SDK 进行即时消息应用开发。在登录成功后初始化该类（只需初始化一次）。调用此接口获取联系人信息。

- **方法原型**

public CMContact getContactByUserName(String userName)

- **入参说明**

username：String 类型，将要获取用户信息的用户名

- **出参说明**

返回 CMContact 类型的对象，未查询到或查询失败返回 null

- **基本异常处理**

异常返回 null

c) 接口调用场景说明

在需要获取某位好友的相关信息时调用。

n) 接口调用实例

```
CMContact cmContact= CMIMHelper .getCmContactManager.
getContactByUserName(_member);
```

5.2.7 修改自己手机号

a) 所属类

com.littlec.sdk.manager. `CMAccountManager`

b) 接口说明

提供给第三方应用使用本 SDK 进行即时消息应用开发。在登录成功后初始化该类（只需初始化一次）。调用此接口修改自己手机号。

- **方法原型**

分为 2 步：

1）传入新手机号，请求获取验证码：

public void requestUpdatePhoneNumber(String newPhoneNumber, OnCMListener updatePhoneNumListener)

2）传入验证码：

```
public void updatePhoneNumber(String code, List<String> contacts, OnCMListener updatePhoneNumListener) {
```

➢ 入参说明

newPhoneNumber：String 类型，手机号码

contacts：,List<String>类型，将要通知到的用户们的 id，可以为空 null（即通知不到别人）

updatePhoneNumListener：OnCMListener 的对象，监听添加好友的一个回调，实现添加成功失败两个方法。不能为 null，否则抛出空指针异常。

code：验证码，4 位

➢ 出参说明

无返回

➢ 基本异常处理

第三方实现回调方法，添加成功则调用 onSuccess 方法，无参数。添加失败则调用 onFailed 的方法，参数为失败信息。

c) 接口调用场景说明

在需要修改自己手机号信息时调用。

o) 接口调用实例

```
CMIMHelper.getCmAccountManager().requestUpdatePhoneNumber(newPhoneNumber,
new OnCMListener() {
        @Override
        public void onSuccess() {
                //成功返回的回调，

        //跳转到输入验证码的界面，
        }

        @Override
        public void onFailed(String errorMsg) {
            CommonUtils.toastShow(UserInfoActivity.this, errorMsg);
        }
    });
```

在界面上输入验证码后，调用如下接口：

```
CMIMHelper.getCmAccountManager().updatePhoneNumber(code, sortedUserList, new
OnCMListener() {
        @Override
        public void onSuccess() {

);

        @Override
        public void onFailed(String errorMsg) {
            CommonUtils.toastShow(VerificationActivity.this, errorMsg);
        }
    });
```

5.2.8 修改自己的昵称

a) 所属类

com.littlec.sdk.manager. CMAccountManager

b) 接口说明

提供给第三方应用使用本 SDK 进行即时消息应用开发。在登录成功后初始化该类（只需初始化一次）。调用此接口修改自己手机号。

- **方法原型**

```
public void updateNickName(String nickName,List<String> contacts,
                    OnCMListener listener)
```

- **入参说明**

nickName：String 类型，新的昵称

contacts：,(可选)，List<String>类型，将要通知到的用户们的 id，可以为 null（即通知不到别人）

listener：OnCMListener 的对象，监听添加好友的一个回调，实现添加成功失败两个方法。不能为 null，否则抛出空指针异常。

- **出参说明**

无返回

- **基本异常处理**

第三方实现回调方法，添加成功则调用 onSuccess 方法，无参数。添加失败则调用 onFailed 的方法，参数为失败信息。

c) 接口调用场景说明

在需要修改自己昵称信息时调用。

p) 接口调用实例

```
CMIMHelper.getCmAccountManager().updateNickName(this. nickName, sortedUserList, new
OnCMListener() {
                @Override
                public void onSuccess() {

);

                @Override
                public void onFailed(String errorMsg) {
                    CommonUtils.toastShow(VerificationActivity.this, errorMsg);
                }
            });
```

5.3 消息模块

5.3.1 联系人变化以及接收消息监听

a) 所属类

com.littlec.sdk.manager

b) 接口说明

提供给第三方应用使用本 SDK 进行即时消息应用开发时，用于成功登录后添加好友变化、接收消息监听。

➢ 方法原型

//添加联系人变化以及接收消息监听

public static void addListeners(CMContactListener contactlistener,CMMessageReceivedCallBack messageReceivedCallBack)

➢ 入参说明

contactChangelistener：（可选）CMContactListener 类型，此处传入 null，则无法收到好友变化的回调，否则传入相应好友变化的回调，参考调用示例。

messageReceivedCallBack：（可选）CMMessageReceivedCallBack 类型，此处传入 null，则无法接受到消息的回调，否则传入相应接收消息的回调，参考调用示例。

➢ 出参说明

无返回值，用于添加好友变化监听、添加接收消息监听等。

➢ 基本异常处理

初始化方法，无异常处理。

c) 接口调用场景说明

建议在联系人加载完后调用，因为考虑到收到消息后的处理，比如说查找消息对应的

联系人。

q) 接口调用实例

Step1：实现 CMContactListener 类，处理对方申请好友请求、被某个好友删除、对方同意自己的好友请求、被对方拒绝好友请求时的通知。

```
CMContactListener cmContactListener = new CMContactListener() {
    @Override
    public void onContactRefused(String fromUserName) {
    // 被对方拒绝好友请求时的处理代码
    }
    @Override
    public void onContactInvited(String fromUserName, String reason) {
    // 被申请加好友时的处理代码
    }
    @Override
    public void onContactDeleted(String userName) {
    // 被某个好友删除时的处理代码
    }
    @Override
    public void onContactAgreed(String fromUserName) {
    // 对方同意自己的好友请求时的处理代码
    }
    @Override
    public void onContactAdded(String userName) {
        // 通讯录增加了联系人的处理代码
    }
    public void onContactInfoUpdated (String from,String newPhoneNumber , String new
NickName) {
        // 通讯联系人变更的通知
    }
```

Step2：实现 CMMessageReceivedCallBack 类，提供接收单聊/群聊消息、群组管理相关的通知。

```
CMMessageReceivedCallBack cmMessageReceivedCallBack = new CMMessageReceivedCallBack() {
    @Override
    public void onReceivedGroupChatMessage(CMMessage message) {
    //  群聊消息收到时候的处理代码
    }
    @Override
    public void onReceivedChatMessage(CMMessage message) {
    //单聊消息收到时候的处理代码
    }
    @Override
    public void onReceivedCreateGroupMessage(CMMessage message, CMGroup cmGroup) {
    //XXX 新建群的消息处理代码
    }
    @Override
    public void onReceivedExitGroupMessage(CMMessage message, String groupId) {
    //XXX 退群消息处理代码
    }

    @Override
    public void onReceivedKickMemberMessage(CMMessage message, String groupId, String kickedId)
{//待开发
    //群成员被删除的处理代码
    }
@Override
    public abstract void onReceivedSetGroupNameMessage(CMMessage message, String groupId,
String newName){
//群名称变更消息
}
    @Override
    public void onReceivedAddMembersMessage(CMMessage message, String groupId,
List<CMMember> addedCMMembers) {
    //群成员被 XXX 添加的处理代码
    }
    public void onReceivedSystemMessage(SystemMessage message) {
    // TODO Auto-generated method stub
//接收系统通知
            }
};
```

step3 调用监听接口

```
CMIMHelper.addListeners(cmContactListener,cmMessageReceivedCallBack);
```

5.3.2 发送消息

a) 所属类

com.littlec.sdk.manager.CMMessageManager

b) 接口说明

用来发送消息，包括单聊和群聊，类型包括文本、语音、图片、视频。消息发送失败的话，会在后台自动重发，每次间隔 10s，总共持续 5 分钟

- **方法原型**

public void sendMessage(CMMessage message, CMCallBack callback)

- **入参说明**

message: CMMessage 类型的对象，成员变量包括

成员变量	类型	描述
packetId	String	消息在网络传输中的 id
From	String	消息的发送者 id
to	String	消息的接受者 id
sendOrRecv	int	发送还是接受

time	long	接收消息时的消息时间
contentType	int	消息内容的类型
chatType	int	单聊还是群聊
Status	Int	消息状态（已发送、正在发送等）
messageBody	MessageBody	文本、语音、图片消息正文

会话类型chatType：

```
MessageConstants.Conversation.TYPE_SINGLE/*单聊消息*/
```

消息状态 status:

```
MessageConstans.Message.STATUS_DRAFT      /*草稿*/
MessageConstans.Message.STATUS_SENDING /*发送中*/
MessageConstans.Message.STATUS_SENT    /*已发送*/
MessageConstans.Message.STATUS_SEND_FAIL /*发送失败*/
MessageConstans.Message.STATUS_RECVING    /*接受中*/
MessageConstans.Message.STATUS_RECVED     /*接收完成*/
MessageConstans.Message.STATUS_RECV_FAIL    /*接收失败*/
```

发送还是接收 sendOrRecv

```
MessageConstans.Message.MSG_SEND    /*表示是发送的消息*/
MessageConstans.Message.MSG_RECV    /*表示是接收到的消息*/
```

- **出参说明**

在回调里有成功、失败、进行中三种状态的回馈

- **基本异常处理**

入参 callback 为 null 时，抛出空指针异常

c) 接口调用场景说明

在发送文本，语音、图片、视频消息时用到。

语音格式支持 amr 和 mp3，语音发送建议最大 180S；

图片格式支持 jpg、jpeg、bmp、png，文件大小最大为 10M；

视频格式支持 mp4，文件大小最大为 10M。

r) 接口调用实例

Step1 生成消息类型 CMMessage 的对象

发送文本时：
CMMessage message = new CMMessage(chatType, mCurrentRecipient, new TextMessageBody(text));// text 为文本内容，String 类型，不能为空

发送语音时：
CMMessage message = new CMMessage(chatType, mCurrentRecipient, new AudioMessageBody(new File(filePath), duration));// filePath 为本地文件路径文件, String 类型，不能为空; duration 为语音时长，单位 s，int 类型

发送图片时：
CMMessage message = new CMMessage(chatType, mCurrentRecipient, new ImageMessageBody(new File(filePath))); // filePath 为本地文件路径文件, String 类型，不能为空; 图片类型支持 png、jpg、jpeg、 bmp 格式

发送视频时：
CMMessage message = new CMMessage(chatType, mCurrentRecipient, new VideoMessageBody(new File(filePath), duration)); // filePath 为本地文件路径文件, String 类型，不能为空; duration 为视频时长，单位 s，int 类型
发送@消息时：
CMMessage message = new CMMessage(MessageConstants.Conversation.TYPE_GROUP, mCurrentRecipient,
new AtTextMessageBody(text, userList));// text 为文本内容，String 类型，不能为空; userList 为@的人员 list, List<String>类型

发送位置消息：
CMMessage message = new CMMessage(chatType, mCurrentRecipient, new LocationMessageBody(latitude, longitude, address, new File(screenShotFilePath)));
// latitude 为纬度，-90~90，longitude 为经度，-180~180，double 类型；address 为地址，String 类型，不能为空；screenShotFilePath 为本地文件路径文件, String 类型，不能为空; 图片类型支持 png、jpg、jpeg、 bmp 格式

Step2 生成回调类 CMCallBack 的对象为

```
CMCallBack callBack =new CMCallBack() {
        @Override
        public void onSuccess(CMMessage message) {
        //发送消息成功的处理
        }
        @Override
        public void onError(CMMessage message ，String errorString) {
        //发送消息失败的处理
        }
        @Override
        public void onProgress(CMMessage message, int progress) {
        //发送消息过程中
            if (progress == CMMessage.PREPARED) {
//准备好了发送
            } else {
            if (progress != message.getProgress()) {
发送进度状态更新
        }
};
```

Step3 调用发送消息的接口

```
CMIMHelper.getCmMessageManager().sendMessage(message, callBack);
```

5.4 群组管理模块

5.4.1 获取群列表

a) 所属类

com.littlec.sdk.manager.CMGroupManager

b) 接口说明

提供给第三方应用使用本 SDK 进行即时消息应用开发。调用此接口获取所在群列表。

- **方法原型**

public void getGroupListFromServer(OnGroupListener listener)；

- **入参说明**

listener：OnGroupListener 的对象，监听获取所在群的一个回调，实现获取成功失败两个方法。不能为 null，否则抛出空指针异常。

- **出参说明**

无返回值，用于获取所在群列表，结果位于回调成功的方法中。

- **基本异常处理**

第三方实现回调方法，获取成功则调用 onSuccess 方法，参数为返回值。获取失败则调用 onFailed 的方法，参数为失败信息。

c) 接口调用场景说明

在需要获取群列表的地方调用。

s) 接口调用实例

```
CMIMHelper.getCmGroupManager().getGroupListFromServer(
        new OnGroupsListener() {
        @Override
        public void onSuccess(List<CMGroup> groups) {
        // TODO Auto-generated method stub
        //获取群列表成功的处理，groups 为拥有的群列表
        }
        @Override
        public void onFailed(String failedMsg) {
        // TODO Auto-generated method stub
        //获取群列表失败的处理，failedMsg 为失败原因
        }
});
```

5.4.2 创建群

a) 所属类

com.littlec.sdk.manager.CMGroupManager

b) 接口说明

用来创建一个群，群成员最大为 100 人

- **方法原型**

CMIMHelper.getCmGroupManager().createGroup(String groupName, String describe, String ownerNick, HashMap<String,String> memberNickMap)

- **入参说明**

参数	类型	描述
groupName	String	群名称，不能超过 50 个字
describe	String	群描述
ownerNick	String	群主的昵称
memberNickMap	HashMap<String,String>	除自己之外的其他成员对应的 id 和昵称

- **出参说明**

返回 CMgroup，包括群 ID，群名称等群的信息

- **基本异常处理**

入参为 null 时，抛出空指针异常

c) 接口调用场景说明

在需要创建群聊时用到

t) 接口调用实例

```
try {
    // 调用创建群接口
HashMap<String, String> membersMap =new HashMap<String, String>();
      for(String memberId : members) {
      membersMap.put(memberId, memberNick);
      }
      CMGroup cmGroup =
      CMIMHelper.getCmGroupManager().createGroup(groupName,groupName, nick,
      membersMap);
    }
    catch(Exception e) {
    //创建过程出现异常
    e.printStackTrace();
    }
```

5.4.3 添加群成员

a) 所属类

com.littlec.sdk.manager.CMGroupManager

b) 接口说明

用来邀请其他成员到已存在的该群中，邀请人员数需要控制群的总人员数小于 100 人

- **方法原型**

public void inviteMembersToGroup(String groupId, HashMap<String, String> memberNickMap)

入参说明

参数	类型	描述

groupId	String	群 ID
memberNickMap	HashMap<String, String>	要添加的成员 map,对应 id 和昵称

- **出参说明**

无返回

- **基本异常处理**

入参为 null 时，抛出空指针异常

c) 接口调用场景说明

在群里需要邀请其他人员时用到

u) 接口调用实例

```
try {
	HashMap<String, String> map = new HashMap<String, String>();
				for(String memberId : addedMembers) {
					map.put(memberId, memberNick);
				}
	//将 addedMembers 中成员的添加到群 ID 为 currentGroup.getGroupId()的群组
	CMIMHelper.getCmGroupManager().inviteMembersToGroup(currentGroup.getGroupId(),
			map);
}
	catch(Exception e) {
	// TODO Auto-generated catch block
	//添加过程中出现了异常
	e.printStackTrace();
}
```

5.4.4 退出群

a) 所属类

com.littlec.sdk.manager.CMGroupManager

b) 接口说明

用来退出一个群

- **方法原型**

public void exitGroup(String groupId)

入参说明

参数	类型	描述
groupId	String	群 ID

- **出参说明**

无返回

- **基本异常处理**

入参为 null 时，抛出空指针异常

c) 接口调用场景说明

在需要退出群聊时用到

v) 接口调用实例

```
try {
    //退出指定群 ID 的群
    CMIMHelper.getCmGroupManager().exitGroup(currentGroup.getGroupId());
    }
    catch(Exception e) {
    // TODO Auto-generated catch block
    //退出群过程中出现了异常
    e.printStackTrace();
}
```

5.4.5 获取群名称

a) 所属类

com.littlec.sdk.manager.CMGroupManager

b) 接口说明

用来获取群名称

- **方法原型**

public String getGroupNamefromServer(String groupId)

入参说明

参数	类型	描述
groupId	String	群 ID

- **出参说明**

返回群名称,获取失败时，返回 null

- **基本异常处理**

c) 接口调用场景说明

在需要获取群聊的名称时用到

w) 接口调用实例

```
//根据群 ID 为 groupId 的群名称
CMIMHelper.getCmGroupManager().getGroupNamefromServer(groupId);
```

5.4.6 获取群成员

a) 所属类

com.littlec.sdk.manager.CMGroupManager

b) 接口说明

用来创建一个群

- **方法原型**

public ArrayList< `CMMember` > getGroupMembersfromServer(String groupId)

入参说明

参数	类型	描述
groupId	String	群 ID

- **出参说明**

返回成员列表，获取失败时，返回 null

- **基本异常处理**

c) 接口调用场景说明

在需要获取群聊里所有成员时

x) 接口调用实例

```
//获取群 ID 为 groupId 的群成员列表
CMIMHelper.getCmGroupManager().getGroupMembersfromServer(groupId);
```

5.4.7 修改群名称

a) 所属类

com.littlec.sdk.manager.CMGroupManager

b) 接口说明

用来获取群名称

- **方法原型**

public void changeGroupName (String groupId, String newGroupName)

入参说明

参数	类型	描述
groupId	String	群 ID，不为空
newGroupName	String	新的群成名，不为空

- **出参说明**

无返回

- **基本异常处理**

出错时抛出异常

c) 接口调用场景说明

在需要修改群聊的名称时用到

y) 接口调用实例

```
//根据群 ID 为 groupId 的群名称
CMIMHelper.getCmGroupManager().changeGroupName (groupId,new
GroupName);
```

六、参考资料

Xmpp 协议，参见网站 http://xmpp.org/

七、附录

消息类型：

MessageConstants.Message.TYPE_TEXT/*文本消息*/

MessageConstants.Message.TYPE_EMOTION/*表情消息*/

MessageConstants.Message.TYPE_NOTIFY/*通知消息*/

MessageConstants.Message.TYPE_AUDIO/*语音消息*/

MessageConstants.Message.TYPE_PIC/*图片消息*/

MessageConstants.Conversation.TYPE_SINGLE/*单聊消息*/

MessageConstants.Conversation.TYPE_GROUP/*群聊消息*/

Emoji 表情的图片各 SDK 可以自己的美工去作图实现，参考的图标位于 demo 工程下