

SDK 开发指南

—Android

日期	操作人	对应 sdk 版本	备注
2014/06/9	simon	V1.0.0	初稿
2014/06/29	simon	V1.0.20140629	1、重新整理并添加部分缓存操作 2、增加自动登录、找回密码的 API 接口 3、调整系统监听中关于获取群组 and 联系人的接口
2014/07/14	simon	V1.0.20140714	4、修改了系统初始化的方式 5、对个人会话做了更详细的描述 6、增加了查找联系人编号的方法
2014/08/11	simon	V1.0.20140811	7、增加了呼叫业务 8、增加了一些缓存操作。 9、增加通过帐号来发送消息的方法。
2014/08/22	simon	V1.0.20140822	10、暂时将导入 jar 包合并为一个。 11、完善退出的方法，区分注销和退出程序。 12、更新修改用户密码的方法 13、更多添加联系人的方法 14、增加系统监听接口
2014/09/07	simon	V1.0.20140907	15、增加操作登录服务器地址的 api 16、增加重发消息的 API
2014/12/24	simon	V1.1.20141224	17、初始化的时候增加后台服务的配置 18、修改了部分系统监听的接口 19、增加了一些缓存操作，例如获取扩展应用列表数据等 20、增加扩展应用相应功能 API
2015/01/01	simon	V1.1.20150101	21、增加接收广播消息的功能
2015/1/20	simon	V1.2.0.90	22、增加漫游消息的功能 23、修改当程序处于后台运行时，有些消息没有系统通知的问题 24、修复与离线游客会话时抛出异常的问题 25、优化我的消息，将只需要提示的一些群消息分离出来，单独区分为系统通知 26、解决系统通知时，表情显示不正常的问题，将显示表情的描述文字

			27、优化网络异常后重新登录的问题 28、新增添加增加个人群组管理功能，包括创建个人群组和讨论组；邀请人进群，退群，移除成员，解散群组等功能；
2015/02/12	simon	V1.2.1.94	29、解决动态 IP 无法验证 APPKEY 的问题 30、优化我的群组、个人群组、联系人界面显示，群组增加成员的直接展示 31、修复修改成员头像后，再次登录没有更新的问题 32、增加好友模式的判断和展示 33、增加好友的邀请、删除等功能 34、增加好友分组的管理功能
2015/03/31	simon	V1.3.0.100	35、文件收发 36、增加修改联系人分组名称的功能 37、修改企业部门和成员的显示方式 38、补充联系人信息展示
2015/06/10	Simon	V1.4.0.110	39、修复帐号不能识别前面带“@”的帐号 40、修复不能加载个人新加入/被删除的部门或项目组 41、优化群组操作的同步刷新

1. 前言.....	6
2. 准备工作.....	6
2.1. 申请应用的appKey和appSecret.....	6
2.2. 导入SDK包	6
2.3. 配置AndroidManifest.....	6
2.3.1. 配置用户权限（重要）	6
3. 在代码中使用.....	7
3.1. 初始化信息.....	7
3.2. 配置系统参数.....	7
3.3. 注册用户.....	8
3.3.1. 普通用户（邮箱）注册.....	8
3.4. 登录授权及退出（重要）	9
3.4.1. 登录授权.....	9
3.4.2. 找回密码.....	11
3.4.3. 退出登录.....	12
3.5. 用户个人信息.....	12
3.5.1. 获取用户个人信息.....	12
3.5.2. 编辑个人信息.....	12
3.5.3. 更新用户名称.....	13
3.5.4. 修改用户密码.....	13
3.5.5. 更新个人备注信息.....	14
3.6. 呼叫操作.....	21

3.6.1.	呼叫用户	21
3.6.2.	呼叫缓存	21
3.6.3.	响应呼叫	22
3.6.4.	呼叫监听	22
3.7.	个人会话操作	23
3.7.1.	查找联系人ID	23
3.7.2.	发送文本消息	23
3.7.3.	发送表情	23
3.7.4.	发送图片	24
3.7.5.	发送语音	25
3.7.6.	接收消息	26
3.7.7.	聊天记录	27
3.8.	群组会话操作	27
3.8.1.	发送文本消息	28
3.8.2.	发送表情	28
3.8.3.	发送图片	28
3.8.4.	发送语音	29
3.9.	动态信息	29
3.9.1.	获取动态历史信息	29
3.9.2.	读取动态信息	30
3.9.3.	清空所有动态信息	30
3.10.	联系人信息	30

3.10.1.	添加联系人.....	30
3.10.2.	编辑联系人.....	31
3.10.3.	删除联系人.....	33
3.10.4.	移动联系人到其它分组.....	33
3.11.	缓存操作.....	34
3.12.	接收监听.....	41
3.12.1.	注册与取消监听.....	41
3.12.2.	监听详细说明.....	42

1. 前言

说明文档适用于所有合作伙伴的安卓版客户端应用，在接入之前，请确认您已在恩布开放平台（<http://www.entboost.com/>）注册，并已获得 App Key, App Secret。如还未获取，请参考业务文档。

2. 准备工作

2.1. 申请应用的 appKey 和 appSecret

注册申请页面 <http://www.entboost.com/register>

2.2. 导入 SDK 包



- 将获取的工具包解压，提取出 libs 文件夹下的 jar 包
- 将获取到的 jar 包导入到您工程目录下的 libs 文件夹

注意:

使用'ADT 17'以下用户需要手动添加'libs 下的 jar 文件到工程 Path 中；

使用'ADT 21'以上的用户需要在您的工程目录下选择 Properties->Java Build Path->Order and Export，勾选 Android Private Libraries 和 Android Dependencies，可消除 classnotfoundexception；

2.3. 配置 AndroidManifest (重要)

2.3.1. 配置用户权限

在工程 AndroidManifest.xml 中配置用户权限

请将下面权限配置代码复制到 AndroidManifest.xml 文件中：

```
<!-- 允许应用程序联网 -->
<uses-permission android:name="android.permission.INTERNET" />
<!-- 检测网络状态 -->
```

```

<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<!-- 获取用户手机的IMEI，用来唯一的标识用户 -->
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<!-- 缓存资源优先存入SDcard -->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<!-- 读取SDcard内容 -->
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<!-- 记录语音 -->
<uses-permission android:name="android.permission.RECORD_AUDIO" />

```

2.3.2. 配置后台服务

在 *AndroidManifest.xml* 文件的 `application` 节点下配置：

```
<service android:name="net.yunim.service.EbClientService" ></service>
```

3. 在代码中使用

3.1. 初始化信息

初始化和配置信息设置的操作，最好放在 `application` 中。

```

public class MyApplication extends Application {
    @Override
    public void onCreate() {
        super.onCreate();
        //TODO 系统相关初始化信息放入此处
    }
}

```

1) . 初始化恩布即时聊天的系统资源

```
Entboost.init(getApplicationContext());
```

3.2. 配置系统参数

1) 配置私有通讯云服务器地址

```
EntboostLC.setLogonCenterAddr(logonCenterAddr);
```

参数说明：

`logonCenterAddr`：服务器地址，格式www.entboost.com:8080

该参数将保存到本地缓存，下次登录将使用缓存中保存的参数，如果未设置该参数，将使用默认的公众服务器地址：`entboost.entboost.com:18012`

```
EntboostLC.getLogonCenterAddr ();
```

获取登录服务器地址

2) . 设置是否显示调试日志

```
Entboost.showSotpLog(true);
```

3.3. 注册用户

3.3.1. 普通用户 (邮箱) 注册

```
/**
 * 普通用户邮箱注册，需要邮箱验证激活帐号
 *
 * @param app_id
 *         开发者ID
 * @param apppwd
 *         开发者密钥
 * @param email
 *         邮箱地址
 * @param sPwd
 *         帐号密码
 * @param listener
 */
EntboostUM.emailRegister(app_id, apppwd, email, sPwd, new RegisterListener(){
    @Override
    public void onFailure(String errMsg) {
        //TODO 注册失败
    }
    @Override
    public void onRegisterSuccess() {
        //TODO 注册成功
    }
});
```

注册后需要注册用户去注册邮箱中激活验证。

3.3.2. 企业用户 (邮箱) 注册

```
/**
 * 企业用户邮箱注册，需要邮箱验证激活帐号，同一企业只能注册一个帐号，其它成员由管理员添加
```

```
 *
 * @param app_id
 *         开发者ID
 * @param apppwd
 *         开发者密钥
 * @param email
 *         邮箱地址
 * @param sPwd
 *         帐号密码
 * @param sEnterpriseName
```



```
*          企业名称
* @param listener
*/

EntboostUM.emailRegister(app_id, apppwd, email, sPwd, sEnterpriseName, new
RegisterListener(){
    @Override
    public void onFailure(String errMsg) {
        //TODO 注册失败
    }
    @Override
    public void onRegisterSuccess() {
        //TODO 注册成功
    }
});
```

3.4. 登录授权及退出 (重要)

3.4.1. 开发者授权

1、开发者授权校验

```
EntboostLC.initAPPKey(appid, appkey, new InitAppKeyListener() {
    @Override
    public void onFailure(String arg0) {

    }
    @Override
    public void onFailure(String arg0) {

    }
});
EntboostLC.initAPPKey(addr,appid, appkey, new InitAppKeyListener() {
    @Override
    public void onFailure(String arg0) {

    }
    @Override
    public void onFailure(String arg0) {

    }
});
```

Addr表示注册的服务器地址，不填表示注册到默认或本地保存的服务器地址

注册开发者appkey, appid的数据类型为Long,数字后面要加上1, 例如874562130982+1

2、缓存中获取开发者授权对象

```
EntboostCache.getAppInfo()
```

AppAccountInfo 对象

这个对象除了保存一些开发者授权相关信息，例如 `appid`, `appkey`, `app_online_key`, `appAddress` 等，还保存了一些应用的配置信息，这些信息在服务器后台配置管理中设置。

2.1、开放注册

```
open_register
// 0 [客户端应用]不开放注册
// 1: 开放注册用户
// 2: 开放注册企业
// 3: 开放注册用户+注册企业
```

2.2、开放游客

```
open_visitor
// 0 [客户端应用]开放游客 0/1
```

2.3、企业 LOGO URL

```
ent_logo_url
```

2.4、保存聊天记录配置

```
save_conversations
public static final int SAVE_CONVERSATIONS_LOCAL=1; // 只保存本地
public static final int SAVE_CONVERSATIONS_ONLINE=2; // 只保存服务器
public static final int SAVE_CONVERSATIONS_ALL=3; // 保存本地和服务器
conversations_url
//漫游消息 url
```

2.5、是否发送注册验证邮件

```
send_reg_mail
// 0 [客户端应用]是否发送注册验证邮件 0/1
```

3.4.2. 登录 IM

1.普通用户登录

```
/**
 * 用户帐号登录系统,默认在线登录
 *
 * @param account
 *         用户名称
 * @param pwd
 *         用户密码
 * @param logonAccountListener
 *         登录的回调函数
 */
EntboostLC.Logon(account,pwd, new LogonAccountListener() {
    @Override
    public void onFailure(String errMsg) {
        //TODO 登录失败
    }
    @Override
    public void onLogonSuccess(AccountInfo pAccountInfo) {
```

```

        //TODO 登录成功
    }
});

/**
 * 用户帐号自动登录系统,默认在线登录（默认用户是最后一次登录的用户）
 *
 * @param logonAccountListener
 *      登录的回调函数
 */
EntboostLC.Logon(new LogonAccountListener() {
    @Override
    public void onFailure(String errMsg) {
        //TODO 登录失败
    }
    @Override
    public void onLogonSuccess(AccountInfo pAccountInfo) {
        //TODO 登录成功
    }
});

```

2.访客登录

```

/**
 * 访客登录
 * @param logonAccountListener 登录的回调函数
 */
EntboostLC.LogonVisitor(new LogonAccountListener() {
    @Override
    public void onFailure(String errMsg) {
        //TODO 登录失败
    }
    @Override
    public void onLogonSuccess(AccountInfo pAccountInfo) {
        //TODO 登录成功
    }
});

```

3.4.3. 删除历史登录记录

```

/**
 * 删除历史登录记录
 * @param account
 */
EntboostLC.delAccountHistory(String account)

```

3.4.4. 找回密码

```

/**

```

```
    * 用户找回密码，输入注册邮箱、手机号码或用户ID，发送重设密码到用户注册邮箱上
    *
    * @param account
    *         注册邮箱、手机号码或用户ID（目前只支持注册邮箱）
    * @param listener
    *         找回密码的回调函数
    */
EntboostLC.findPwd(account,new FindPWDListener() {

    @Override
    public void onFailure(String errMsg) {
        //TODO 找回密码失败
    }

    @Override
    public void onFindPWDSuccess() {
        //TODO 找回密码成功
    }
});
```

3.4.5. 退出登录

代码示例

```
EntboostLC.logout();
```

说明：调用此函数，注销当前用户登录并保存当前用户信息到本地；一般用于注销用户。

```
EntboostLC.exit();
```

说明：调用此函数，退出用户登录、保存当前用户信息到本地、退出线程、退出程序等；

一般用于退出程序。

3.5. 用户个人信息

3.5.1. 获取用户个人信息

```
EntboostCache.getUser(); //获取保存在本地的账户信息
```

```
EntboostCache.getUid() //获取保存在本地的帐号ID
```

3.5.2. 编辑个人信息

```
/**
 * 编辑个人信息
 *
 * @param sNewUserName 用户名称
```

```
    * @param sNewDescription 备注信息
    * @param listener 编辑监听
    */
EntboostUM.editUserInfo(sNewUserName, sNewDescription, new EditInfoListener() {

    @Override
    public void onFailure(String errMsg) {
        //TODO 修改失败
    }

    @Override
    public void onEditInfoSuccess() {
        //TODO 修改成功
    }

});
```

3.5.3. 更新用户名称

```
/**
 * 修改用户名称
 *
 * @param sNewUserName 用户名称
 * @param listener 编辑监听
 */
EntboostUM.changeUserName(sNewUserName, new EditInfoListener() {

    @Override
    public void onFailure(String errMsg) {
        //TODO 修改失败
    }

    @Override
    public void onEditInfoSuccess() {
        //TODO 修改成功
    }

});
```

3.5.4. 修改用户密码

```
/**
 * 修改密码
 *
 * @param sNewPassword 新密码
 * @param oldPassword 旧密码
 * @param listener 编辑监听
```

```

*/
EntboostUM.changePassword(sNewPassword, oldPassword, new EditInfoListener() {

    @Override
    public void onFailure(String errMsg) {
        //TODO 修改失败
    }

    @Override
    public void onEditInfoSuccess() {
        //TODO 修改成功
    }
});

```

3.5.5. 更新个人备注信息

```

/**
 * 更新帐号备注
 *
 * @param sNewDescription 备注信息
 * @param listener 编辑监听
 */
EntboostUM.setDescription(sNewDescription, new EditInfoListener() {

    @Override
    public void onFailure(String errMsg) {
        //TODO 修改失败
    }

    @Override
    public void onEditInfoSuccess() {
        //TODO 修改成功
    }
});

```

3.5.6. 设置个人的默认部门编号

```

/**
 * 设置个人的默认部门编号
 *
 * @param sNewDefaultMemberCode
 *         默认群组（部门）编号，用于在外部呼叫系统自动生成电子名片
 */
EntboostUM.setMyDefaultMemberCode(sNewDefaultMemberCode, new EditInfoListener(){

    @Override
    public void onFailure(String errMsg) {
        //TODO 修改失败
    }
});

```

```

    }
    @Override
    public void onEditInfoSuccess() {
        //TODO 修改成功
    }
});

```

3.6. 个人群组

3.6.1. 添加个人群组

```
EntboostUM.addPersonGroup(PersonGroupInfo group,
    EditGroupListener listener)
```

`PersonGroupInfo` 个人群组对象

`EditGroupListener` 添加个人群组的回调监听

3.6.2. 邀请加入个人群组

```
EntboostUM.addToPersonGroup(Set<Long> dep_codes, Long emp_uid, final
    AddToPersonGroupListener listener)
```

`dep_codes` 群组编号集合

`emp_uid` 邀请的用户编号

`AddToPersonGroupListener` 邀请加入的回调监听

```
EntboostUM.addToPersonGroup(Long dep_code, Set<Long> emp_uids,
    final AddToPersonGroupListener listener)
```

`dep_code` 群组编号

`emp_uids` 邀请的用户编号集合

`AddToPersonGroupListener` 邀请加入的回调监听

3.6.3. 移除个人群组

```

/**
 * 删除群组成员信息
 *
 * @param member
 *         成员对象
 * @param listener
 */
EntboostUM.delGroupMember(MemberInfo member,
    final DelMemberListener listener)

```

3.6.4. 解散个人群组

```

/**
 * 解散个人群组
 *
 * @param dep_id

```

```
        *          群组编号
        * @param listener
        */
EntboostUM.delPersonGroup(Long dep_id, DelGroupListener listener)
```

3.7. 部门

3.7.1. 企业资料编辑

```
/**
 * 企业资料编辑
 *
 * @param phone
 *          电话
 * @param fax
 *          传真
 * @param email
 *          企业邮箱
 * @param url
 *          企业网站
 * @param address
 *          企业地址
 * @param description
 *          企业描述
 * @param listener
 */
EntboostUM.updateEnterprise(phone, fax, email,url, address, description,new
EditEntListener()){
    @Override
    public void onFailure(String errMsg) {
        //TODO 修改失败
    }
    @Override
    public void onEditEntSuccess() {
        //TODO 修改成功
    }
})
```

3.7.2. 添加企业部门

```
/**
 * 添加新的企业部门
 *
 * @param dep_name
 *          部门名称
```



```
* @param parent_code
*      上级部门编号（为空表示添加顶级部门）
* @param phone
*      部门电话
* @param fax
*      部门传真
* @param email
*      部门邮箱
* @param url
*      部门网址
* @param address
*      部门地址
* @param description
*      部门描述
* @param listener
*/
EntboostUM. addDepartment(dep_name, parent_code,
    phone, fax, email, url, address,
    description, new EditGroupListener(){
        @Override
        public void onFailure(String errMsg) {
            //TODO 修改失败
        }
        @Override
        public void onEditGroupSuccess(Long dep_code) {
            //TODO 修改成功
        }
    });
```

3.7.3. 编辑企业部门

```
/**
 * 编辑企业部门信息
 *
 * @param dep_id
 *      部门编号
 * @param dep_name
 *      部门名称
 * @param parent_code
 *      上级部门编号（为空表示添加顶级部门）
 * @param phone
 *      部门电话
 * @param fax
 *      部门传真
 * @param email
 *      部门邮箱
```

```
    * @param url
    *          部门网址
    * @param address
    *          部门地址
    * @param description
    *          部门描述
    * @param listener
    */
EntboostUM. editDepartment(dep_id ,dep_name, parent_code,
    phone, fax, email, url, address,
    description,new EditGroupListener(){
        @Override
        public void onFailure(String errMsg) {
            //TODO 修改失败
        }
        @Override
        public void onEditGroupSuccess(Long dep_code) {
            //TODO 修改成功
        }
    });
```

3.7.4. 删除企业群组

```
/**
 * 删除企业群组（群组中还有成员则无法删除）
 *
 * @param dep_id
 *          群组编号
 * @param listener
 */
EntboostUM. delDepartment(dep_id,new DelGroupListener(){
    @Override
    public void onFailure(String errMsg) {
        //TODO 删除失败
    }
    @Override
    public void onDelGroupSuccess(Long dep_code) {
        //TODO 删除成功
    }
});
```

3.8. 部门成员

3.8.1. 添加部门成员

```
/**
```

```

    * 添加新的部门雇员信息，同时创建雇员账号，如果是邮箱注册会发送注册邮件以便
    * 激活帐号，目前只支持邮箱注册。
    *
    * @param dep_code
    *         部门编号
    * @param emp_account
    *         部门成员帐号（邮箱、电话等）
    * @param username
    *         成员名称
    * @param listener
    */
EntboostUM. addEmployer (dep_code, emp_account, username,new EditMemberListener (){
    @Override
    public void onFailure(String errMsg) {
        //TODO 修改失败
    }
    @Override
    public void onEditMemberSuccess(Long dep_code) {
        //TODO 修改成功
    }
});

```

3.8.2. 编辑部门成员

```

/**
 * 编辑部门雇员信息
 *
 * @param dep_code
 *         部门编号
 * @param emp_code
 *         成员编号
 * @param username
 *         成员名称
 * @param gender
 *         性别
 * @param birthday
 *         生日
 * @param job_title
 *         职称
 * @param job_position
 *         职位
 * @param cell_phone
 *         电话
 * @param fax
 *         传真
 * @param work_phone

```

```

*          工作电话
* @param email
*          邮箱
* @param address
*          地址
* @param description
*          描述
* @param manager_level 参考EB_MANAGER_LEVEL枚举类
*          管理级别
* @param listener
*/
EntboostUM. editEmployer (dep_code, emp_code, username, gender, birthday, job_title,
job_position, cell_phone, fax, work_phone, email, address, description,
manager_level, new EditMemberListener () {
    @Override
    public void onFailure(String errMsg) {
        //TODO 修改失败
    }
    @Override
    public void onEditMemberSuccess(Long dep_code) {
        //TODO 修改成功
    }
});
```

3.8.3. 删除部门成员

```

/**
* 删除部门成员信息，并删除成员账号
*
* @param emp_code 成员编号
* @param listener
*/
EntboostUM. delEmployer (emp_code, new DelMemberListener () {
    @Override
    public void onFailure(String errMsg) {
        //TODO 删除失败
    }
    @Override
    public void onDelMemberSuccess() {
        //TODO 删除成功
    }
});
```

3.8.4. 成员调转部门

```

/**
* 移动成员到另一个部门
```

```

*
* @param dep_code
*         部门编号
* @param emp_code
*         成员编号
* @param listener
*/
EntboostUM.moveEmployer (dep_code, emp_code, new EditMemberListener () {
    @Override
    public void onFailure(String errMsg) {
        //TODO 修改失败
    }
    @Override
    public void onEditMemberSuccess(Long dep_code) {
        //TODO 修改成功
    }
});

```

3.9. 呼叫操作

会话之前需要呼叫对方，同企业、同群组的用户在接收到呼叫后，会自动接听。联系人之间接收到呼叫也可以自动接听，这些都可以通过个人设置来配置。默认是全部自动接听。

3.9.1. 呼叫用户

```

/**
 * 呼叫用户
 *
 * @param uid
 *         呼叫对方的ID
 * @param listener
 *         呼叫监听
 */
EntboostCM.callUser(Long uid, CallUserListener listener)

/**
 * 呼叫用户
 *
 * @param account
 *         呼叫对方的帐号
 * @param listener
 *         呼叫监听
 */
EntboostCM.callUser(String account, CallUserListener listener)

```

3.9.2. 呼叫缓存

呼叫之后，呼叫信息会保存到缓存，用户可以通过缓存来处理呼叫。

首先在历史动态信息中有记录呼叫的信息。

`dynamicNews.getType() == DynamicNews.TYPE_CALL` //表示动态信息的类型是呼叫

获取所有的呼叫信息列表

```
/**
 * 获取呼叫信息列表
 * @return Vector<CardInfo>
 */
EntboostCache.getCallCards()
```

3.9.3. 响应呼叫

```
/**
 * 响应呼叫
 * @param call_id 呼叫的编号
 * @param to 呼叫对方编号
 * @param bAccept 是否接收呼叫
 */
EntboostCM.callAnswer(Long call_id, Long to, boolean bAccept)
```

3.9.4. 呼叫监听

监听的实现方式请参考系统监听部分。

```
/**
 * 有呼叫呼入事件处理
 */
public abstract void onCallIncoming(CardInfo card);

/**
 * 收到用户拒绝会话处理
 *
 * @param callInfo
 */
public abstract void onCallReject(CardInfo card);

/**
 * 收到用户挂断会话处理
 */
public abstract void onCallHangup(Long to);

/**
 * 收到会话超时处理
 *
 * @param callInfo
 */
public abstract void onCallTimeout(CardInfo card);

/**
```

```
* 当用户加入会话时处理
*
* @param connectInfo
*/
public abstract void onCallConnected(Long to);
```

3.10. 个人会话操作

3.10.1.查找联系人 ID

```
/**
 * 查询联系人帐号，获取用户UID
 *
 * @param account
 *         帐号
 * @param listener
 */
EntboostUM.queryContact(account,listener);
```

所有发送操作都需要先获取发送对象的 ID，该方法为通过账号信息查找 ID。

3.10.2.发送文本消息

```
/**
 * 给个人发送文本信息
 *
 * @param sTo
 *         发送对方的ID
 * @param text
 *         文本信息
 */
EntboostCM.sendText (sTo, text);
```

```
/**
 * 给个人发送文本信息
 *
 * @param account
 *         发送对方的账号
 * @param text
 *         文本信息
 */
EntboostCM.sendText (account, text);
```

一般来说发送对方的ID，在登陆后就会获取到群组成员信息，这些信息里面会有ID。
如果是联系人，在没有ID信息的情况下，可以调用查找联系人ID的方法来获取。

3.10.3.发送表情

发送表情首先要获取所有的表情资源，提供展示界面，方便在输入信息时选择。
登陆后会自动加载表情资源信息，用户在需要界面展示的时候，可以从缓存中获取表情资源列表。

`Resource`是表情资源对象，从中我们可以获取资源对象的编号，来得到资源图片的对象。

```
/**
 * 获取所有的表情资源
 *
 * @return Vector<Resource> 返回表情资源对象的集合
 */
EntboostCache. getEmotionslist();
/**
 * 通过表情资源编号获取表情图片的Bitmap对象
 * @param rid 表情资源编号,通过Resource对象来获取 resource.getRes_id()
 * @return
 */
EmotionUtils.getEmotionBitmap(Long rid)
在展示在界面上的时候，我们往往使用ImageView来加载表情
ImageView.setImageBitmap(EmotionUtils.getEmotionBitmap(rid));
```

在发送表情信息的时候，我们可能会用到将表情加载到文本框

```
/**
 * 将表情资源添加到文本编辑框中
 * @param mContentEdit 文本编辑框
 * @param emotions 表情资源对象
 */
UIUtils. addEmotions(EditText mContentEdit,Resource emotions)
以GridView展示表情资源为例：
```

```
expressionGriView = (GridView) this
    .findViewById(R.id.expressionGridView);
emotionsImageAdapter = new EmotionsImageAdapter(this);
expressionGriView.setAdapter(emotionsImageAdapter);
expressionGriView.setOnItemClickListener(new OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view,
        int position, long id) {
        final Resource emotions = (Resource) emotionsImageAdapter
            .getItem(position);
        UIUtils.addEmotions(mContentEdit, emotions);
    }
});
```

发表情和文本是相同的，都是获取到文本框中的内容后发送

```
EntboostCM. sendText (sTo, text);
EntboostCM. sendText (account, text);
```

3.10.4.发送图片

首先获取本地图片的地址，或者先拍照再获取图片地址也可以。

```
/**
```



```
* 给个人发送图片消息
*
* @param sTo
*         发送对方的ID
* @param pic
*         图片存放地址
*/
EntboostCM. sendPic (sTo, pic);
/**
* 给个人发送图片消息
*
* @param account
*         发送对方的账号
* @param pic
*         图片存放地址
*/
EntboostCM. sendPic (account, pic);
```

3.10.5.发送语音

发送语音的时候需要借助ExtAudioRecorder语音工具类

1、首先打开录音

```
ExtAudioRecorder recorder = EntboostCM.startRecording();
```

2、录音结束后，需要关闭录音

```
recorder.stopRecord();
```

3、在录音结束后，可以获取到录音文件地址

```
recorder.getFilePath()
```

4、发送录音

```
/**
* 给个人发送语音信息
*
* @param sTo
*         发送对方的ID
* @param voice
*         语音文件存放地址
*/
EntboostCM. sendVoice(sTo, voice)
/**
* 给个人发送语音信息
*
* @param account
*         发送对方的账号
* @param voice
*         语音文件存放地址
*/
```

```
EntboostCM. sendVoice(account, voice)
```

接收的时候获取到 ChatRoomRichMsg 对象

5、先判断是否为语音信息

```
mChatMsg.getType()==ChatRoomRichMsg.CHATROOMRICHMSG_TYPE_VOICE
```

7、获取语音文件地址 mChatMsg.getVoicePath();

8、播放语音文件

```
ExtAudioRecorder.play(mChatMsg.getVoicePath());
```

3.10.6.收发文件

取消文件的发送（可以在未发送和发送中执行），首先需要获取到消息的对象 ChatRoomRichMsg

```
EntboostCM. cancelSendFile(ChatRoomRichMsg msg)
```

发送文件（sTo 发送对方的用户编号，file 发送的文件地址）

```
EntboostCM.sendFile(Long sTo, String file)
```

接收文件（对方发送文件过来）

```
EntboostCM. acceptFile (ChatRoomRichMsg msg)
```

拒绝接收文件（对方发送文件过来）

```
EntboostCM. rejectFile (ChatRoomRichMsg msg)
```

收到文件时的监听(用来更新接收文件的进度)

```
onReceiveFileIncoming(FileCache fileCache)
```

发送文件中的监听(用来更新发送文件的进度)

```
onSendFileIng(ChatRoomRichMsg msg)
```

3.10.7.重发消息

消息未发送成功，可以重发消息，首先需要获取到消息的对象 ChatRoomRichMsg

```
EntboostCM. reSendMsg(pRichMsg)
```

重发后，该消息对象的状态将会发生变化，并自动出发 onSendStatusChanged 事件，

3.10.8.接收消息

接收消息需要注册系统监听，请参考后面关于接收监听的章节，这里只介绍接收消息的监听方法

```
/**
 * 接收到用户发送的消息
 *
 * @param msg
 */
public abstract void onReceiveUserMessage(ChatRoomRichMsg msg);
```

用户实现这个接口，就可以完成接收到消息后的操作，方法中的参数 ChatRoomRichMsg 为传入的消息对象，我们可以从这个对象中获取想要的信息。

1、获取会话的类型

```
msg.getType()
```

```
ChatRoomRichMsg.CHATTYPE_PERSON = 0; //个人会话
```

```
ChatRoomRichMsg.CHATTYPE_GROUP = 1; //群组会话
```

2、获取消息的类型

```
msg.getChatType()
```

```
ChatRoomRichMsg.CHATROOMRICHMSG_TYPE_RICH = 12; // 文本、表情
ChatRoomRichMsg.CHATROOMRICHMSG_TYPE_PIC = 0;
ChatRoomRichMsg.CHATROOMRICHMSG_TYPE_VOICE = 11;
```

3、获取富文本信息

```
/**
 * 获取动态显示的提示信息（包括图片），如果是文字则直接显示文字
 * ChatRoomRichMsg:封装的会话对象，接收会话的时候会获取到
 * @return
 */
msg.getTipHtml()
```

为了可以在界面中显示表情和文本信息，需要用到UIUtils的方法

```
TextView.setText(UIUtils.getTipCharSequence(msg.getTipHtml()));
```

4、获取图片信息的图片文件地址

```
msg.getPicPath()
```

5、获取语音信息的语音文件地址

```
msg.getVoicePath()
```

6、获取消息发送者编号

```
msg.getSender()
```

7、获取消息接收者编号

```
msg.getToUid()
```

8、获取消息接收的时间

```
msg.getMsgTime()
```

```
/**
 * 发送的消息状态改变：0发送中，1发送成功，2发送失败
 *
 * @param msg
 */
public abstract void onSendStatusChanged(ChatRoomRichMsg msg);
```

接收到消息发送的状态变化，可以用来刷新界面以显示这个状态。

3.10.9.聊天记录

1、获取缓存中的聊天记录，uid表示聊天对方的编号，个人会话就是用户编号，群组会话是群组编号

```
EntboostCache.getChatMsgs(uid)
```

2、保存聊天记录到本地数据库

```
EntboostCache.saveChatCache();
```

3、加载本地聊天记录信息

个人会话和群组会话的加载方法不同，目前加载聊天记录还不支持分页加载。

```
EntboostCM.LoadPersonChatDiscCache(uid);
```

```
EntboostCM.LoadGroupChatDiscCache(did);
```

3.11. 群组会话操作

群组操作大部分和个人会话类似，请参考个人会话操作

3.11.1.发送文本消息

```
/**
 * 给群组发送文本信息
 *
 * @param sTo
 *         发送群组的ID
 * @param text
 *         文本信息
 */
EntboostCM. sendGroupText (sTo, text);

/**
 * 给群组发送文本信息
 *
 * @param account
 *         发送群组的账号
 * @param text
 *         文本信息
 */
EntboostCM. sendGroupText (account, text);
```

3.11.2.发送表情

发送表情的方法与发送文本完全相同，其它需要注意的 API 与个人会话相同。

```
EntboostCM. sendGroupText (sTo, text);
EntboostCM. sendGroupText (account, text);
```

3.11.3. 发送图片

```
/**
 * 给群组发送图片消息
 *
 * @param sTo
 *         发送对方群组的ID
 * @param pic
 *         图片存放地址
 */
EntboostCM. sendGroupPic (sTo, pic);

/**
 * 给群组发送图片消息
 *
 * @param account
 *         发送对方群组的账号
 * @param pic
 *         图片存放地址
 */
```

```
EntboostCM. sendGroupPic (account, pic);
```

3.11.4.发送语音

```
/**
 * 给群组发送语音信息
 *
 * @param sTo
 *      发送对方群组的ID
 * @param voice
 *      语音文件存放地址
 */
EntboostCM. sendGroupVoice (sTo, voice);

/**
 * 给群组发送语音信息
 *
 * @param account
 *      发送对方群组的账号
 * @param voice
 *      语音文件存放地址
 */
EntboostCM. sendGroupVoice (account, voice);
```

3.12. 动态信息

动态信息主要用于动态信息列表的展示,显示用户接受的最新消息列表、应用信息列表等等。所有的动态历史信息,都系统在运行过程中放入缓存的,例如聊天消息信息,系统通知信息等。

3.12.1.监听动态消息

添加删除监听动态消息请参考系统监听部分。

```
/**
 * 接收到动态消息
 * @param news
 */
public abstract void onReceiveDynamicNews(DynamicNews news);
```

3.12.2.获取动态历史信息

EntboostCache.getHistoryMsgList()

返回DynamicNews对象,我们可以从这个对象获取动态信息的标题、内容、未读消息数等等

EntboostCache. getUnreadNumDynamicNews()

获取所有动态历史信息的未读数量。

dynamicNews.getNoReadNum() //未读数量

```
dynamicNews.getTitle() //标题
```

```
UIUtils.getTipCharSequence(dynamicNews.getContent()) //内容
```

```
dynamicNews.getTime() //时间
```

3.12.3. 读取动态信息

```
dynamicNews.readAll();//表示动态信息的未读数量将置为 0
```

```
EntboostCache.readMsg(Long sender)//表示指定的动态信息未读数量将置为 0
```

3.12.4. 清空所有动态信息

```
EntboostCache.clearAllMsgHistory();
```

3.13. 联系人信息

3.13.1. 添加联系人

```
/**
 * 添加联系人
 *
 * @param contact 联系人帐号（邮箱地址、电话等）
 * @param group 分组名称
 * @param listener 编辑监听
 */
EntboostUM.addContact(contact, group, new EditContactListener() {

    @Override
    public void onFailure(String errMsg) {
        //TODO 添加失败
    }

    @Override
    public void onEditContactSuccess() {
        //TODO 添加成功
    }
});

/**
 *
 * @param contact
 *         联系人帐号（邮箱地址、电话等）
 * @param name
 *         联系人名称
 * @param desc
 *         备注
 * @param group
 *         分组名称
 * @param listener
```

```

        *          添加监听
    */

EntboostUM.addContact(contact,name,desc,group, new EditContactListener() {

    @Override
    public void onFailure(String errMsg) {
        //TODO 添加失败
    }

    @Override
    public void onEditContactSuccess() {
        //TODO 添加成功
    }

});

/**
 * 添加联系人
 *
 * @param card
 *          呼叫是收到的名片信息
 * @param listener
 *          添加监听
 */
EntboostUM.addContact (card, new EditContactListener() {

    @Override
    public void onFailure(String errMsg) {
        //TODO 添加失败
    }

    @Override
    public void onEditContactSuccess() {
        //TODO 添加成功
    }

});

```

3.13.2.编辑联系人

```

/**
 * 编辑联系人的备注名称和描述
 *
 * @param contact 联系人帐号（邮箱地址、电话等）
 * @param name 联系人名称
 * @param description 联系人描述
 * @param listener 编辑监听
 */

```

```
EntboostUM.editContact(contact , name, description, new EditContactListener() {
```

```
    @Override
    public void onFailure(String errMsg) {
        //TODO 修改失败
    }

    @Override
    public void onEditContactSuccess() {
        //TODO 修改成功
    }
});
```

3.13.3.编辑联系人 (详细)

```
/**
 * 编辑联系人详细信息
 *
 * @param contact
 *         联系人帐号
 * @param name
 *         名称
 * @param phone
 *         电话
 * @param email
 *         邮箱
 * @param address
 *         地址
 * @param description
 *         描述
 * @param company
 *         公司
 * @param job_title
 *         职位
 * @param fax
 *         传真
 * @param tel
 *         电话
 * @param url
 *         网址
 * @param listener
 */
EntboostUM. editContact (contact, name, phone,email, address, description,
company,job_title, fax, tel, url ,new EditContactListener (){
    @Override
    public void onFailure(String errMsg) {
```



```
        //TODO 修改失败
    }
    @Override
    public void onEditContactSuccess() {
        //TODO 修改成功
    }
});
```

3.13.4.删除联系人

```
/**
 * 删除联系人
 *
 * @param contact 联系人帐号（邮箱地址、电话等）
 * @param listener 编辑监听
 */
EntboostUM.delContact(contact, new DelContactListener() {

    @Override
    public void onFailure(String errMsg) {
        //TODO 删除失败
    }

    @Override
    public void onDelContactSuccess() {
        //TODO 删除成功
    }

});
```

3.13.5.移动联系人到其它分组

```
/**
 * 移动联系人到新的分组
 *
 * @param contact 联系人帐号（邮箱地址、电话等）
 * @param newGroup 分组名称
 * @param listener 编辑监听
 */
EntboostUM.moveContact(contact, newGroup, new EditContactListener() {

    @Override
    public void onFailure(String errMsg) {
        //TODO 修改失败
    }

    @Override
```

```
        public void onEditContactSuccess() {  
            //TODO 修改成功  
        }  
    });
```

3.14. 扩展应用

扩展应用配置需要到服务器后台配置，开放 IM 平台会默认加载一些扩展应用，如果是私有服务器版本，请参考服务后台管理的功能说明。

3.14.1. 应用列表的加载

用户登录后，会自动加载应用列表。

3.14.2. 获取应用列表

EntboostCache.getFuncInfos()

返回 FuncInfo 对象，可以获取应用的图标、链接、应用名称等信息。

3.14.3. 加载应用功能列表

EntboostUM.loadFnav(FuncInfoId, LoadFnavListener);

根据应用编号加载应用的具体功能列表，返回列表注意需要区分级别，目前暂时只提供两个级别的功能菜单。

3.14.4. 监听扩展应用功能界面的链接

```
EntboostUM.fnavArgs(url, new FnavArgsListener() {  
    //关闭浏览器窗口  
    public void closeWindow() {  
    }  
  
    //开始执行链接操作  
    public void onStart() {  
    }  
  
    //发送群组会话成功  
    public void send_group_success(Long depCode, GroupInfo group) {  
    }  
  
    //发送个人会话成功  
    public void send_account_success(Long uid, CardInfo cardInfo) {  
    }  
  
    //其它不需要系统监听的链接  
    public void defaultLink() {  
    }  
  
    //功能操作失败、超时等
```

```
        public void onFailure(String errMsg) {  
        }
```

```
    });
```

在使用 web 浏览器的时候加载拦截链接的监听：

```
mWebView.setWebViewClient(new WebViewClient() {  
    public boolean shouldOverrideUrlLoading(final WebView view,  
        final String url) {  
        //执行加载监听的方法  
    }  
});
```

3.15. 系统广播消息

3.15.1. 增加系统监听接口

```
/**  
 * 接收到系统广播消息  
 *  
 * @param depName  
 */  
public abstract void onReceiveBroadcastMessage(BroadcastMessage message);
```

使用系统监听接口的方法请参考系统监听部分。

3.15.2. 获取系统广播消息列表

EntboostCache.getBroadcastMessages()

返回 BroadcastMessage 对象列表

3.16. 缓存操作

```
/**  
 * 获取app信息  
 * @return  
 */  
EntboostCache.getAppInfo()  
/**  
 * 获取我的消息的系统扩展应用对象  
 *  
 * @return  
 */  
EntboostCache.getMyMessageFuncInfo()  
  
/**  
 * 根据用户编号来获取名片信息  
 *  
 * @param uid  
 * @return
```

```
*/  
EntboostCache.getCardInfo(Long uid)  
  
/**  
 * 获取所有需要处理的呼叫信息列表  
 *  
 * @return  
 */  
EntboostCache.getAllDealCallInfos()  
  
/**  
 * 获取最后一次登录的帐号  
 *  
 * @return  
 */  
EntboostCache.getLastLoginAccount()  
  
/**  
 * 获取所有登录过的帐号数组  
 *  
 * @return  
 */  
EntboostCache.getAccountHistorys()  
  
/**  
 * 获取扩展应用列表  
 *  
 * @return  
 */  
EntboostCache.getFuncInfos()  
  
/**  
 * 获取登录服务器地址  
 *  
 * @return  
 */  
EntboostCache.getSharedLogonCenterAddr()  
  
/**  
 * 保存登录服务器地址  
 *  
 * @param addr  
 */  
EntboostCache.setLogonCenterAddr(String addr)
```

```
/**
 * 是否默认登录
 *
 * @return
 */
EntboostCache.getDefaultLogon()

/**
 * 根据成员编号获取成员信息对象
 *
 * @param emp_code
 * @return
 */
EntboostCache.getMemberByCode(Long emp_code)

/**
 * 根据用户编号和群组编号获取群组成员对象
 * @param uid
 * @param groupid
 * @return
 */
EntboostCache.getMember(Long uid, Long groupid)

/**
 * 获取登录用户的默认群组成员对象
 *
 * @return
 */
EntboostCache.getMyDefaultMember()

/**
 * 通过编号获取名称
 *
 * @param uid
 * @return
 */
EntboostCache.getName(Long uid)

/**
 * 查询联系人
 *
 * @param contacts
 * @param s
 */
EntboostCache.searchContact(List<SearchResultInfo> result, String s)
```

```
/**
 * 查询群组
 *
 * @param result
 * @param s
 */
EntboostCache.searchGroup(List<SearchResultInfo> result, String s)
```

```
/**
 * 获取未读信息的总数量
 *
 * @return
 */
EntboostCache.getUnreadNumDynamicNews()
```

```
/**
 * 读取信息
 *
 * @param sender
 */
EntboostCache.readMsg(Long sender)
```

```
/**
 * 清空所有的动态信息
 */
EntboostCache.clearAllMsgHistory()
```

```
/**
 * 清除一条动态信息
 * @param news
 */
EntboostCache.clearMsgHistory(DynamicNews news)
```

```
/**
 * 获取当前登录用户的UID
 *
 * @return
 */
EntboostCache.getUid()
```

```
/**
 * 获取当前登录用户对象
 *
 * @return
```

```
*/  
EntboostCache.getUser()  
  
/**  
 * 获取所有联系人信息  
 *  
 * @return  
 */  
EntboostCache.getContactInfos()  
  
/**  
 * 根据用户编号获取联系人对象  
 * @param uid  
 * @return  
 */  
EntboostCache.getContactInfo(Long uid)  
  
/**  
 * 根据联系人名称获取联系人对象  
 * @param contact  
 * @return  
 */  
EntboostCache.getContactInfo(String contact)  
  
/**  
 * 获取所有个人群组信息  
 *  
 * @return  
 */  
EntboostCache.getPersonGroups()  
  
/**  
 * 获取所有群组成员信息  
 *  
 * @return  
 */  
EntboostCache.getGroupMemberInfos()  
  
/**  
 * 获取所有的默认表情资源  
 *  
 * @return  
 */  
EntboostCache.getEmotionslist()
```

```
/**
 * 获取所有的默认头像资源
 *
 * @return
 */
EntboostCache.getHeadlist()

/**
 * 获取所有历史信息列表
 *
 * @return
 */
EntboostCache.getHistoryMsgList()

/**
 * 获取历史信息
 *
 * @return
 */
EntboostCache.getHistoryMsg(Long sender)

/**
 * 获取会话信息
 *
 * @param to
 * @return
 */
EntboostCache.getChatMsgs(Long to)

/**
 * 加载本地群组会话信息
 *
 * @param groupid
 */
EntboostCache.loadGroupChatMsg(Long groupid)

/**
 * 加载本地个人会话信息
 *
 * @param uid
 */
EntboostCache.loadPersonChatMsg(Long uid)

/**
 * 根据群组编号获取群组信息
```



```
* @param depid
* @return
*/
EntboostCache.getDepartment(Long depid)

/**
 * 获取所有我的部门，包括项目组
 *
 * @return
 */
EntboostCache.getMyDepartments()

/**
 * 获取企业信息
 *
 * @return
 */
EntboostCache.getEnterpriseInfo()

/**
 * 获取所有企业部门信息
 *
 * @return
 */
EntboostCache.getDepartments()

/**
 * 获取所有群组成员信息
 *
 * @return
 */
EntboostCache.getDepartmentMemberInfos()
```

3.17. 接收监听

目前系统只有一个监听对象，注册后就可以接收所有系统下发的各种消息，例如服务器网络状况、即时消息的内容、各种系统通知等。

```
//系统监听的接口
public interface EntboostIMListenerInterface
//系统监听的默认实现
public class EntboostIMListener implements EntboostIMListenerInterface
```

3.17.1.注册与取消监听

注册有两种方式，一种是实现接口，一种是继承默认实现。下面介绍两种最常用的实现

方式:

1、主界面注册监听接口

//一般在主界面中实现系统监听接口,或者为所有界面提供一个基类,在基类中实现接口,不同界面如果具体监听实现不同,可以覆盖接口,编辑特定界面的自定义内容

```
public class MainActivity extends Activity implements EntboostIMListenerInterface {...}
//注册监听
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Entboost.addListener(this);
}
//取消监听
@Override
protected void onDestroy() {
    super.onDestroy();
    Entboost.removeListener(this);
}
```

2、实例化监听实现的对象

//实例化监听对象的好处在于不用实现所有的接口,可以根据需要覆盖特定的方法,也可以根据需要 anywhere 注册和取消监听

```
Entboost.addListener(this.getClass().getName(),
    new EntboostIMListener() {...});
//取消监听
Entboost.removeListener(this.getClass().getName());
```

3.17.2.监听详细说明

```
public interface EntboostIMListenerInterface {

    /**
     * 服务器连接不上
     */
    public abstract void disconnect(String err);

    /**
     * 服务器连接恢复正常
     */
    public abstract void network();

    /**
     * 修改在线状态
     */
    public abstract void onUserStateChange(Long uid);

    /**
     * 接收到用户发送的消息
     */
}
```

```
*
* @param msg
*/
public abstract void onReceiveUserMessage(ChatRoomRichMsg msg);

/**
 * 发送的消息状态改变: 0发送中, 1发送成功, 2发送失败
 *
 * @param msg
 */
public abstract void onSendStatusChanged(ChatRoomRichMsg msg);

/**
 * 有呼叫呼入事件处理
 */
public abstract void onCallIncoming(CardInfo card);

/**
 * 收到用户拒绝会话处理
 *
 * @param callInfo
 */
public abstract void onCallReject(CardInfo card);

/**
 * 收到用户挂断会话处理
 */
public abstract void onCallHangup(Long to);

/**
 * 收到会话超时处理
 *
 * @param callInfo
 */
public abstract void onCallTimeout(CardInfo card);

/**
 * 当用户加入会话时处理
 *
 * @param connectInfo
 */
public abstract void onCallConnected(Long to);

/**
 * 接收到我的系统消息
```

```

    *
    * @param depName
    */
    public abstract void onReceiveMyMessage(SysMessage message);
    /**
    * 接收到系统广播消息
    *
    */

    */
    public abstract void onReceiveBroadcastMessage(BroadcastMessage message);
    /**
    * 接收到动态消息
    *
    */
    public abstract void onReceiveDynamicNews(DynamicNews news);
    /**
    * 接收文件中
    *
    */
    public abstract void onReceiveFileIncoming(FileCache fileCache);
    /**
    * 发送文件中
    *
    */
    public abstract void onSendFileIng(ChatRoomRichMsg msg);
    /**
    * 通知自己，已经在其他地方登录，退出前一个连接
    */
    public abstract void online_another();
}

```