# Android 插件化开发指南

**（IM插件）**

[Android 插件化开发指南 1](#_Toc26239)

[一、 Android版本IM插件的介绍及导入 7](#_Toc19961)

[1.1 IM插件的介绍及结构 7](#_Toc12156)

[1.2 引用插件 8](#_Toc12877)

[1.3 配置工程 10](#_Toc9788)

[二、IM插件初始化和配置 17](#_Toc15863)

[2.1 在onCreate()中初始化 17](#_Toc8734)

[2.2 IM功能配置 17](#_Toc8664)

[三、登录和登出 30](#_Toc21388)

[3.1 登录 30](#_Toc19316)

[3.2 登出 34](#_Toc7587)

[四、可接入界面 34](#_Toc5779)

[4.1 会话列表（ConversationListFragment） 34](#_Toc17845)

[4.2群组列表（GroupListFragment） 35](#_Toc28970)

[其效果图如下图： 36](#_Toc28715)

[五、主动调用的方法 36](#_Toc11061)

[5.1 获取未读消息 36](#_Toc23909)

[5.2 发起单聊 37](#_Toc15009)

[5.3 发起群聊 37](#_Toc11893)

[5.4 根据groupID获取群成员 38](#_Toc15246)

[5.5 IM全局设置 38](#_Toc30965)

[5.6 IM全局设置状态的获取 39](#_Toc30555)

## Android版本IM插件的介绍及导入

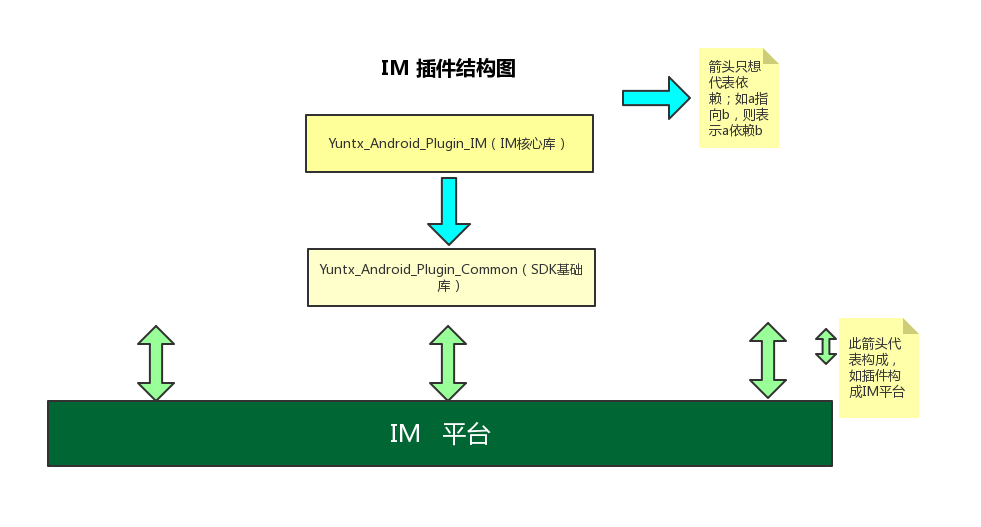
温馨提示：有[下划线](#_Android版本IM插件的介绍及导入)的文字为超链接，超链接跳转请同时按ctrl和鼠标左键

### IM插件的介绍及结构

#### 1.1.1 插件介绍

IM插件为开发者开发 IM 相关的应用提供的一套完善的开发框架，将IM基础能力和界面集成到aar文件中，方便开发者快速高效的集成IM通讯能力，开发者可以基于我们提供的IM插件实现自己的应用，也可以基于Demo开发。

#### 1.1.2 插件结构



#### 1.1.3 IM插件主要对外暴露类

1、WrapManger: 提供插件完整能力 有登录/登出方法/可接入页面的；

2、IMPluginManager:里面主要有获取未读消息、发起群聊/单聊、IM全局的设置、获取IM全局设置状态的方法；

3、IMPanel:这个类为聊天页面底部的扩展基类，可以设置扩展按钮名字、icon及点击事件。

1. ChattingImpl：这个类作为IM插件核心功能事件回调实现类，重写可在此类中实现跳转到自定义页面，比如选择联系人页面等。

## 二、工程配置

### 2.1 权限配置

<uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE" />

<uses-permission android:name="android.permission.SYSTEM\_ALERT\_WINDOW" />

// 以上为v5.1.9r 版本增加

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

<!-- 允许程序访问有关GSM网络信息 -->

<uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE" />

<!-- 允许程序访问Wi-Fi网络状态信息 -->

<uses-permission android:name="android.permission.ACCESS\_WIFI\_STATE" />

<uses-permission android:name="android.permission.GET\_TASKS" />

<!-- 允许访问设备媒体信息 -->

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

<uses-permission android:name="android.permission.RECORD\_AUDIO" />

<uses-permission android:name="android.permission.CALL\_PHONE" />

<uses-permission android:name="android.permission.WAKE\_LOCK" />

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

<uses-permission android:name="android.permission.READ\_PHONE\_STATE"/>

<!--&lt;!&ndash; 允许程序修改全局音频设置 &ndash;&gt;-->

<uses-permission android:name="android.permission.MODIFY\_AUDIO\_SETTINGS" />

<uses-permission android:name="android.permission.SYSTEM\_ALERT\_WINDOW" />

### 2.2 依赖库

//需要依赖的第三方版本库

glide\_transfor : "jp.wasabeef:glide-transformations:4.1.0",

rxbinding : "com.jakewharton.rxbinding2:rxbinding:2.0.0",

subsamplingImage : "com.davemorrissey.labs:subsampling-scale-image-view:3.6.0",

Luban : 'top.zibin:Luban:1.1.8',

rxlifecycle : "com.trello.rxlifecycle3:rxlifecycle:3.0.0",

rxlifecycle\_components : "com.trello.rxlifecycle3:rxlifecycle-components:3.0.0",

//阿里路由

arouter\_api : 'com.alibaba:arouter-api:1.5.2',

arouter\_compiler : 'com.alibaba:arouter-compiler:1.5.2',

eventbus\_processor : "org.greenrobot:eventbus-annotation-processor:3.2.0"

kotlin\_stdlib\_jdk8 : "org.jetbrains.kotlin:kotlin-stdlib-jdk8:1.5.21"

wcdb : "com.tencent.wcdb:wcdb-android:1.0.6.2"

TencentLocationSdk : "com.tencent.map.geolocation:TencentLocationSdk-openplatform:7.2.6"

tablayout : "io.github.h07000223:flycoTabLayout:3.0.0"

// 容联IM能力依赖

api 'com.github.JiangJun919690767.rongxin\_ability\_sdk:im\_wrap:v0.0.19'

### 2.3 gradle配置

需要在引用的module的build.gradle路由配置

android **{**

defaultConfig **{**

javaCompileOptions **{**

annotationProcessorOptions **{**

arguments = [AROUTER\_MODULE\_NAME: project.getName()]

**}**

**}**

**}**

**}**

### 2.4 sdk日志开关配置 开发需要时进行打开日志开关

//java层次日志

<meta-data

android:name="DEBUG\_LOG"

android:value="true" />

//信令库日志

<meta-data

android:name="TRACE\_LOG"

android:value="true" />

///媒体库日志

<meta-data

android:name="LOG\_LEVEL"

android:value="15" />

//白板日志

<meta-data

android:name="CORE\_LOG"

android:value="false" />

LOG\_LEVEL\_WARNING=11,

LOG\_LEVEL\_INFO=12,

LOG\_LEVEL\_DEBUG=13,

LOG\_LEVEL\_DEBUG\_WBSS=14,//白板日志开启级别

LOG\_LEVEL\_MEDIA\_ERR=20,//媒体库日志,打印ECmedia调用日志和媒体库Error日志；

LOG\_LEVEL\_MEDIA\_WARNING=21,//添加严重错误的日志；

LOG\_LEVEL\_MEDIA\_INFO=22,//添加警告日志；

LOG\_LEVEL\_MEDIA\_DEFAULT=23,//媒体库默认日志，一般打开这个；

LOG\_LEVEL\_MEDIA\_DEBUG=24,//添加调试信息；

LOG\_LEVEL\_MEDIA\_ALL=25,//大于24： 所有日志都打开，级别最好，把媒体流的信息都打印出来

LOG\_LEVEL\_END=99

## 三插件初始化和使用

**注意事项**：具体API可以查看DOC

### 3.1 初始化

**注意事项：**请开发者拷贝下面代码到自己的application的onCreate()方法中，提供重载方法

开发者按照需要选这一种方法进行配置就可以

WrapManager.*getInstance*().app\_init(this);

*/\*\**

*\* 初始化 放在Application中*

*\** ***@param*** *application*

*\** ***@param*** *isDebug true 打开aar层的日志，false关闭*

*\** ***@param*** *impl 会议一直配置可以通过重写ConfImpl，进行注入*

*\*/*

WrapManager.*getInstance*().app\_init(this,false, ConfImpl.*getInstance*());

### 3.2 登录，登出，重连

*/\*\**

*\* 插件登录接口 初始化用户信息*

*\*登录前需要调用initPermission初始化权限，流程可参考 LoginActivity*

*\** ***@param*** *privateAddress 服务器IP地址*

*\** ***@param*** *user 用户基本信息*

*\** ***@param*** *listener 登陆成功的响应回调*

*\*/*

WrapManager.*getInstance*().app\_LoginIM(account, psw, new SDKCoreHelper.OnConnectStateListener());

*/\*\**

*\* 登出*

*\** ***@param*** *mExitType WrapManager.TYPE\_LOGOUT 退出账号*

*\* WrapManager.TYPE\_EXIT 退出应用，只断开SDK连接*

*\** ***@param*** *listener 登出成功响应的回调*

*\*/*

WrapManager.*getInstance*().app\_Logout(type, (i, s) -> {

});

*/\*\**

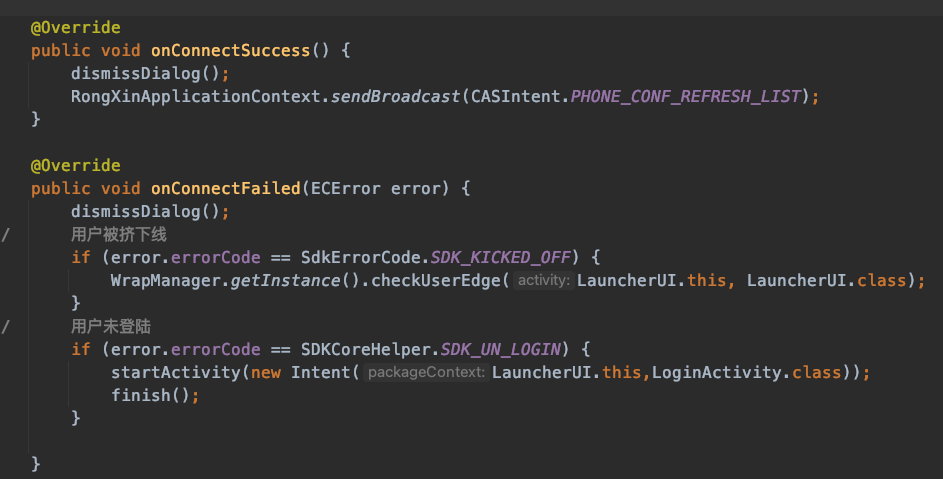
*\* app已经登陆过，杀进程在进来 主页面调用sdk重新连接*

*\*/*

WrapManager.*getInstance*().app\_AutoConnect(SDKCoreHelper.OnConnectStateListener listener);

OnConnectStateListener 回调处理 可参考demo逻辑

### 3.3 获取异地登录状态



3.4 注册事件

在主页activity中onCreate 中调用registerPluginReceiver

WrapManager.*getInstance*().registerPluginReceiver(LauncherUI.this);

在主页activityonDestroy中调用unregisterPluginReceiver

WrapManager.*getInstance*().unregisterPluginReceiver()

-------完成以上步骤后，即开始调用能力了

## 四、可接入界面

### 4.1 会话列表（ConversationListFragment）

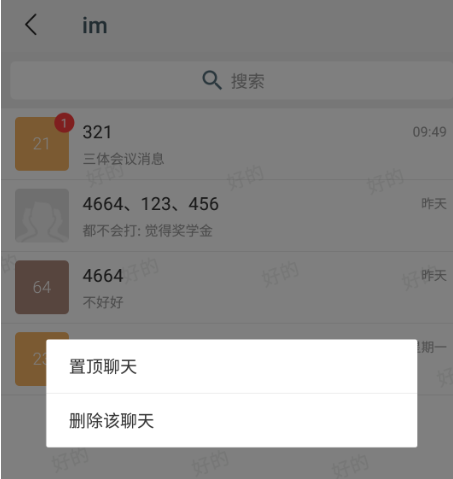
IM插件提供会话列表（ConversationListFragment），开发者可根据需求接入在自己的项目中，使用示例：

/\*\*

\* 在开发者指定的activity中复制下面代码，\*/

WrapManager.getInstance().ConversationListFragment()

效果图如下：



**页面功能：**

1、显示单聊或者群聊消息；

2、单击列表中Item进入对应的单聊或者群聊里面；

3、长按列表中Item，会提示置顶聊天和删除该聊天；

4、自动根据每个会话的未读消息的在列表对应的Item上显示未读数量；

5、其他功能。

4.2通讯录列表（TabContactFragment）

使用示例：

/\*\*

\* 开发者在容器activity中按照一般fragment嵌入得方法使用

\*/

WrapManager.*getInstance*().getTabContactFragment(this)

4.3创建群组（createGroupUI）

使用示例：

/\*\*

\* 开发者在容器activity中进行跳转

\*/

WrapManager.*getInstance*().createGroupUI(getActivity());

## 五、主动调用的方法

### 5.1 获取未读消息

**注意事项：**结合文档[配置未读消息数量改变的监听](#_4)_配置未读消息数量改变的监听)使用

**使用场景：1）**在开发者完成文档[配置未读消息数量改变的监听](#_4)_配置未读消息数量改变的监听)后，在[OnUpdateMsgUnreadCounts()](#_4)_配置未读消息数量改变的监听)中调用改方法；**2）** 需要获取未读消息的总数量时调用

/\*\*

\* 此方法可主动调用，用于获取全部未读消息的数量

\* **@return** 类型：int 作用：可提示的全部未读信息的数量

\*/

int unReadMsgCount = IMPluginManager.getManager().getUnReadMsgCount();

### 5.2 发起单聊

**使用场景：**开发者若需要和某人发起聊天，可主动调用该方法，需传入聊天对象的userId

/\*\*

\* 发起单聊

\* **@param** context 类型：上下文

\* **@param** userId 类型：String 传入：指定聊天对象的userID

\*/

IMPluginManager.getManager().startSingChat(context,"与之聊天的userId");

### 5.3 发起群聊

**使用场景：**开发者若需要创建群组，可主动调用该方法，传入需要添加进群的userId的字符串集合或者数组，注意：无需传入创建者（自己）的userId

/\*\*

\* 此方法为发起群聊

\* **@params** Context 上下文

\* **@params** 第二个参数 可以为List<String> 也可以为 String[] 两者均盛装需添加进群的userId

\* **@params** OnCreateGroupStateListener 创建群组的状态监听器

\*/

IMPluginManager.getManager().startGroupChat(context, list, new OnCreateGroupStateListener() {

// 创建开始

@Override

public void onStart() {

// 开发者根据实际情况处理

}

// 创建成功

@Override

public void onSuccess(Context context, String groupId) {

// 开发者根据实际情况处理

}

// 创建失败

@Override

public void onFailed(int errorCode) {

// 开发者根据实际情况处理

}

});

### 5.4 根据groupID获取群成员

**使用场景：**譬如开发者在群聊页面底部面板添加发起视频会议按钮，选取发起视频对象前，可能需要知道本群的全部成员时，可调用此方法

/\*\*

\* **@param** groupID 群组的groupID

\*/

IMPluginManager.getManager().getGroupMember(groupID);

### 5.5 IM全局设置

**作用位置：（此图为示例，实际页面应是开发者的设置页面）**



**注意事项：**如果把[1)接收新消息通知](#_1)_接收新消息通知)设置为false，那么[2)声音提示新消息](#_2)_声音提示新消息)和[3)震动提示新消息](#_3)_震动提示新消息)也会强制设置为false

**使用场景：**开发者应把这些设置放置在自己的某个页面内来进行全局设置IM，根据下面提供的方法来进行设置，并根据[5.5](#_5.5_IM全局设置状态的获取)来获取这些设置的状态，更新开发者的这些开关状态。true打开，false关闭，默认是true，

#### 1) 接收新消息通知

/\*\*

\* 关闭／开启 消息通知

\* **@param** boolean 开启true 关闭false

\*/

IMPluginManager.getManager().setReceiveMessagesNotify(true);

#### 2) 声音提示新消息

/\*\*

\* 接收到新消息时使用声音提醒，开启true 关闭false

\*/

IMPluginManager.getManager().useSoundToNotify(true);

#### 3) 震动提示新消息

/\*\*

\* 接收到新消息时使用震动提醒，开启true 关闭false

\*/

IMPluginManager.getManager().useShakeToNotify(true);

#### 4) 听筒/扬声器接听语音

/\*\*

\* 默认使用听筒播放声音，设置false后使用外音

\* **@param** value

\*/

IMPluginManager.getManager().useHandSetToPlayVoice(true);

#### 5) 清除所有聊天记录

/\*\*

\* 清除所有聊天记录

\*/

IMPluginManager.getManager().clearAllChatRecord();

### 5.6 IM全局设置状态的获取

**使用场景：**开发者若需要获取下列设置的状态来配置自己的设置页面，则可以调用对应的方法对自己设置页面的控件更新

#### 1) 获取接收新消息设置的状态

/\*\*

\* 获取 接收新消息设置 的状态 ture为打开，默认为true

\*/

boolean receiveMessagesNotifyState = IMPluginManager.getManager().getReceiveMessagesNotifyState();

#### 2) 获取声音提示设置的状态

/\*\*

\* 获取 声音提示新消息设置 的状态 ture为打开，默认为true

\*/

boolean soundNotifyState = IMPluginManager.getManager().getSoundNotifyState();

#### 3) 获取震动提示设置的状态

/\*\*

\* 获取 震动提示新消息设置 的状态 true为打开，默认为true

\*/

boolean shakeNotifyState = IMPluginManager.getManager().getShakeNotifyState();

#### 获取听筒接听语音设置的状态及个人信息配置

/\*\*

\* 获取 听筒接收语音设置 的状态 true为打开，默认为true

\*/

boolean handSetSate = IMPluginManager.getManager().getHandSetSate();

/\*\*

\*此方法可以根据conversation(会话页面)列表中传过来的userId

\* 搜索外部数据 然后将这些已搜索到的外部数据源设置进会话列表中

\* 此方法的入参userid如果是非本机登录账号 就是sessionID

\* 如果是本机账号 userid 就是登录ID

\* 例如登录IM插件 输入账号90 那么本机的userID等于90

\* @param userId

\* @return

\*/

@Override

public PerosnInfos queryPeronInfoByUserId(String userId) {

}

## 六 IM功能配置

#### 6.1 基本配置

重写ChattingImpl：这个类作为IM插件核心功能事件回调实现类，可在此类中实现跳转到自定义页面，比如选择联系人页面等如下

##### 1)配置IM通知消息点击事件

**作用位置：**



**注意事项：**点击事件中，开发者只需要配置好intent。指定跳转的页面及传递的数据,开发者不需要调用startActivity()

*/\*\**

*\* 消息通知*

*\**

*\** ***@param*** *ctx 上下文*

*\** ***@param*** *fromUserName 消息发送着*

*\** ***@param*** *sessionId 消息会话id*

*\** ***@param*** *isShowNotifyInfo 通知消息是否显示详情*

*\*/*

public static void notify(final Context ctx, String fromUserName, final String sessionId, boolean isShowNotifyInfo) {

boolean sound = *isPlaySound*();

boolean shake = *isShake*();

final int defaults;

if ((sound && shake)) {

defaults = Notification.*DEFAULT\_ALL*;

shake = false;

} else if ((sound)) {

defaults = Notification.*DEFAULT\_SOUND*;

shake = false;

} else if (shake) {

defaults = Notification.*DEFAULT\_VIBRATE*;

shake = true;

} else {

defaults = Notification.*DEFAULT\_LIGHTS*;

shake = true;

}

final String ticker;

final String title;

final String text;

final boolean onlyVibrate = shake;

final RXConversation conversation = DBRXConversationTools.*getInstance*().queryConversionBySessionId(sessionId);

if (conversation == null) return;

if (isShowNotifyInfo) {

ticker = *getTickerText*(ctx, fromUserName, conversation);

title = *getContentTitle*(ctx, fromUserName, sessionId, conversation.getUserData());

text = *getContentText*(ctx, fromUserName, conversation);

} else {

// 清除所有的状态消息，合成一条

*forceCancelNotification*();

ticker = title = RXConfig.*APP\_NAME*;

int unreadSessioncount = DBRXConversationTools.*getInstance*().getUnreadSessionCount();

text = unreadSessioncount + "个联系人发来" + IMPluginManager.*getUnReadMsgCount*() + "条消息";

if (unreadSessioncount <= 0) return;

}

final PendingIntent intent = *getPendingIntent*(ctx, sessionId, fromUserName, conversation.getUserData());

final String url = conversation == null ? "" : IMPluginHelper.*getHeadUrlbyId*(ctx, conversation.getContactId());

Observable.*just*(url).map(new Func1<String, Bitmap>() {

@Override

public Bitmap call(String s) {

/\*String imagePath = FileAccessor.getAvatarPathName() + "/" + DemoUtils.md5(sessionId);

if (BackwardSupportUtil.isPeerChat(sessionId)) {

return getLargeBitmap(ctx, "file://" + imagePath);

}

return getLargeBitmap(ctx, url);\*/

return *getLargeBitmap*(ctx, s, conversation);

}

}).observeOn(Schedulers.*io*())

.subscribeOn(Schedulers.*io*())

.observeOn(AndroidSchedulers.*mainThread*())

.subscribe(new Action1<Bitmap>() {

@Override

public void call(Bitmap bitmap) {

// 桌面图标显示未读消息数

int count = IMPluginManager.*getUnReadMsgCount*();

Notification notification = *build*(ctx, ticker, title, text, intent, defaults, onlyVibrate, bitmap);

notification.flags = (Notification.*FLAG\_AUTO\_CANCEL* | notification.flags);

BusinessNotification.*setAppBadgeCount*(ctx, notification, count);

RXNotificationMgr.*notify*(ctx, sessionId.hashCode(), notification);

}

});

}

*/\*\**

*\* 获取消息状态栏点击事件处理*

*\**

*\** ***@param*** *context 上下文*

*\** ***@param*** *sessionId 会话id*

*\** ***@param*** *username 消息会话昵称*

*\** ***@return*** *PendingIntent*

*\*/*

private static PendingIntent getPendingIntent(Context context, String sessionId, String username, String userData) {

Intent intent = new Intent();

if (RXConfig.*MONITOR\_PLATFORM\_SESSION\_ID*.equals(sessionId)) {

intent.setClassName(context, "com.yuntongxun.plugin.monitor.MonitorMessageListActivity");

} else if (RXConfig.*ATTEN\_DANCE\_SESSION\_ID*.equals(sessionId)) {

DoubleTapFilter.*clear*();

intent.setClassName(context, "com.yuntongxun.rongxin.lite.ui.LauncherUI");

intent.putExtra("spped\_atten\_dance\_notice", true);

intent.addFlags(Intent.*FLAG\_ACTIVITY\_CLEAR\_TOP*);

intent.addFlags(Intent.*FLAG\_ACTIVITY\_SINGLE\_TOP*);

return PendingIntent.*getActivity*(context, 35, intent, PendingIntent.*FLAG\_ONE\_SHOT*);

} else if (sessionId.startsWith(IMChattingHelper.*MINIAPP\_PUSH\_ACCOUNT*)) {// 第三方应用推送消息

intent.setClassName(context, "com.yuntongxun.plugin.workstore.ui.miniapp.MiniAppMsgListActivity");

intent.putExtra("appName", MiniAppHelper.*getInstance*().parseTitle(userData));

intent.putExtra("appaccount", sessionId);

} else {

//默认跳转到聊天界面

intent.setClass(context, ChattingActivity.class);

intent.addFlags(Intent.*FLAG\_ACTIVITY\_CLEAR\_TOP*);

intent.putExtra(ChattingFragment.*RECIPIENTS*, sessionId);

intent.putExtra(ChattingFragment.*CONTACT\_USER*, username);

}

return PendingIntent.*getActivity*(context, 35, intent, PendingIntent.*FLAG\_UPDATE\_CURRENT*);

}

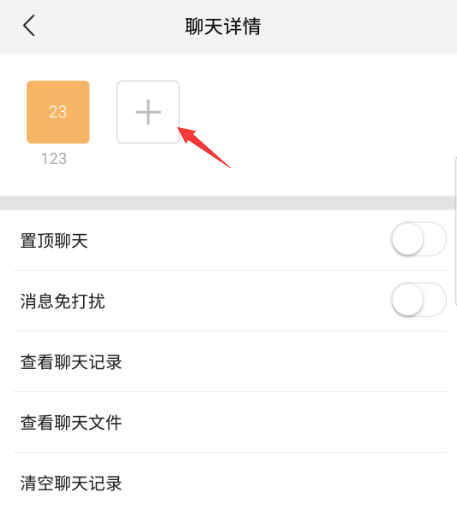
提示：开发者在自己指定的页面可以根据sessionId来再一次使用下面所示方法跳转到聊天页面，如此页面过渡会较为顺滑；也可以不进一步跳转，开发者可根据需求自定义。

##### 2)配置添加成员和转发消息事件

**作用位置（添加成员）：**

**注意事项：**单聊信息里的加号是为创建群组，群组信息里加号是添加群成员





**作用位置（转发消息）：**

###### a 声明回调和设置监听事件（配置页面）

**使用场景：**声明发送文件消息和一些基础控件的回调，然后设置监听事件

IMPluginManager.*getManager*().setHandleSendFileMessageListener(ChattingImpl.*getInstance*());

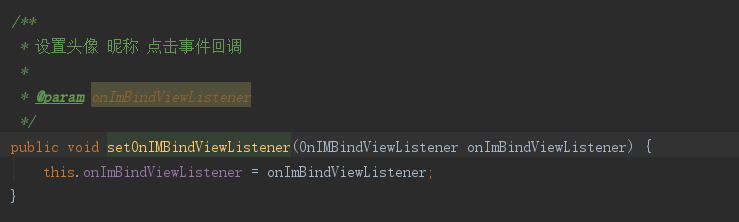
IMPluginManager.*getManager*().setOnIMBindViewListener(ChattingImpl.*getInstance*());

##### 3)配置昵称、头像及点击事件

**作用位置：**



**使用场景：**如果不配置[方法3](#_3)（方法3）配置昵称、头像及点击事件)，那么默认把userId当做昵称，头像使用默认头像，头像默认不能点击



public interface OnIMBindViewListener {

String onBindNickName(Context context, String userId);

String onBindGroupDisplayName(Context context, String groupId, String mRecipients);

boolean isLocalEmployeExit(String userId);

void onBindAvatarView(Context context, ImageView avatar, String userId);

void OnAvatarClickListener(Context context, String id);

String onBindAvatarByUrl(Context context, String userId);

void onStartSystemCall(Context context, String phoneNumber, String username);

*/\*\**

*\* 更新个人信息*

*\** ***@param*** *context*

*\** ***@param*** *userId*

*\*/*

void onUpdateUserInfo(Context context, String userId);

void onUpdateaAcountInfo(Context context, String acount);

}

@Override

public String onBindNickName(Context context, String userId) {

LogUtil.*d*(*TAG*, "onBindNickName " + userId);

if (TextUtils.*isEmpty*(userId)) {

return userId;

}

String userName = RXContactHelper.*getInstance*().getUserName(userId);

LogUtil.*d*(*TAG*, "onBindNickName " + userName + " ,userId " + userId);

if (BackwardSupportUtil.*isNullOrNil*(userName) && RXContactHelper.*getInstance*().queryRXEmployee(userId)) {

if (IMPluginManager.*getManager*().onImBindViewListener != null) {

IMPluginManager.*getManager*().onImBindViewListener.onUpdateaAcountInfo(RongXinApplicationContext.*getContext*(), userId);

}

}

return BackwardSupportUtil.*isNullOrNil*(userName) ? userId : userName;

}

@Override

public String onBindGroupDisplayName(Context context, String groupId, String mRecipients) {

String groupDisplayName = DBRXGroupMemberTools.*getInstance*().queryECGroupMemberDisplayNameByAccount(groupId, mRecipients);

return TextUtil.*isEmpty*(groupDisplayName) ? onBindNickName(context, mRecipients) : groupDisplayName;

}

@Override

public void onBindAvatarView(Context context, ImageView avatar, String userId) {

LogUtil.*d*(*TAG*, "onBindAvatarView " + userId);

RXPhotoGlideHelper.*display*(context, userId, avatar, false);

}

@Override

public void OnAvatarClickListener(Context context, String id) {

}

@Override

public String onBindAvatarByUrl(Context context, String userId) {

LogUtil.*d*(*TAG*, "onBindAvatarByUrl " + userId);

RXEmployee employee = RXContactHelper.*getInstance*().getRXEmployee(userId);

if (employee == null) {

return "";

}

return employee.getUrl();

}

//...

#### 6.2 建议配置

##### 1)聊天界面底部面板的扩展

**作用位置：**

**使用场景：**如果开发者需要扩展聊天页面底部面板的功能时，请在[IMConfiguration](#_1)_推荐配置方式一)中按照本步骤进行配置

###### 创建IMPanel实例

//此Bean是 聊天界面底部面板的基类

IMPanel imPanel = new IMPanel

/\*\*

\* **@params** String panel 的名字

\* **@params** int 图片的资源id

\* **@params** int panel的类型 三种选择

\* IMPanel.PANELTYPE.SINGLECHAT, 表示添加到单聊

\* IMPanel.PANELTYPE.GROUPCHAT,表示添加到群聊

\* IMPanel.PANELTYPE.BOTH，表示两者均添加

\*/

.PanelBuilder("panel的名字", R.drawable.ic\_launcher, IMPanel.PANELTYPE.GROUPCHAT)

//此方法为 设置panel 点击事件

.setOnIMPannelClickListener(new OnIMPanelClickListener() {

/\*\*

\* 此方法是为 设置扩展条目的 点击事件

\* **@param** context

\* **@param** contactId单聊时代表聊天对象的userId 群聊时代表群组的groupId

\*/

@Override

public void onIMPannelClick(Context context, String contactId) {

//此处开发者进行点击事件的处理 示例：

ToastUtil.showMessage("测试");

}

}).build();

###### 添加到addPanel()

/\*\*

\* 此方法为了 把开发者创建的IMPanel实例添加到聊天界面底部面板

\* **@params** ImPanel...为可变数组，可填入多个IMPanel对象

\*/

.addPanel(imPanel)

##### 2)设置消息转发接口

**作用：当收到IM消息之后可以自定义实现该消息的处理方式(1.开发者处理该消息，2.交给插件内部处理)**

// IM消息转发接口

.setOnMessagePreproccessListener(mOnMessagePreproccessListener)

private OnMessagePreproccessListener mOnMessagePreproccessListener = new OnMessagePreproccessListener() {

@Override

public boolean dispatchMessage(ECMessage message) {

// 返回true意义是消费当前这条消息不交给插件内部处理,返回false意义是交给插件进行处理

if (message.getForm().equals("10086")) {

LogUtil.*d*(TAG, "dispatchMessage 10086...");

return true;

}

return false;

}

};

##### 配置消息已读未读状态

**作用：参数值**

**true 表示开启已读未读功能（该功能对私有云插件有效，对公有云插件无效）**

**false 表示关闭已读未读功能（可不调用，默认为关闭状态）**

##### 4) 配置未读消息数量改变的监听

**作用位置：**



**注意事项：**结合文档[5.1](#_5.1_获取未读消息)使用

**使用场景：**如果开发者需要根据未读消息数量来更新某个界面(如ConversationActivity)的控件，那么该页面需要实现OnUpdateMsgUnreadCountsListener接口，在OnUpdateMsgUnreadCounts()中调用文档[5.1](#_4.2.1_获取未读消息)后获取到消息数量，然后更新控件角标。

/\*\*

\* 此监听用于未读消息数量变化

\* 用法：需要根据未读消息数量来对界面UI刷新的页面实现

\* OnUpdateMsgUnreadCountsListener 此方法结合文档[4.2.1](#_4.2.1_获取未读消息)使用

\*/

public class ConversationActivity extends Activity implements OnUpdateMsgUnreadCountsListener {

@Override

public void OnUpdateMsgUnreadCounts() {

//里面调用方法获取未读消息数量之后，设置角标

//示例：结合 获取未读消息使用

int count = IMPluginManager.getManager().getUnReadMsgCount();

unReadMsg.setText("未读消息＝" + count);

}

//.........

}

##### 5) 配置点对点音视频插件

###### 先参考VoIP插件开发指南文档集成到IM工程中（已经集成可忽略，注意通话来电权限）

###### 实现接口

// 实现VoIP插件的IVoipCallBack接口在onCallFinish调用IM方法

*/\*\**

*\* 通话结束*

*\* <p>*

*\* int STATE\_100 = 100;//正常通话结束(通话时长)*

*\* int STATE\_101 = 101;//主动挂机并且未接通 ===已取消*

*\* int STATE\_102 = 102;//被叫方拒绝===对方已拒绝*

*\* int STATE\_201 = 201;//主叫方取消并且未接通===对方已取消*

*\* int STATE\_202 = 202;//自己拒绝==已拒绝*

*\**

*\** ***@param*** *entry 通话信息*

*\*/*

@Override

public void onCallFinish(CallFinishEntry entry) {

LogUtil.d(TAG, "[onCallFinish] entry:" + entry);

String bodyText = null;

int state = entry.status;

if (state == 100) {

// 通话正常结束

bodyText = "通话结束 " + DateUtil.formatCallTime((entry.endTime - entry.startTime) / 1000);

} else if (state == 101) {

// 主动取消(未接通)

bodyText = "已取消";

} else if (state == 102) {

// 被叫拒绝

bodyText = "对方已拒绝";

} else if (state == 201) {

// 主叫方取消(未接通)

bodyText = "对方已取消";

} else if (state == 202) {

// 主动拒绝

bodyText = "已拒绝";

}

IMChattingHelper.getInstance().insertVoIPCallMessage(entry.remote, entry.self, entry.isComingCall, entry.callType, bodyText);

}