# 1.云服务器数据库连接

（1）租用阿里云服务器，部署mysql数据库

（2）安装相关外部库

mysql-connector-java-5.1.49.jar对应mysql5.7版本

（3）DBUntil类中通过JDBC进行配置连接

**private static final** String ***databaseName*** = **"android\_test"**; *// 数据库名称***private static final** String ***ip*** = **"rm-cn-pe339dysm0016r8o.rwlb.rds.aliyuncs.com"**; *// 主机名***private static final** String ***port*** = **"3306"**; *// 端口号 例如 mysql 的默认端口号为 3306***private static final** String ***user*** = **"testuser"**;*// 用户名***private static final** String ***password*** = **"Qwe13681045826"**;*// 密码***private static** String *url* = **"jdbc:mysql://"** + ***ip*** + **":"** + ***port*** + **"/"** + ***databaseName***;  
*//注册msql驱动***static**{  
 **try**{  
 Class.*forName*(**"com.mysql.jdbc.Driver"**);  
 Connection conn = DriverManager.*getConnection*(*url*,***user***,***password***);  
 }**catch** (ClassNotFoundException | SQLException e){  
 e.printStackTrace();  
 }  
}

（4）类User中定义用户表映像

**public class** User {  
 **private int id**;  
 **private** String **name**;  
 **private** String **password**;  
  
 **public int** getId() {  
 **return id**;  
 }  
  
 **public void** setId(**int** id) {  
 **this**.**id** = id;  
 }  
  
 **public** String getName() {  
 **return name**;  
 }  
  
 **public void** setName(String name) {  
 **this**.**name** = name;  
 }  
  
 **public** String getPassword() {  
 **return password**;  
 }  
  
 **public void** setPassword(String password) {  
 **this**.**password** = password;  
 }  
}

（5）类UserDao实现相关数据库操作函数

例如根据输入的登录用户和密码在数据库中进行查询，是否存在

**public static** User login (String name, String password) {  
 Connection conn = DBUtil.*getConnection*();  
 String sql = **"select \* from user where name=? and password=?"**;  
 User user = **null**;  
 PreparedStatement ps = **null**;  
 **try** {  
 ps = conn.prepareStatement(sql);  
 ps.setString(1,name);  
 ps.setString(2,password);  
 ResultSet rs = ps.executeQuery();  
 *// bean d导入* **if**(rs.next()){  
 user = **new** User();  
 user.setName(rs.getString(**"name"**));  
 user.setPassword(rs.getString(**"password"**));  
 }  
 DBUtil.*release*(rs, **null**, ps, conn);  
 }**catch** (Exception e){  
 e.printStackTrace();  
 }  
 **return** user;

# socket连接及心跳包机制

服务器端首先创建客户端线程管理类，保存连接的客户端信息：存储客户端的地址和对应的Socket对象。

创建聊天服务线程类，创建ServerSocket监听客户端请求，保存到客户端线程管理类中，并监听指定端口提供 startServer() 方法，用于启动服务器。该方法会创建一个新的 ServerThread 对象并启动它的线程。

拆分客户端发送的信息：

sbText.append(split[0]);//端口

sbText.append("//");

sbText.append(split[1]);//所有ip地址和端口列表/数据

sbText.append("//");

sbText.append(split[2]);//时间/本地ip地址端口

sbText.append("//");

sbText.append(split[3]);//选项

根据选项的不同调用不同的功能执行0查询，1私聊，2群发。

0：端口//所有ip地址和端口列表//本地ip地址端口//0

1：端口//数据//本地ip地址端口//1

2：端口//数据//时间//2

提供 shutdown() 方法，用于关闭服务器和清空 clientList 中的客户端连接。

提供 sendMsgAll() 方法，用于将消息群发给所有客户端。提供 sendMsg()用于私发给指定ip地址对象

客户端创立socket根据服务器端ip和端口连接

socket = new Socket("10.0.2.2", 10001);

定义输入流，不断从输入流中读取数据，通过meesage机制返回给定义的handler处理

InputStream inputStream = socket.getInputStream();

byte[] buffer = new byte[1024];

int len;

while ((len = inputStream.read(buffer)) != -1) {

// 循环不断从bufer中读取数据

// 通过 inputStream.read(buffer) 方法返回实际读取的字节数。

String data = new String(buffer, 0, len);

// 发到主线程中 收到的数据

Message message = Message.obtain();

// 实例化

message.what = 1;

message.obj = data;

all\_handler.sendMessage(message);

// 将消息发送给主线程进行处理和显示

}

可以通过获得的文本，点击按钮后通过输出流按对应格式发送给服务器处理。

**all\_bt**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 **final** String data = **all\_et**.getText().toString();  
*// client.send(0,data);* **new** Thread(**new** Runnable() {  
 @Override  
 **public void** run() {  
 **try** {  
  
  
 Socket s=GlobalVariables.*getInstance*().getSelf\_socket();  
  
 OutputStream outputStream=**os**;  
*// OutputStream outputStream = socket.getOutputStream();  
// 0群发1私聊2查询* SimpleDateFormat df = **new** SimpleDateFormat(**"HH:mm:ss"**); *//设置日期格式* outputStream.write((**socket**.getLocalPort() +**"//"** + data + **"//"** + df.format(**new** Date()) +**"//"** + **"0"**).getBytes(**"utf-8"**));  
 outputStream.flush();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
 }).start();  
 }

心跳包机制断线重连机制，客户端每隔5秒发送一个心跳信号，服务器根据收到的数据判断连接是否正常，当客户端检测到连接断开未收到服务器的回应后，进行断线重连。

class Client\_heart implements Runnable{

private Socket socket;

private ObjectOutputStream oos;

Client\_heart(Socket socket, ObjectOutputStream oos){

this.socket = socket;

this.oos = oos;

}

@Override

public void run() {

try {

System.out.println("心跳包线程已启动...");

while (true){

Thread.sleep(5000);

JSONObject object = new JSONObject();

object.put("type","heart");

object.put("msg","心跳包");

oos.writeObject(object);

oos.flush();

}

}catch (Exception e){

e.printStackTrace();

try {

socket.close();

Client.connection\_state = false;

Client.reconnect();

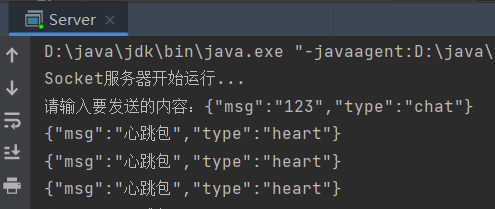
}catch (Exception ee){

ee.printStackTrace();

}

}

}



# Fragmnet导航

在activity\_fragment.xml中，设置导航栏的布局。

在使用 <selector> 文件时出现报错，是由于文件位置错误：选择器文件的要放在 res/drawable 或 res/color 目录下。

推荐使用 FragmentManager 的替代类 FragmentMaangerX。

创建MyFragment类，创建一个继承自Fragment的子类，并实现onCreateView()方法来定义Fragment的布局和界面元素。在onCreateView()方法中，通过LayoutInflater来加载Fragment的布局文件。

在onCreateView中通过定义if条件，来根据用户点击导航项后，FragmentActivity中返回的信息，执行不同的功能。通过inflater.inflate绑定对应的xml功能布局。然后可以像一个正常活动一样写代码。

if(content.equals("第一个Fragment")){

// 在线聊天：所有人一起聊天

view = inflater.inflate(R.layout.all\_chat,container,false);

获取上下文信息则是通过getActivity()，而不是this.

在编写一个FragmentActivity类，操控导航栏，通过FragmentTransaction 实例，用于处理 Fragment 相关的操作。之后可以使用该实例执行添加、显示和隐藏 Fragment 的操作，并通过调用 commit() 方法提交事务。

FragmentTransaction fTransaction = getSupportFragmentManager().beginTransaction();

hideAllFragment(fTransaction);

根据点击的导航项，执行MyFragment中的函数

if(v.getId()==R.id.fg\_list){

setSelected();

fglist.setSelected(true);

if(fg1 == null){

fg1 = new MyFragment("第一个Fragment");

fTransaction.add(R.id.fg,fg1);

}else{

fTransaction.show(fg1);

}

}

fTransaction.commit();

#### 类FragmentActivity

操控导航栏

类MyFragment 核心界面的核心代码，四个模块的操作都在这里。

点击第一个导航项：

在线程中连接服务器，不断接收数据，并将数据通过消息机制传回主线程处理。

**new** Thread(**new** Runnable() {  
 @Override  
 **public void** run() {  
 **try** {  
*// socket = new Socket("127.0.0.1", 10000);  
// socket = new Socket("10.0.2.2", 9999);* **socket** = **new** Socket(**"10.0.2.2"**, 10001);  
*////// 申请连接* Log.*i*(**"提示"**,**"连接成功"**);  
*// 把这个连接移动到共享部分  
// socket = new Socket("10.0.2.2", 10001);  
//// 申请连接  
// Log.i("提示","连接成功");* InputStream inputStream = **socket**.getInputStream();  
 **byte**[] buffer = **new byte**[1024];  
 **int** len;  
 **while** ((len = inputStream.read(buffer)) != -1) {  
*// 循环不断从bufer中读取数据  
// 通过 inputStream.read(buffer) 方法返回实际读取的字节数。* String data = **new** String(buffer, 0, len);  
 *// 发到主线程中 收到的数据* Message message = Message.*obtain*();  
*// 实例化* message.**what** = 1;  
 message.**obj** = data;  
  
 all\_handler.sendMessage(message);  
*// 将消息发送给主线程进行处理和显示* }  
  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
 }).start();

绑定按钮：用户输入信息点击，发送信息给服务端。设置发送信息类型*0群发1私聊2查询*

发送信息：outputStream.write((**socket**.getLocalPort() +**"//"** + data + **"//"** + df.format(**new** Date()) +**"//"** + **"0"**).getBytes(**"utf-8"**));

**all\_bt**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 **final** String data = **all\_et**.getText().toString();  
*// client.send(0,data);* **new** Thread(**new** Runnable() {  
 @Override  
 **public void** run() {  
 **try** {  
  
 OutputStream outputStream = **socket**.getOutputStream();  
*// 0群发1私聊2查询* SimpleDateFormat df = **new** SimpleDateFormat(**"HH:mm:ss"**); *//设置日期格式* outputStream.write((**socket**.getLocalPort() +**"//"** + data + **"//"** + df.format(**new** Date()) +**"//"** + **"0"**).getBytes(**"utf-8"**));  
 outputStream.flush();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
 }).start();  
 }

在Handle中接收消息

**else**{  
*// 聊天室* Log.*i*(**"提示"**,**"聊天室开始"**);  
 **if** (split[0].equals(localPort + **""**)) {  
 ChatMsg chatMsg = **new** ChatMsg(split[1], 0, split[2], **"我："**);  
 **all\_list**.add(chatMsg);  
 } **else** {  
 ChatMsg chatMsg = **new** ChatMsg(split[1], 1, split[2], (**"来自："** + split[0]));  
 **all\_list**.add(chatMsg);  
 }  
 Log.*i*(**"提示"**,**"输出信息ChatHandler:"**+**all\_list**.get(0).getMsg());  
  
 *// 向适配器set数据* **all\_adapter**.setData(**all\_list**);  
 **all\_rv**.setAdapter(**all\_adapter**);  
 LinearLayoutManager manager = **new** LinearLayoutManager(getActivity(), LinearLayoutManager.***VERTICAL***, **false**);  
 **all\_rv**.setLayoutManager(manager);  
*// 创建一个 LinearLayoutManager 对象，并将其设置为 RecyclerView 的布局管理器，用于控制列表项的显示方式。* }

点击第二个导航项：

点击刷新按钮，向服务器发送查询信息。

outputStream.write((**socket**.getLocalPort() + **"//"** + **"查询"** + **"//"** + df.format(**new** Date()) + **"//"** + **"2"**).getBytes(**"utf-8"**));

服务器返回目前连接的所有客户端ip。

在GlobalVariables中存储。

设置点击事件，点击在线用户后找到指定ip给服务器发送信息，要求服务器进行私聊。

**list\_bu**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 Log.*i*(**"提示"**,**"刷新通讯录列表"**);  
  
 **new** Thread(**new** Runnable() {  
 @Override  
 **public void** run() {  
 **try** {  
 Log.*i*(**"提示"**,**"准备发出"**);  
*// 发送服务器查询请求2* OutputStream outputStream = **socket**.getOutputStream();  
 Log.*i*(**"提示"**,**"重新定义输出流"**);  
 SimpleDateFormat df = **new** SimpleDateFormat(**"HH:mm:ss"**); *//设置日期格式  
// 可以选择在日期的地方改为ip* outputStream.write((**socket**.getLocalPort() + **"//"** + **"查询"** + **"//"** + df.format(**new** Date()) + **"//"** + **"2"**).getBytes(**"utf-8"**));  
 outputStream.flush();  
 Log.*i*(**"提示"**,**"发出了查询"**);  
  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
 }).start();

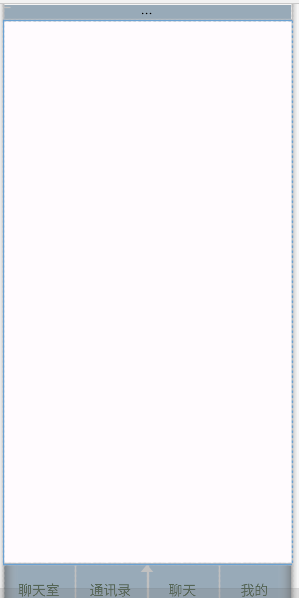
#### tab\_menu\_text.xml

设置导航项的选择器

<**selector xmlns:android="http://schemas.android.com/apk/res/android"**>  
 <**item android:color="#4CAF50" android:state\_selected="true"** />  
 <**item android:color="#546555"** />  
</**selector**>

#### activity\_fragment.xml

设置导航栏布局



# 4.RecycleView

添加recycleView的依赖，在build.gradle里添加语句，implementation 'androidx.recyclerview:recyclerview:1.0.0'

创立类AllAdapter聊天室消息界面适配器。通过判断时我发送的消息还是他人发送的消息，来设置不同列表项，加载到RecyclerView列表中。

创立allchat.xml布局文件，其中通过RecyclerView来加载聊天记录。

#### 类AllAdapter

聊天室消息界面适配器

通过判断时我发送的消息还是他人发送的消息，来设置不同列表项，加载到RecyclerView列表中。

重写onCreateViewHolder函数用来创建ViewHolder实例，再将加载好的布局传入构造函数，最后返回ViewHolder实例。它表示每个列表项的视图容器，用于保存列表项中的视图控件引用。根据指定的布局文件创建一个视图对象，以便在 RecyclerView 的列表项中使用。 通过调用 inflate() 方法，将布局文件转换为一个视图对象，并将其存储在 view 变量中供后续使用。

**if** (viewType==***T\_I***){  
 View view = LayoutInflater.*from*(**context**).inflate(R.layout.*all\_mymsg\_layout*,parent,**false**);  
 holder = **new** T\_IViewHolder(view);

**public void** onBindViewHolder(@NonNull RecyclerView.ViewHolder holder, **int** position) {  
  
 **int** itemViewType = getItemViewType(position);  
 **if** (itemViewType==***T\_I***) {  
 T\_IViewHolder myViewHolder = (T\_IViewHolder) holder;  
 myViewHolder.**my\_tv**.setText(**Msg**.get(position).getMsg());  
 myViewHolder.**my\_tv2**.setText(**Msg**.get(position).getName());  
 }  
 **else**{  
 T\_HeViewHolder HisViewHolder = (T\_HeViewHolder) holder;  
 HisViewHolder.**his\_tv**.setText(**Msg**.get(position).getMsg());  
 HisViewHolder.**his\_tv2**.setText(**Msg**.get(position).getName());  
 }  
}

返回指定位置的列表项类型

@Override  
 **public int** getItemViewType(**int** position) {  
 **return Msg**.get(position).getNumber();  
 }

获取目前列表尺寸

public int getItemCount() {

int c=0;

if(Msg != null&& Msg.size() > 0){

c=Msg.size();

}

return c;

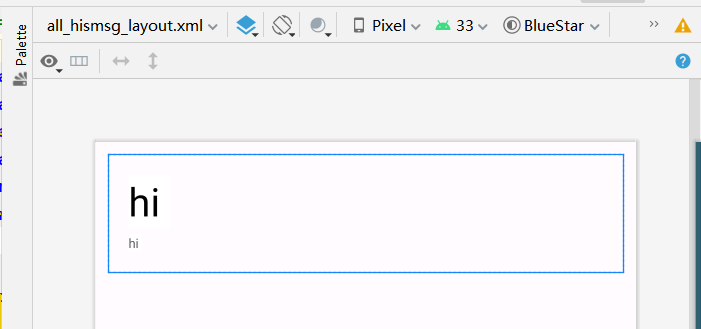
}

#### all\_chat.xml



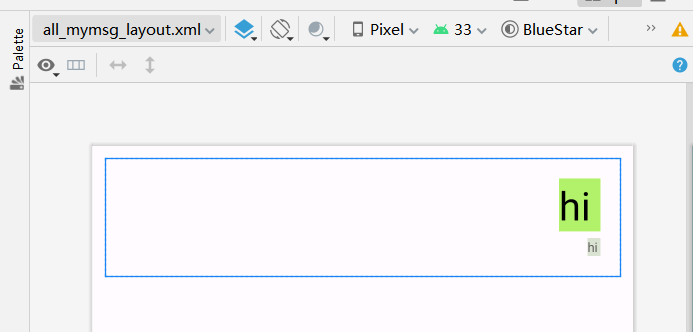
#### all\_hismsg\_layout.xml

聊天室中其他人发布的消息，显示发布人名称



#### all\_mymsg\_layout.xml

聊天室中我发布的消息，显示发布人名称：我



# 5.Message线程传递消息机制

在子线程中不断监听服务端输入流得到的数据，通过message对象来传给主线程中的Handler进行处理

**new** Thread(**new** Runnable() {  
 @Override  
 **public void** run() {  
 **try** {  
*// socket = new Socket("127.0.0.1", 10000);  
// socket = new Socket("10.0.2.2", 9999);* **socket** = **new** Socket(**"10.0.2.2"**, 10001);  
*////// 申请连接* Log.*i*(**"提示"**,**"连接成功"**);  
*// 把这个连接移动到共享部分  
// socket = new Socket("10.0.2.2", 10001);  
//// 申请连接  
// Log.i("提示","连接成功");  
  
// GlobalVariables.getInstance().setSelf\_socket(socket);  
// Socket s=GlobalVariables.getInstance().getSelf\_socket();  
// 起不到任何用处  
  
// 如果我在建立连接时，就定义输出流，是不是可以解决问题* **os**=**socket**.getOutputStream();  
  
 InputStream inputStream = **socket**.getInputStream();  
 **byte**[] buffer = **new byte**[1024];  
 **int** len;  
 **while** ((len = inputStream.read(buffer)) != -1) {  
*// 循环不断从bufer中读取数据  
// 通过 inputStream.read(buffer) 方法返回实际读取的字节数。* String data = **new** String(buffer, 0, len);  
 *// 发到主线程中 收到的数据* Message message = Message.*obtain*();  
*// 实例化* message.**what** = 1;  
 message.**obj** = data;  
  
 all\_handler.sendMessage(message);  
*// 将消息发送给主线程进行处理和显示* }  
  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
 }).start();

在主线程中定义Handler对象，接收子线程中发送的消息，然后对消息进行拆分，根据不同类别，实现群聊，私聊，通讯录页面UI修改。

**private class** AllChatHandler **extends** Handler {  
 @Override  
 **public void** handleMessage(Message msg) {  
 **super**.handleMessage(msg);  
 **if** (msg.**what** == 1) {  
 **int** localPort = **socket**.getLocalPort();  
*// int localPort=GlobalVariables.getInstance().getSelf\_socket().getLocalPort();  
// 获取socket本地端口号* String[] split = ((String) msg.**obj**).split(**"//"**);  
 **if**(Objects.*equals*(split[3], **"2"**)){  
*// 查询* String[] onlineAddress =split[1].split(**"\_"**);  
 GlobalVariables.*getInstance*().setSelfAddress(split[2]);  
 GlobalVariables.*getInstance*().setChatobject(onlineAddress);  
 Log.*i*(**"提示"**,**"查询开始"**);  
  
*// 将适配器绑定  
  
// 突然发现我可以很轻松的实现自动刷新页面啊,等测试如果没有bug就用这个  
// list\_adapter.setData(list);  
// list\_rv.setAdapter(list);  
// LinearLayoutManager manager = new LinearLayoutManager(getActivity(), LinearLayoutManager.VERTICAL, false);  
// list\_rv.setLayoutManager(manager);  
 // 向适配器set数据  
// all\_adapter.setData(all\_list);* **list\_adapter1**.setData(GlobalVariables.*getInstance*().getChatobject());  
 **list\_rv**.setAdapter(**list\_adapter1**);  
 LinearLayoutManager manager = **new** LinearLayoutManager(getActivity(), LinearLayoutManager.***VERTICAL***, **false**);  
 **list\_rv**.setLayoutManager(manager);  
*// 创建一个 LinearLayoutManager 对象，并将其设置为 RecyclerView 的布局管理器，用于控制列表项的显示方式。  
// 点击事件方法* **list\_adapter1**.setOnItemClickListener(**new** listAdapter1.OnItemClickListener() {  
 @Override  
 **public void** onItemClick(**int** position) {  
 onListItemClick(position);  
 }  
 });  
  
 }  
 **else if** (Objects.*equals*(split[3], **"1"**)){  
*// 私聊* Log.*i*(**"提示"**,**"私聊开始"**);  
 String[] onlineAddress =split[1].split(**"\_"**);  
  
  
 }  
 **else**{  
*// 聊天室* Log.*i*(**"提示"**,**"聊天室开始"**);  
 **if** (split[0].equals(localPort + **""**)) {  
 ChatMsg chatMsg = **new** ChatMsg(split[1], 0, split[2], **"我："**);  
 **all\_list**.add(chatMsg);  
 } **else** {  
 ChatMsg chatMsg = **new** ChatMsg(split[1], 1, split[2], (**"来自："** + split[0]));  
 **all\_list**.add(chatMsg);  
 }  
 Log.*i*(**"提示"**,**"输出信息ChatHandler:"**+**all\_list**.get(0).getMsg());  
  
 *// 向适配器set数据* **all\_adapter**.setData(**all\_list**);  
 **all\_rv**.setAdapter(**all\_adapter**);  
 LinearLayoutManager manager = **new** LinearLayoutManager(getActivity(), LinearLayoutManager.***VERTICAL***, **false**);  
 **all\_rv**.setLayoutManager(manager);  
*// 创建一个 LinearLayoutManager 对象，并将其设置为 RecyclerView 的布局管理器，用于控制列表项的显示方式。* }  
 }  
 }  
 }

# 定义全局变量

通过单例模式建立全局类，其中包含要共享的变量，并提供公共的访问方法set/get。

**public class** GlobalVariables {  
 **private static** GlobalVariables *instance*;  
 **private int counter** = 0;  
 **private** String[] **Chatobject**=**null**;  
 **private** String **SelfAddress**=**null**;  
  
 **private** Client **client**=**null**;  
  
 **private** Socket **self\_socket**=**null**;  
  
 **private** GlobalVariables() {  
 *// 私有构造函数* }  
  
 **public static synchronized** GlobalVariables getInstance() {  
 **if** (*instance* == **null**) {  
 *instance* = **new** GlobalVariables();  
 }  
 **return** *instance*;  
 }  
  
 **public int** getCounter() {  
 **return counter**;  
 }  
 **public void** setCounter(**int** value) {  
 **counter** = value;  
 }  
  
 **public** String[] getChatobject() {  
 **return Chatobject**;  
 }  
 **public void** setChatobject(String[] list) {  
 **Chatobject** = list;  
 }  
  
 **public** String getSelfAddress() {  
 **return SelfAddress**;  
 }  
 **public void** setSelfAddress(String address) {  
 **SelfAddress** = address;  
 }  
  
 **public** Socket getSelf\_socket() {  
 **return this**.**self\_socket**;  
 }  
 **public void** setSelf\_socket(Socket self\_socket) {  
 **this**.**self\_socket** = self\_socket;  
 }  
}

通过定义的全局类中提供的静态公共方法：赋值与获取方法来在代码中使用

GlobalVariables.*getInstance*().setSelf\_socket(**socket**);  
Socket s=GlobalVariables.*getInstance*().getSelf\_socket();