计算机网络实验二：建立聊天工具

1617301 061710402 陈冠余

项目简介

本项目采用Java语言编写，连接本地MySql数据库，前端采用swing组件。核心是socket编程中的流套接字技术，其中C/S模式使用Socket，P2P模式使用DatagramSocket。

文件基本结构如下：

```
|--com.simplechat
  |--client
    |--View                  //构建展示窗口
    |--NetworkService        //一个客户端进程有两个线程，一个监听服务端的消息，一个监听客户端的消息
  |--server
    |--ChatSocket            //实现服务端对单个进程的请求的处理
    |--ClientManager         //实现服务端对多个进程交互的处理
    |--SimpleChatService     //开关服务端
    |--View                  //服务端的窗口，仅有开关服务器
  |--database
    |--DBconn                //处理对数据库的连接操作
    |--user                  //实体类，对应数据库中的表，每个对象对应表中的一条记录。好像没用
    |--message
    |--process
```

Socket

在计算机通信领域，socket 被翻译为“套接字”，它是计算机之间进行通信的一种约定或一种方式。通信三元组：协议+IP地址+端口。

Socket是应用层与TCP/IP协议族通信的中间软件抽象层，它是一组接口。在设计模式中，Socket其实就是一个门面模式，它把复杂的TCP/IP协议族隐藏在Socket接口后面，对用户来说，只要调用接口，就能按照协议约定来组织数据。

以下通过代码实现来介绍它的工作流程，其中省略了一些无关的功能实现代码。

采用TCP协议：socket

连接：（对应TCP三次握手）

```
/*客户端*/
    public void connect(String host, int port) {    //服务端的IP地址+端口号
        try {
            socket = new Socket(host, port);        // 创建套接字对象，与服务器建立连
接
            beginListening();                       // 开始侦听是否有聊天消息到来
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    private void beginListening() {
        Runnable listening = new Runnable() {       //通过Runnable接口实现多线程
            @Override
            public void run() {
                try {
                    //接收来自服务端的消息
                    inputStream = new DataInputStream(socket.getInputStream());
                    while (true) {
                        String msg = inputStream.readUTF();
                        //判断消息类型，进行处理
                    }
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        };
        (new Thread(listening)).start();            //启动线程
    }

/*服务端*/
    public void startup() {
        Thread thread = new Thread(new Runnable() { //依旧是在实现多线程
            @Override
            public void run() {
                runStartup();
            }

        });
        thread.start(); //启动线程
    }
    private void runStartup() {
        try {
            serverSocket = new ServerSocket(8765);  //服务端绑定特定端口号
            while (true) {                          //循环监听等待客户端连接
                Socket socket = serverSocket.accept();
                clientManager.addClientSocket(socket);//为每个新连接的客户端创建一个
子线程
            }
```

```
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

断开连接：（对应TCP四次握手）

```
/*客户端*/
    public void disconnect() {
        try {
            if (socket != null) {            //断开连接
                socket.close();
            }
            if (inputStream!= null) {        //关闭输入流
                inputStream.close();
            }
            if (outputStream != null) {      //关闭输出流
                outputStream.close();
            }
            isConnected = false;             // 通知外界连接断开
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

/*服务端*/
    public void shutdown() {
        try {
            clientManager.close();
            serverSocket.close();            //关闭服务端连接
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    //clientManager类中的函数
    public void close() throws IOException {
        synchronized (chatSockets) {         //关闭各个进程的连接
            for (ChatSocket socket : chatSockets) {
                socket.close();
            }
            chatSockets.clear();
        }
    }
```

采用UDP协议：DatagramSocket

连接：

```
UDPsocket = new DatagramSocket();   //不用与服务端连接，随机分配端口号

//监听来自客户端的消息
private void beginListeningUDP() {
    Runnable listeningUDP = new Runnable() {
        @Override
        public void run() {
            try {
```

```
            byte[] buffer=new byte[256];
            DatagramPacket packet=null;
            while (true) {
                for(int i=0;i<buffer.length;i++)
                    buffer[i]=(byte)0;
                packet=new DatagramPacket(buffer, buffer.length);//构建数据报
                UDPsocket.receive(packet);                          //接收数据报
                int len =packet.getLength();
                String msg = new String(buffer,0,len);          //将数据包内容转化
为String
                //处理消息
            }
        } catch (IOException e) {
        e.printStackTrace();
        }
    };
    (new Thread(listeningUDP)).start();
}

//发送消息
DatagramPacket packet = new DatagramPacket(msg2.getBytes(),
msg2.getBytes().length, InetAddress.getByName(host),port);   //接收方客户端的host和
port
UDPsocket.send(packet);
```

断开连接：

```
if(UDPsocket != null){
    UDPsocket.close();
}
```

数据库

表的设计：

1.用户表：user

id，用户名name，密码password，电话phone，是否在线is_online(0或1)，好友friends（存id的字符串，如”gnc,“"gnc,cgy,"后面跟着逗号）。【is_online没用上】

2.消息表：message

id，发送者from_name，接收者to_name，内容content。

3.进程表：process

id，该进程的客户的用户名name，IP地址host，端口port。【host和port是采用UDP协议的DatagramSocket所连接的】

注意，为实现P2P，将IP地址和资源（name）写对应，写在一张process表里。对于UDPsocket，每次登录或注册时，需要将当前UDPsocket绑定的IP地址和端口号、当前进程的用户名，作为一条记录写入表中。

```
String host = UDPsocket.getLocalAddress().toString();
int port = UDPsocket.getLocalPort();
DBconn.init();
DBconn.addUpdDel("insert into process(name,host,port) " +
    "values('" + name + "','" + host + "','" + port + " ')");
DBconn.closeConn();
```

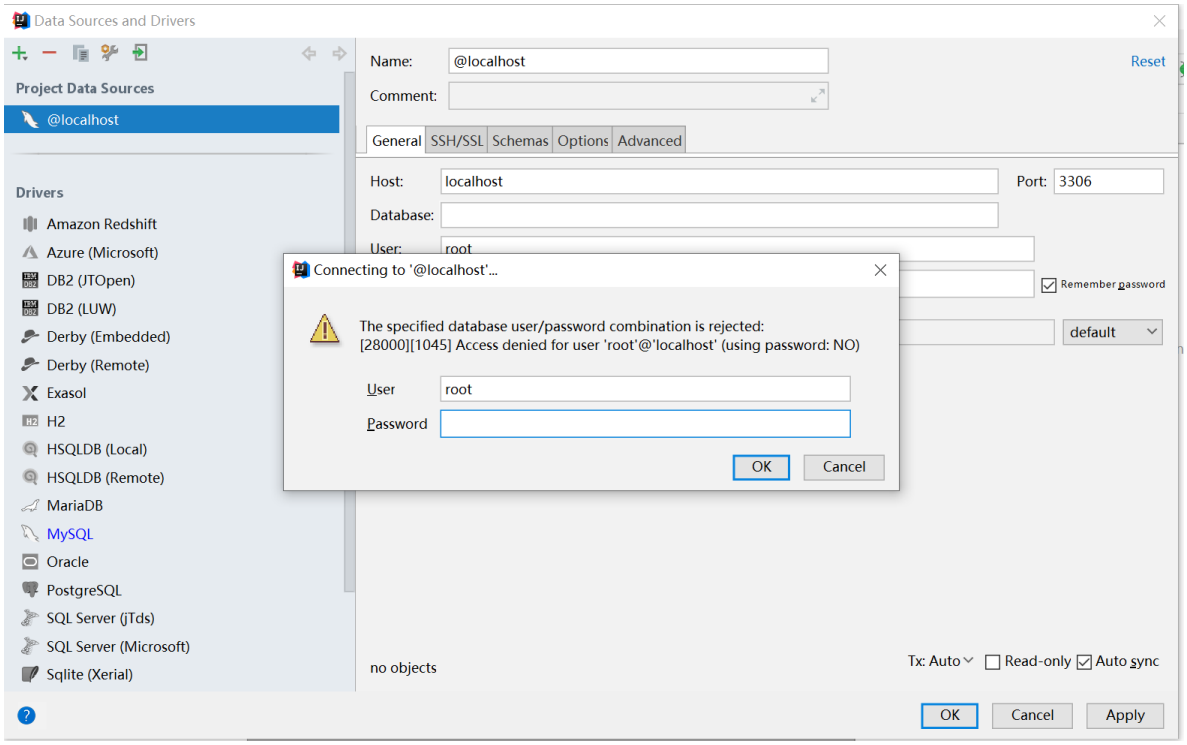
而每次断开连接，也就是进程结束，需要删去这一条记录。

```
DBconn.init();
DBconn.addUpdDel("delete from process where name='"+name+"'");
DBconn.closeConn();
```

```
+------------------+
| message          |
| process          |
| user             |
+------------------+
3 rows in set (0.00 sec)

mysql> desc user;
+-----------+-------------+------+-----+---------+----------------+
| Field     | Type        | Null | Key | Default | Extra          |
+-----------+-------------+------+-----+---------+----------------+
| id        | int(11)     | NO   | PRI | NULL    | auto_increment |
| name      | varchar(40) | NO   |     | NULL    |                |
| password  | varchar(40) | NO   |     | NULL    |                |
| phone     | varchar(40) | NO   |     | NULL    |                |
| is_online | int(11)     | NO   |     | NULL    |                |
| friends   | varchar(40) | NO   |     | NULL    |                |
+-----------+-------------+------+-----+---------+----------------+
6 rows in set (0.03 sec)

mysql> desc message;
+-----------+--------------+------+-----+---------+----------------+
| Field     | Type         | Null | Key | Default | Extra          |
+-----------+--------------+------+-----+---------+----------------+
| id        | int(11)      | NO   | PRI | NULL    | auto_increment |
| from_name | varchar(40)  | NO   |     | NULL    |                |
| to_name   | varchar(40)  | NO   |     | NULL    |                |
| content   | varchar(100) | NO   |     | NULL    |                |
+-----------+--------------+------+-----+---------+----------------+
4 rows in set (0.01 sec)

mysql> desc process;
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| id    | int(11)     | NO   | PRI | NULL    | auto_increment |
| name  | varchar(40) | NO   |     | NULL    |                |
| host  | varchar(40) | NO   |     | NULL    |                |
| port  | int(11)     | NO   |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
```

idea连接数据库，添加数据源，导入jar包。使用JDBC操作数据库。

以下是连接数据库的一张过程图：



功能演示及实现

以下功能实现是以窗口的切换流程为思路的，会尽量贴合给出的实验要求的顺序。实现写在演示后。

为了区分消息类型，规定消息的第一个字节表示消息类型：

login是‘a’，register是‘b’，加好友请求是‘c’，同意好友申请是‘d‘，不同意好友申请是‘e‘，拉群的消息是‘f’，群聊消息是‘g’，发起双人聊天消息是'h'， 双人聊天消息是‘i’，双人聊天的图片消息是‘j’。

客户端对消息的判断和处理在NetworkService中；

服务端对消息的判断和处理在ChatSocket中。

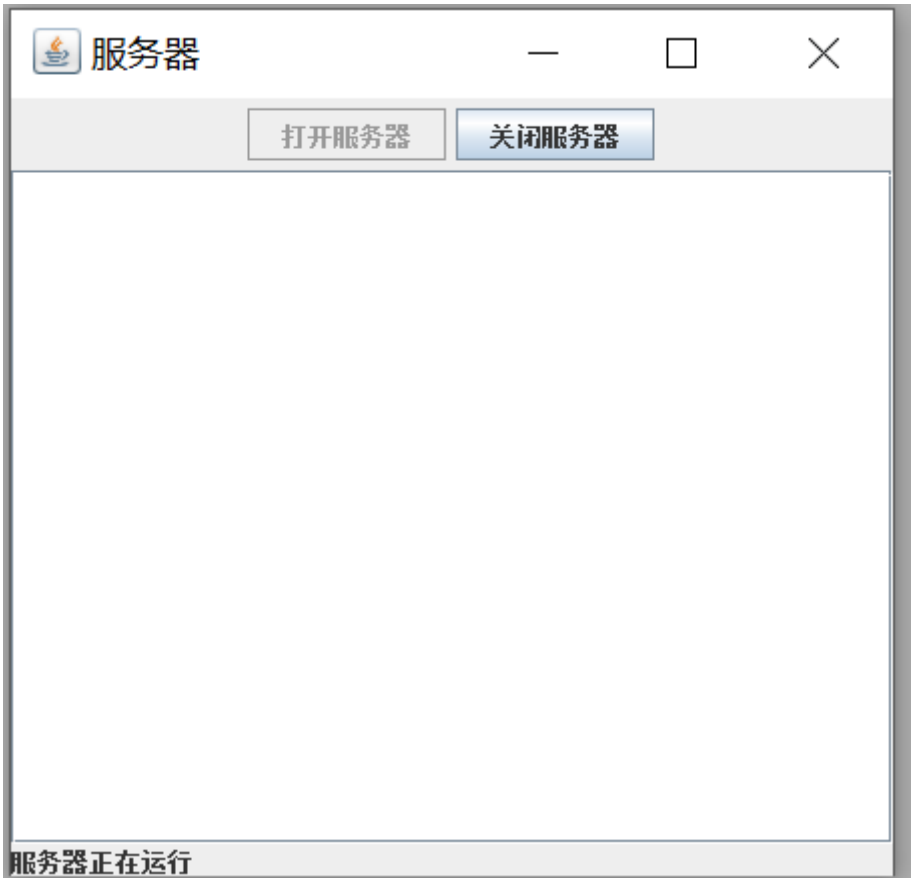
1.门户页面

也就是程序启动进入的页面。

首先，需要开启数据库。

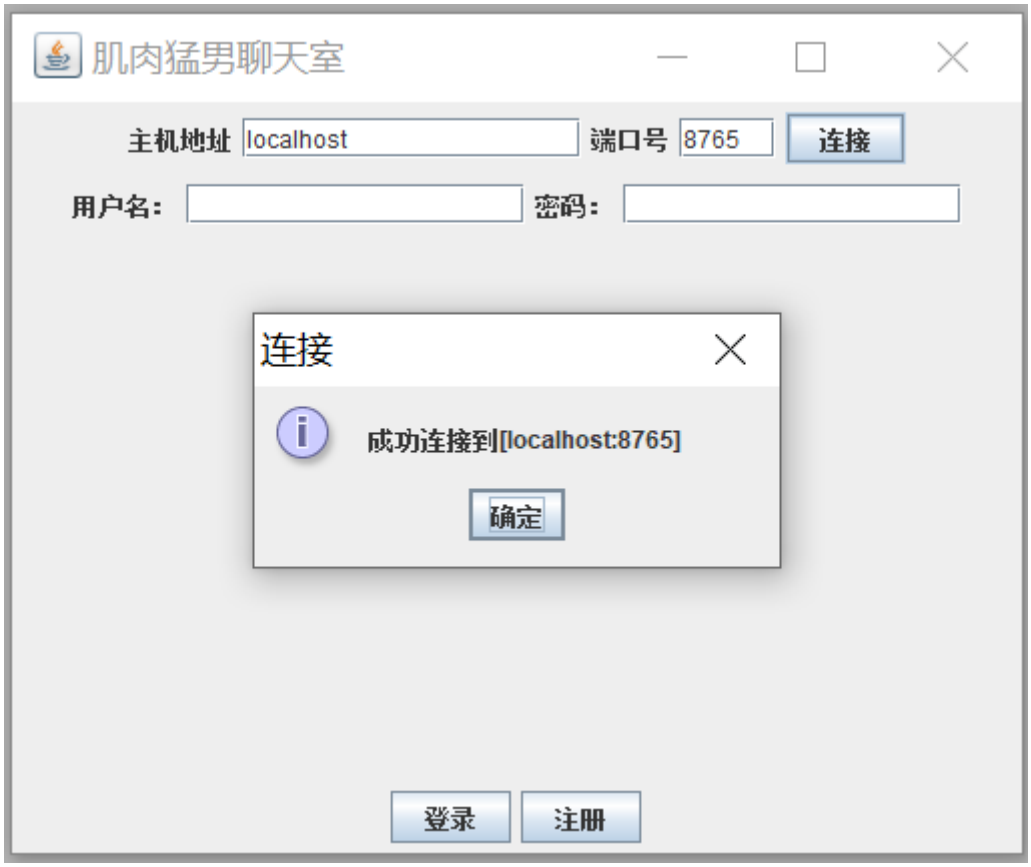
```
net start Mysql
Mysql -u root -p
```

接着，需要开启服务端：



接着，开启客户端，进入门户页面：

点击右上角“连接”，连接到服务器。之后方可进行登录/注册操作。



实现

连接的操作就同上面讲的socket操作。

用JFrame容器类，添加ActionListener监听，就能实现点击按钮调用方法。

一个窗口的基本代码为：

```
public class View extends JFrame implements
        ActionListener {
    private void initView() {
        btnConnect.addActionListener(this);
    }

    @Override
    public void actionPerformed(ActionEvent e){
    }

    //构造方法
    public View() {
        initView()
    }
}
```

要说明，在NetworkService类中定义了一个回调接口callback，用于在客户端收到各种不同类型的消息后做出反应。比如这里弹出的提示信息框，就是callback接口中的onConnected()方法实现的。

弹出提示信息框的代码是：

```
private void alert(String title, String message) {
    JOptionPane.showMessageDialog(this, message, title,
JOptionPane.INFORMATION_MESSAGE);
}
```

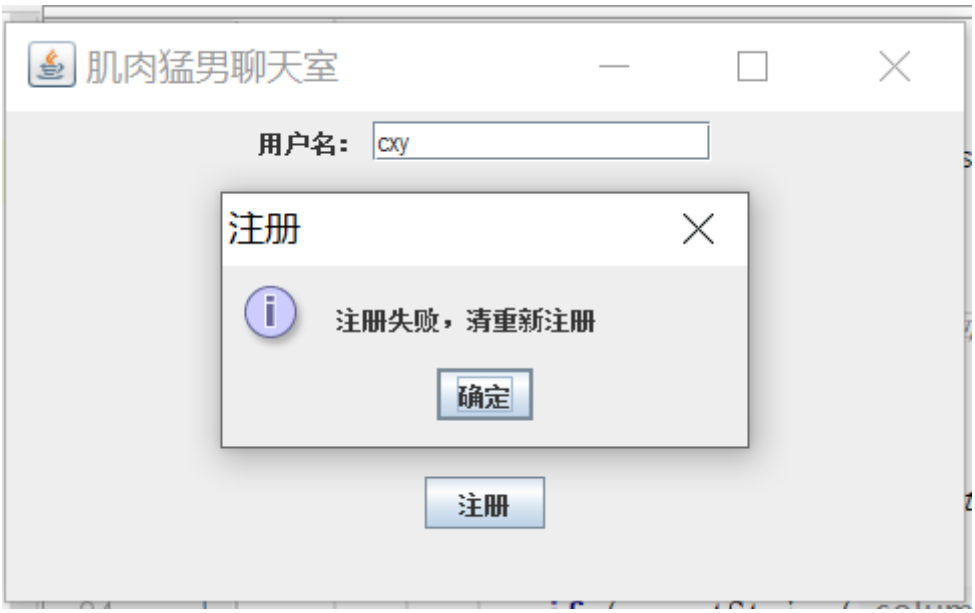
每次点击右上角的叉叉，就会调用networkService.disconnect()关闭连接。

2.注册

点击“注册”按钮，则从门户页面跳到注册页面。

注册需输入name、password、phone。is_online初始化为1，friends初始化为‘’。

注册经过服务器审核，失败则弹出消息如图。点击确定则窗口消息，回到登录页面。



实现

服务器审批在ChatSocket里，这里做得很简单，只审核了是否格式符合要求（不为空），以及是否已存在。并且这里为了实现简便，是用name去判断是否已存在的，也就是说user表中的name都是不重复的。

审批代码如下：isValid表示是否有效，经服务端传回客户端，调用callback接口的onRegistration()方法，在View中处理。如果为true则通过，false则不通过，弹出上图。

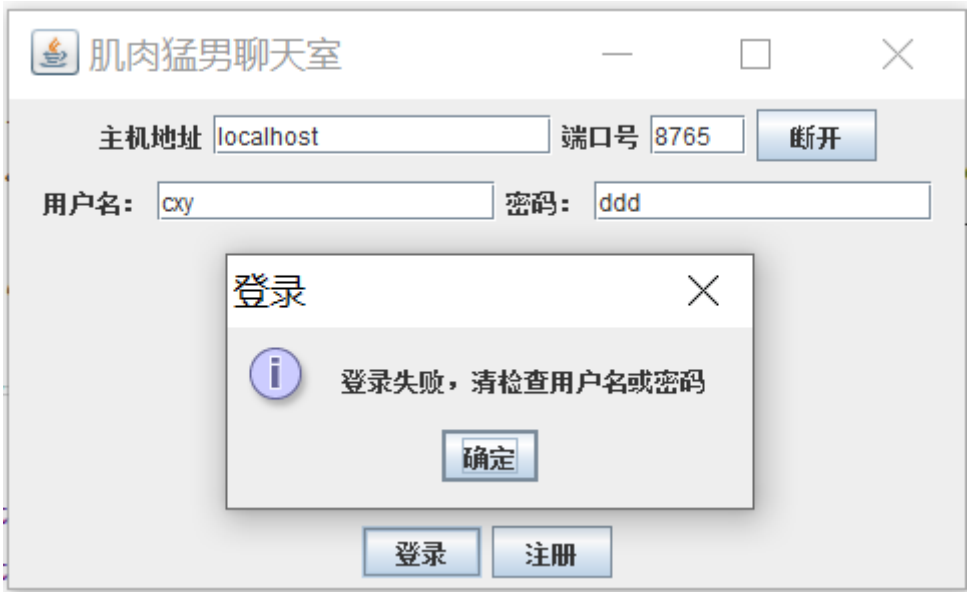
```
//检查格式，这里检查得非常简单，实际情况更复杂
if (s[0] == null || "".equals(s[0]) || s[1] == null || "".equals(s[1]) || s[2]
== null || "".equals(s[2])) {
    isValid="false";
}
 //检查是否已存在，若已存在则返回错误信息，理应做出提示“用户已存在”，但这里为实现简单不做提示
try {       //数据库查询，login等功能也用到
    DBconn.init();
    ResultSet rs = DBconn.selectSql("select * from user where name='" + s[0] +
"'");
    while (rs.next()) {
        if (rs.getString("name").equals(s[0]) ) {
            isValid="false";
            }
        }
    DBconn.closeConn();
    } catch (SQLException e) {
        e.printStackTrace();
    }
```

数据库插入代码，实现注册。

```
DBconn.addUpdDel("insert into user(name,password,phone,is_online,friends) " +
                "values('"+s[0]+"','"+s[1]+"','"+s[2]+"',"+1+",'"+" ')");
```

3.登录

登录成功则跳转到功能页面，失败则弹出失败信息如图。这里登录失败是因为密码输错了，不是”ddd“而是”xxx“。密码保存在本地数据库中。



实现

服务端登录验证的代码，和上面注册判isValid的一样，区别就是多了一个password作为查找条件。

页面跳转的实现代码：

```
FunctionView view = new FunctionView(host,port,name);
this.dispose(); //视情况而定，比如CheckView就不需要关闭现有窗口
```

要说明的是，为了保证在进入Portal点击"连接"后就一直保持连接，本项目将host和port作为参数传给新的窗口。但其实也并不是**一直**保持连接，只是在initNetworkService()中第一步是connect。在new新窗口时，调用构造函数，则initNetworkService()，保证新窗口也连接上。

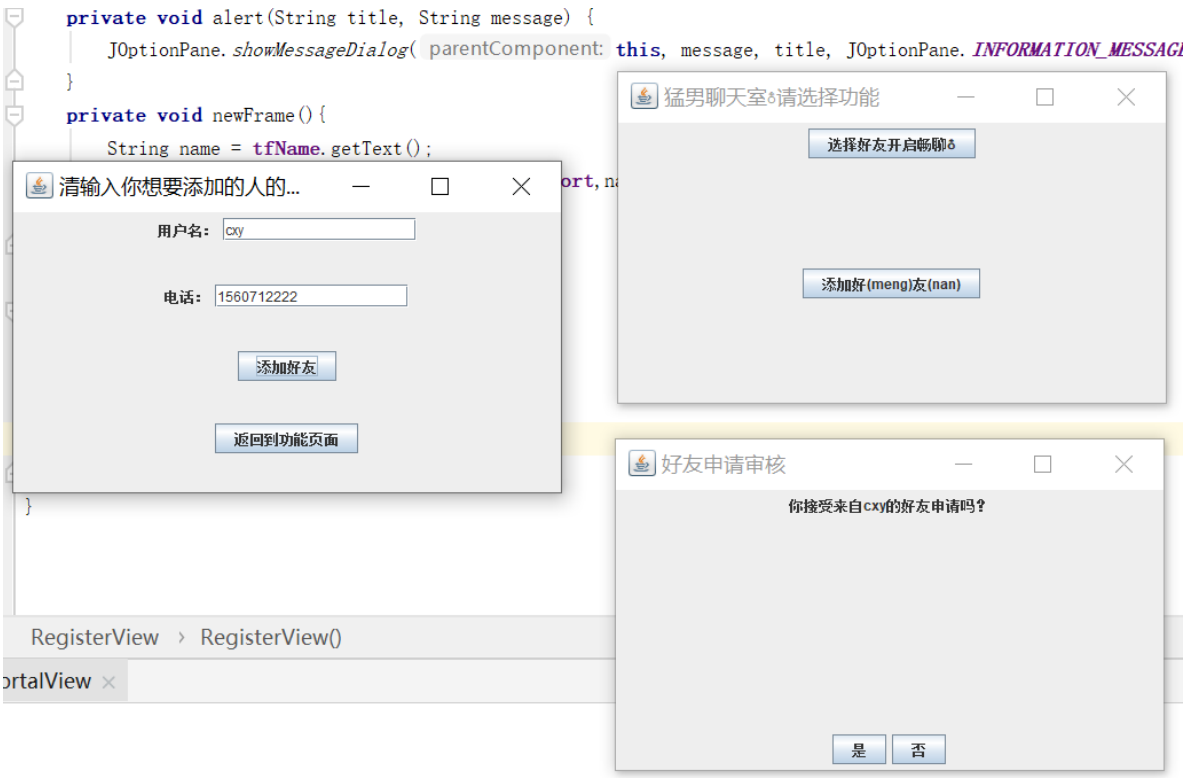
另外，为了之后发消息时确定收发方，将name也作为参数传进新窗口，保证后面的功能View都知道自己的name是什么。

4.功能页面

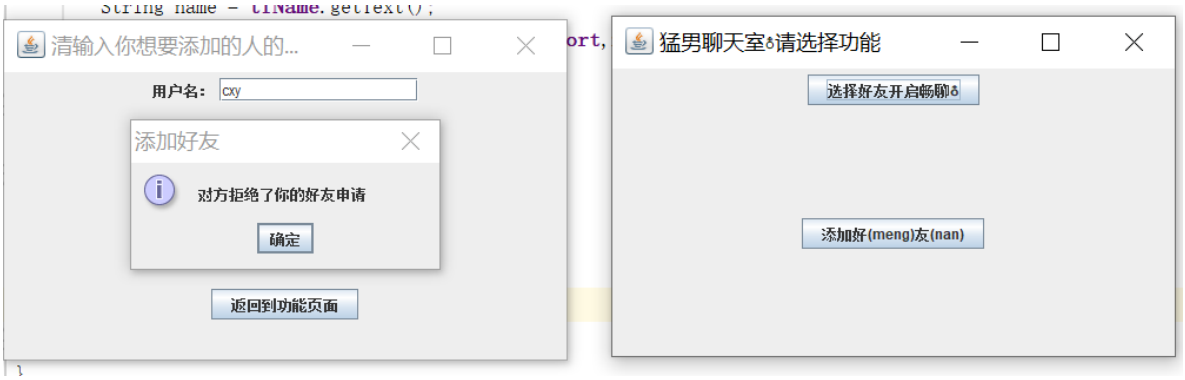
功能有：聊天、加好友。就是一个中转站，为了简便才这样做。

5.加好友

客户1输入客户2的name和phone，发送加好友申请，经过服务端到客户2。客户2可以同意或不同意好友申请，经过服务端将验证消息反馈到客户1。



如果cxy拒绝gnc的好友申请。



如图，则实现了gnc加cxy好友，双方的frends都更新了。

实现

以一个添加好友的流程的形式说明实现。

客户1：点击"添加好友"，发送加好友请求。监听事件调用networkService.addFriends()方法

```
if(e.getSource() == btnAdd){
    String name = tfName.getText();
    String phone = tfPhone.getText();
    networkService.addFriends(name,phone,myName);
}
```

networkService.addFriends()方法：省略非关键步骤

```
public void addFriends(String name,String phone,String from_name){
    //判断该用户是否存在，感觉在这里开数据库有安全问题，不过我也不明白
    if(flag.equals("false")){
        callback.onNotExist();  //用于给客户1返回”查找人不存在“的消息
        return;
    }
    // 将消息写入套接字的输出流
    try {
    outputStream = new DataOutputStream(socket.getOutputStream());
    outputStream.writeUTF("c" + name + "#" + phone +"#"+ from_name);
    outputStream.flush();
    } catch (IOException e) {
        e.printStackTrace();
        }
    }
```

服务端的ChatSocket获得输入流，实现callback接口的onReadSocket()方法，方法的实现在ClientManager中。

调用sendAll向其他客户发送好友请求消息。

```
//向除了本socket以外的所有socket发送该条消息，也就是向其他客户发消息
public void sendAll(ChatSocket chatSocket, String msg) {
    synchronized (chatSockets) {
    for (ChatSocket cs : chatSockets) {
        if (!cs.equals(chatSocket)) {
            cs.send(msg);
            }
        }
    }
}
```

客户2在NetworkService收到这条消息，实现callback接口的onRequestChecked方法对消息进行处理。

注意此时所有其他用户也都收到了这条消息，但要它们不做出反应，也就是onRequestChecked()方法要判断该用户是否是这条消息的接收方。在FunctionView中实现onRequestChecked()方法。

这几点上做了简化处理：用户1发送好友申请后不能关闭页面，要等到用户2同意/不同意之后才能关闭；用户1发送好友申请时，用户2要在线才能收到。

安全性上也有几点顾虑：输入name和phone判断该用户是否存在的方法写在客户端，也就是客户端打开数据库查询，我怀疑这样是否安全【选择好友聊天那里甚至在View中开了数据库】；sendAll()方法我也怀疑是否安全，因为消息毕竟传到了别的客户那里去。

onRequestChecked方法：

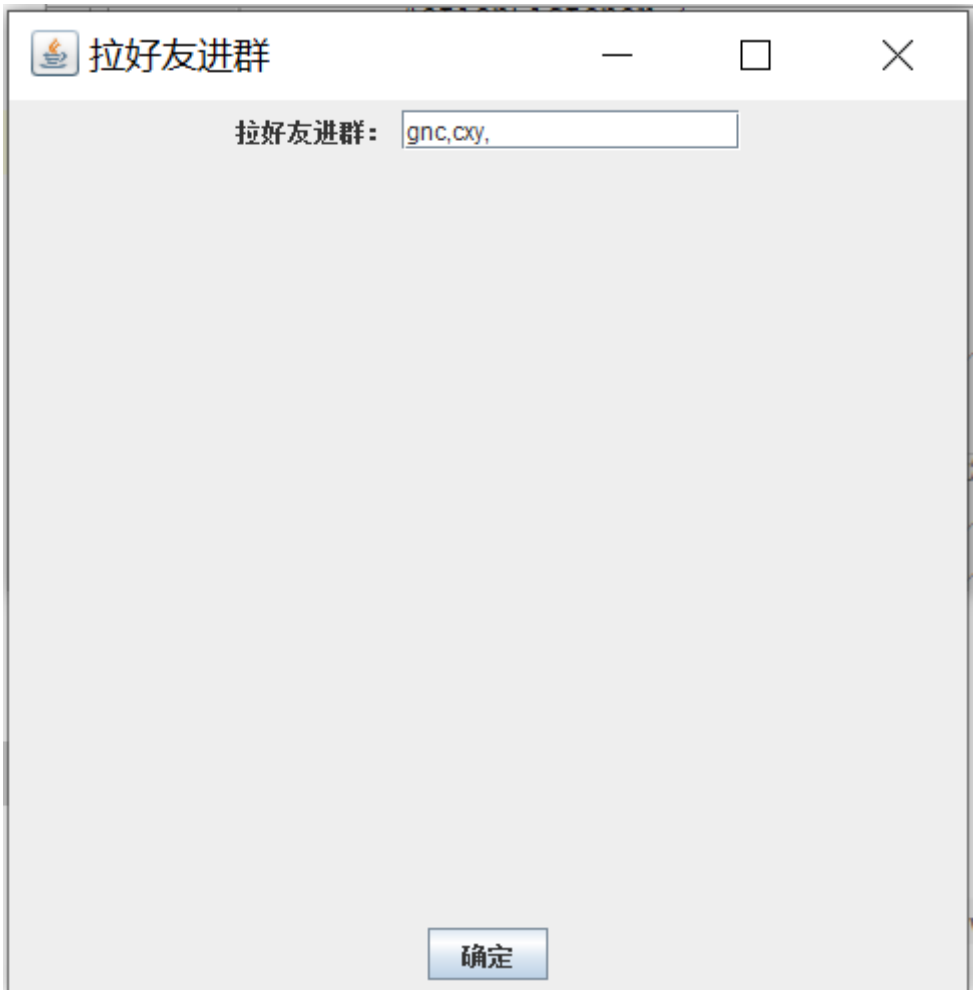
```
public void onRequestChecked(String msg){
    String []s = msg.split("#");
    if(s[0].equals(myName)) {   //判断自己是否是接收方
        CheckView view = new CheckView(host,port,msg);  //打开”是否通过好友请求“窗口
    }
}
```

以上就完成了**好友请求信息**的客户1->服务器->客户2过程。客户2在CheckView还要选择”是“或者”否“来同意或拒绝好友请求。该**验证消息**也要经过客户2->服务器->客户1的过程。

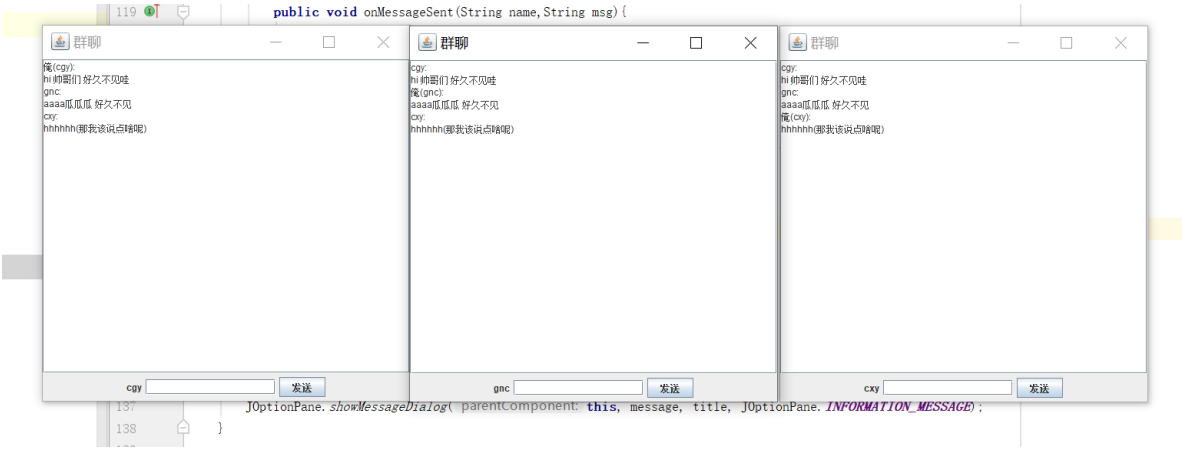
和前面是相似的就不再赘述，要注意的是**sendAll()**方法有一些不同。第二个过程的sendAll是发给所有socket，包括自己，否则会出错，错误是客户1的socket接收不到返回的验证消息。

6.群聊

客户1发起群聊，选择群聊好友（以逗号隔开，逗号结尾）



点击确定后，自己和被拉群的人就会弹出群聊窗口，然后就可以开始群聊了。



实现

群聊的实现原理和上面的加好友差不多。区别是View的设计。

为了显示消息，群聊view使用了JTextArea。并且自己发出消息跟收到消息的实现不同，注意前者没有经过服务端，直接在networkservice调用接口实现。具体代码如下：

```
private JTextArea taChatList;   // 聊天内容区
private JScrollPane scrollPane;

private void initView(){
    taChatList = new JTextArea(20, 20);
    taChatList.setEditable(false);
    scrollPane = new JScrollPane(taChatList);
}

private void initNetworkService(){
    //发出群聊消息
    @Override
    public void onMessageSent(String name,String msg){
        tfMessage.setText("");  //清空发送消息的区域
        taChatList.append("俺(" + name + "):\r\n" + msg + "\r\n");    //在Area显示
消息的方法
    }
    //收到群聊消息
    @Override
    public void onGroupReceived(String from_name, String msg,String group){
         String []s = group.split(",");
         for(int i=0;i<s.length;i++){
            if(s[i].equals(name)){  //确定自己是群聊消息的接收者 name是自己的name
                taChatList.append(from_name + ":\r\n" + msg + "\r\n");
                break;
             }
         }
    }
}
```

----------注意：**以下**才实现实验要求中的**xiv：发给指定用户的，不能发给其他结点**和**xiii：不能经过服务器**----------

7.离线聊天记录

cgy给gnc发了两条消息，因gnc不在线，消息存在message表中。

待gnc登录，则收到离线聊天记录如图：

实现

注意做了以下简化：所有发给该用户的离线消息均展示在一个窗口，该窗口仅作展示用不能聊天，不对图片消息做处理。

P2P发送消息时，通过onlineCheck函数判断接收者是否在线，若不在线，则将消息插入message表中。

```
//通过在process表中查找name判断接收者是否在线，即是否存在name为该name的进程
private String onlineCheck(String name){
    String isOnline = "false";
    try {
        DBconn.init();
        ResultSet rs = DBconn.selectSql("select * from process where name='" +
name + "'");
        while (rs.next()) {
            if (rs.getString("name").equals(name)) {
                isOnline = "true";
            }
        }
        DBconn.closeConn();
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return isOnline;
}
//离线消息插入message表
DBconn.init();
DBconn.addUpdDel("insert into message(from_name,to_name,content) " +
    "values('" + from_name + "','" + to_name + "','" + msg  + " ')");
DBconn.closeConn();
```

对于接收者，登录/注册成功的回调接口则要补充实现，显示所有发给自己的离线消息。

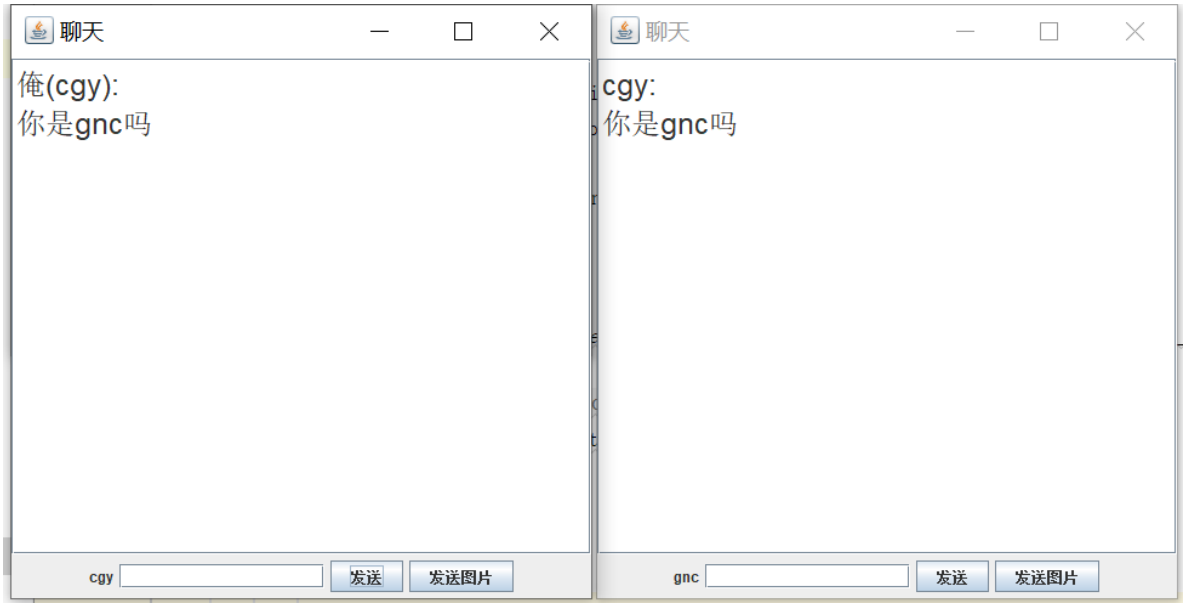
另外要注意显示完之后要删去已读的离线消息。

```
public void readOfflineMessage(String name,String host,int port){
    ArrayList<String> list = new ArrayList<>();
    try {
        DBconn.init();
        ResultSet rs = DBconn.selectSql("select * from message where to_name='"
+ name + "'");
```

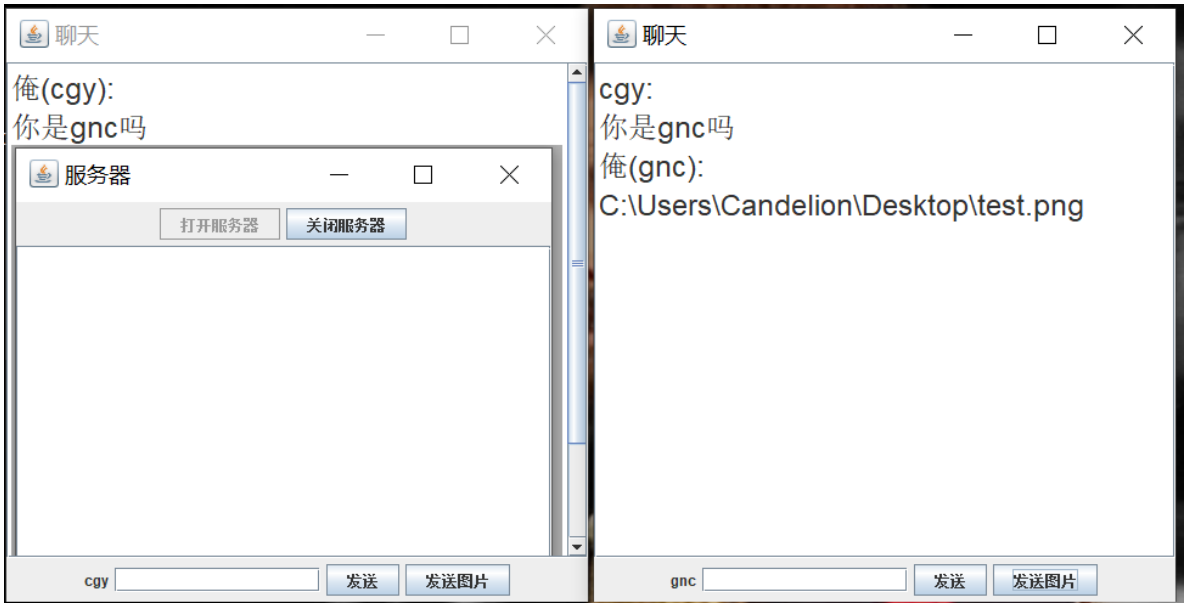
```
        while (rs.next()) {
            if (rs.getString("to_name").equals(name) ) {
                //为了简化实现，离线消息不处理图片
                list.add(rs.getString("from_name")+" send the message:  "+
rs.getString("content"));
                }
            }
            DBconn.closeConn();
        } catch (SQLException e) {
            e.printStackTrace();
    }
    //如果有记录，就弹出一个窗口显示所有离线消息
    if(list.size()>0){
        OfflineView view = new OfflineView(host,port,list);
    }
    //已经收到离线消息，则该删去该to_name的message
    DBconn.init();
    DBconn.addUpdDel("delete from message where name='"+ name +"'");
    DBconn.closeConn();
}
```

8.P2P在线聊天

客户1点击“发起聊天”按钮，选择客户2进行聊天。客户2弹出聊天框。客户1发出消息，如图：



也可以发送图片，要求在输入框输入图片的路径：



实现

客户1在确认好友发起聊天后，自己弹出聊天框，同时调用P2Prequest()方法。

P2Prequest()方法会判断对方是否在线，如果在线，则会通过P2P方式让对方弹出聊天框，也就是用DatagramSocket.send()。注意在构建数据报时，要查找数据库中的process表，得到接收方客户端的host和port。查找的代码如下：

```
DBconn.init();
ResultSet rs = DBconn.selectSql("select * from process where name='" + to_name + "'");
while (rs.next()) {
    if (rs.getString("name").equals(to_name)) {
        to_port = rs.getInt("port");
    }
}
DBconn.closeConn();
```

客户2通过beginListeningUDP()线程接收到这个请求消息，则会弹出聊天框。

注意这里跟前面做了不同的处理，因为该线程无法调用callback，所以将chatView作为全局静态变量，初始值为null，有两种给它赋值的情况：一种是现在的情况，因收到别人的发起聊天请求而需弹出聊天框，给chatView赋值；另一种是发起request时，也要将自己在FriendsView中new好的chatView作为参数传递进来，并在P2Prequest()方法中给networkservice的全局静态变量chatView赋值。代码如下：

```
static private ChatView chatView = null;

//情况1
private void beginListeningUDP() {
    ...收到消息
    if(msg.charAt(0) == 'h') {
        chatView= new ChatView(s[2],Integer.parseInt(s[3]),s[1],s[0]);
    }
}

//情况2
public void P2Prequest(...ChatView chatView) throws IOException {
    this.chatView = chatView;
}
```

弹出来的聊天框ChatView，注意它与之前的群聊GroupView也有所不同。为了能显示图片，采用**JTextPane**。

```
public JTextPane taChatList = new JTextPane();
taChatList.setEditable(false);
private JScrollPane scrollPane = new JScrollPane(taChatList);
add(scrollPane, BorderLayout.CENTER);

//插入文字
SimpleAttributeSet attrset = new SimpleAttributeSet();  //设置字体大小
StyleConstants.setFontSize(attrset,24);
Document docs = taChatList.getDocument();               //获得文本对象
try {
    docs.insertString(docs.getLength(), "msg", attrset);//对文本进权行追加
    } catch (BadLocationException e) {
    e.printStackTrace();
}
//插入图片
taChatList.insertIcon(new ImageIcon(路径));
```

发送消息调用sendP2P()方法：

```
DatagramPacket packet = new DatagramPacket(msg2.getBytes(),
msg2.getBytes().length, InetAddress.getByName("127.0.0.1") ,to_port);
UDPsocket.send(packet);
callback.onMessageSent(from_name,msg);
```

接收消息，需判断是文字消息还是图片消息：

```
UDPsocket.receive(packet);
String msg = new String(buffer,0,len);
if(msg.charAt(0) == 'i') ...
if(msg.charAt(0) == 'j') ...
```

遇到问题及解决

1.关于加好友，我一开始想在服务端根据name和phone找到客户2，但发现做不到，服务端只能管理多个socket的加入连接，并不能识别出这个socket对应的是哪个客户。因此选择了将[判断自己是否是接收者]推延到客户端去解决。

2.如何在从门户页面进入其他页面后依然保持跟服务端的连接。我现在用了传参的方法，但我觉得这种每new一个新窗口就connect一次的方法，会得到新的socket。如何做到一个客户进程只用一个socket？我还没想到方法。

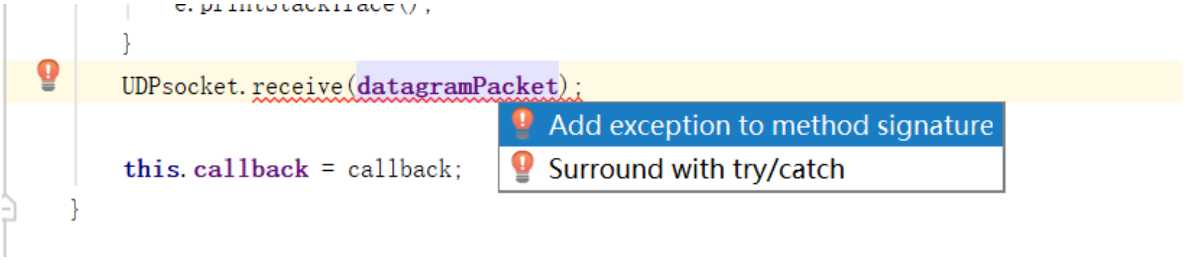
3.对于像"弹出好友申请""别人发起聊天"这类的别人操作导致自己出个窗口的情况，有两种实现方法。一种是在networkservice里接收数据之后，调用接口，然后在functionview里实现该接口的对应方法。一种是直接在networkservice里new个View，但注意要传递host、port等参数。

4.一些粗心的细节问题：可能alt+enter导入的库错了。不小心手滑点到什么奇怪的String类型，把上面的import删掉就好。Java里String类型不能用==判相等，要用equals。在View中，并不是接口实现不能用this.dispose()，而是这里的this指的是callback，在接口实现方法外用this.dispose然后调用这个方法就行。因为用了第一个字节作为消息类型，msg传来传去容易忘了有没有substring(1)过。使用JDBC操

作数据库时，得到Redultset之后要先执行结果集的rs.next()方法，再取值。 x##这种情况没办法split("#")，会出错。

5.我之前很疑惑，serversocket只有一个，但所有socket都与它建立连接，这样可以吗？根据博客https://blog.csdn.net/u011580175/article/details/80306414，服务端端口不变，但客户端端口是不同的；一个服务端可以连接多个客户端，但同一时间只有一个监听线程。

6.报UnknownHostException或者java.io.IOException错，是因为没有捕获异常。可以alt+enter，第一种是throw并为整个类添加异常处理，第二种是为这几行代码添加try catch。



7.虽然process表中显示host是0.0.0.0，但send时如果InetAddress.getByName("0.0.0.0")会报错java.net.BindException: Cannot assign requested address。可以改成"127.0.0.1"发送到本地，因为本来也是绑定本地。

不足且需要改进之处（除了上文提到过的以外）：

1.msg是用户名和密码用"#"隔开，所以消息里别的地方不能出现"#"。

2.注册时服务器审批可以在服务端弹出一个窗口，由服务端选择是否通过注册。为了简化省略了这一步，仅仅是在服务端进行了基本检查。

3.每次view中要实现callback所有方法，但有些用不上所以就是空方法，有没有办法只实现自己要用的。

4.一开始数据库设置phone意义应该是通过name或者phone都能查找到用户，后面忘了这样写。如使用name+phone来确定收发人，也可以提高准确度，但因为也没这么写。

5.没有测试消息冲突情况，比如被加好友时又有人发来聊天消息。

6.关于接口实现中的this.dispose()，一开始因为理解不清楚导致写的很复杂。简洁的写法是，在接口实现中new新的Frame，然后调用接口实现外的写有dispose的方法就行，只需要写这一句。

7.离线聊天记录，是新用户登录/注册后，会弹出所有其他用户给他发的消息，在一个窗口。现实情况是要根据发送者的不同，分别弹出不同的聊天窗口。

8.图片仅仅是发送了一个目录在窗口中显示出来，而非整个文件。

9.现在还不明白为什么第二个线程就不能用callback了。

走过的弯路，想用却没用上的东西：

1.采用mongodb，表对应collection，字段等内容是data（json文件），索引是index。后来发data是json格式，不太懂所以放弃。

2.用maven配置MVCSpring项目。遇到的问题有：

配置tomcat时：找不到应用程序。修改default为chrome。

导入依赖时全部爆红：原因很多，看log。试过了换版本，配置镜像。还要清缓存重启几次才能好，很玄学。

后来还是不懂这个框架，所以放弃，直接用了最小白的项目。

3.JavaSpring写前端的JavaWeb项目。觉得太麻烦了所以放弃，用了最简单的Java自带GUI。

总结：

先在网上查资料，看到一篇博客：https://www.jianshu.com/p/b4e2a555ef4c。这篇博客只实现了简单的C/S的demo，功能只有对所有用户发消息，但多线程socket的方法值得参考。于是在这篇博客的基础上，我接着实现了登录、注册、加好友功能。其中登录和注册只涉及到单线程，加好友涉及到多线程。随着功能的实现自己的理解也在加深。

接下来实现聊天功能。由于实验要求里有“发给指定用户，不经过其他节点”和“P2P”和离线聊天记录，所以之前的sendAll发消息做法行不通。继续查资料找合适的做法。刚好最后一节课有同学讲了P2P，我意识到自己之前sendAll的方法可能就类似于洪泛法。而它最后讲的分布式散列表则给我提供了思路，可以建立一张**process表**，将ip地址、端口和资源（这里可以是name）对应起来，这样想给某个客户发消息时，可以先查表得到该客户ip地址和端口再P2P发送。这种实现方法是**客户1->服务器告知客户2端口和IP**的简化。

“如何找到客户”解决了，那么下一个要解决的就是“如何实现两个客户之间的直接通信”。看https://blog.csdn.net/qq_40866897/article/details/82958395?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-3&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-3里讲的：“客户端与服务器通信时，传递注册名和地址要保证传送的准确性和可靠，故选择TCP连接客户端和服务器，使用Socket对象；客户端与客户端之间进行通信时，要求实时性，不需要无比的准确，故选择UDP连接客户端与客户端，使用DatagramSocket对象进行通信，DatagramPacket为数据包。UDP为无连接传输，在DatagramSocket传送DatagramPacket时，只需要知道IP地址和端口号即可进行通信。”

接下来通过和同学讨论，想到了两个线程的写法。以及显示图片的JTextPane。

写到最后感觉我写的逻辑的确不清楚，还有待提高。