一、实验目的

本次实验需要完成一个实时聊天系统。该系统由聊天客户端和聊天服务器组成。

客户端即可以给指定客户端发送信息,也可以把消息广播给所有客户端。

通过这个实验,使得学生掌握 socket API 编程。

三、实验内容:

该章节将会详细的列出服务器和客户端的功能要求。

1.服务器的功能

（1）服务器能够并行处理客户端发送过来的消息或者命令

（2)服务器把从新连接的客户端中收到的第一条消息作为该客户端的用户名称

（3）服务器能够处理从客户端发送过来的两种类型的信息:(1)广播信息,服　　务器收到信息后将其广播给其他客户端(2)点对点信息,服务器将消息转　　发给指定客户端

（4）服务器需要把某个客户端发送的广播聊天信息,广播给和该客户端在同　　一个频道的所有客户端(但是不包括该用户端自己)。

这个广播消息的格 式应为[<name>]:<message>。其中<name>为发送聊天信息的客户端名 称,<message>为该客户端发送的消息。

（5）当某个客户端离线时,服务器需要广播一个离线消息给所有的客户端。

（6）容错处理。当客户端发送的数据和规定的格式不一致时,服务器能够处理这个错误并且返回错误信息。

2.客户端的功能:

（1）从服务器返回的消息需要显示在控制台下,并且需要去除末尾多余的空格。

（2）为了区别于其他人发送的消息,客户端自己发送的消息,需要在控制台中以“[me]:”显示。

四、操作环境

1.操作系统：Mac OS

2.编写语言：Java

3.编译软件: Eclipse

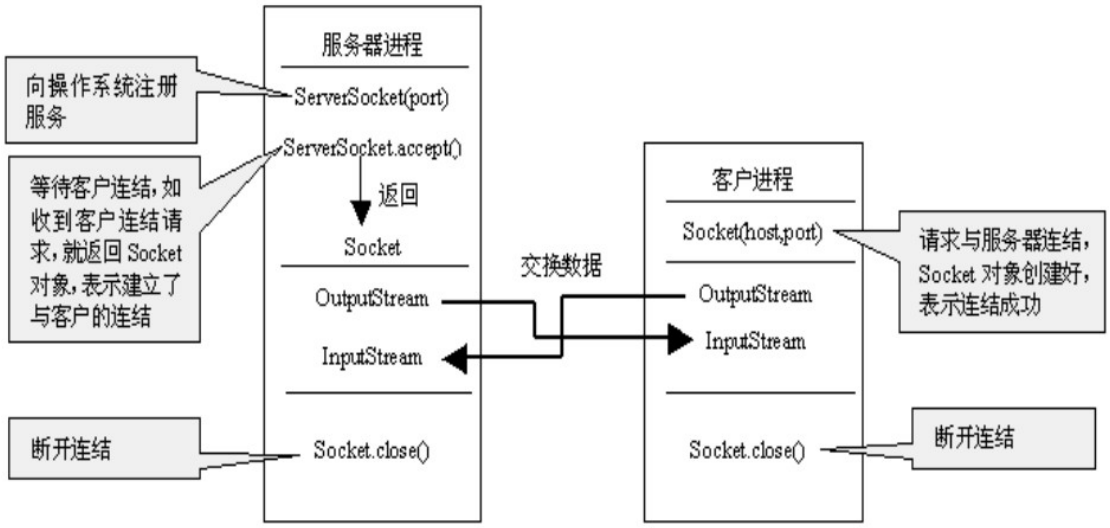
五、问题分析

网络编程中两个主要的问题一个是如何准确的定位网络上一台或多台主机，另一个就是找到主机后如何可靠高效的进行数据传输。在 TCP/IP 协议中 IP 层主要负责网络主机的定位，数据传输的路由，由 IP 地址可以唯一地确定 Internet 上的一台主机。而 TCP 层则提供面向应用的可靠（TCP）的或非可靠（UDP）的数据传输机制，这是网络编程的主要对象，一般不需要关心 IP 层是如何处理数据的。目前较为流行的网络编程模型是客户机/服务器（C/S）结构。即通信双方一方作为服务器等待客户提出请求并予以响应。客户则在需要服务时向服务器提出申请。服务器一般作为守护进程始终运行，监听网络端口，一旦有客户请求，就会启动一个服务进程来响应该客户，同时自己继续监听服务端口，使后来的客户也 能及时得到服务。

两类传输协议：TCP；UDP。TCP 是 Tranfer Control Protocol 的 简称，是一种面向连接的保证可靠传输的协议。通过 TCP 协议传输，得到的是一个顺序的无差错的数据流。发送方和接收方的成对的两个 socket 之间必须建立连接，以便在 TCP 协议的基础上进行通信，当一个 socket（通常都是 server socket）等待建立连接时，另一个 socket 可以要求进行连接，一旦这两个 socket 连接起来，它们就可以进行双向数据传输，双方都可以进行发送或接收操作。

UDP 是 User Datagram Protocol 的简称，是一种无连接的协议，每个数据报都是一个独立的信息，包括完整的源地址或目的地址，它在网络上以任何可能的路径传往目的地，因此能否到达目的地，到达目的地的时间以及内容的正确性都是不能被保证的。

经过慎重的选择，本程序基于 TCP 传输协议，本人独立完成了整体设计，并部分借鉴了部分教材上使用的架构设计，逐项完成了实验要求的功能。



六、详细设计

1.总体设计思路

Part1:单连接、单线程

如果仅仅是客户端与服务器进行通信，则设计较为简单，仅需要在服务器开辟端口建立 ServerSocket 即可，用死等函数.accept()等待客户端 Socket 连接，一旦连接成功则在服务器建立 Socket 对象匹配客户端。具体描述如下：

（1）用指定的端口实例化一个 SeverSocket 对象。服务器就可以用这个端口监听从客户端发来的连接请求。

（2）调用 ServerSocket 的 accept()方法，以在等待连接期间造成阻塞，

监听连接从端口上发来的连接请求。

（3）利用 accept 方法返回的客户端的 Socket 对象，进行读写 IO 的操作

（4）关闭打开的流和 Socket 对象

其中，IO 操作可以利用 Socket 提供的 getInputStream 和 getOutputStream 方法得到输入和输出流，为方便起见，可以采用 BufferedReader/PrintStream 封装。

```
try{
ServerSocket ss = new ServerSocket(9999);
Socket s1  = ss.accept();
jta1.append("连接成功\n");

bf1=new BufferedReader(new InputStreamReader(s1.getInputStream()));
ps1=new PrintStream(s1.getOutputStream());
new Thread(this).start();
}catch (Exception ex) {}

}
```

客户端较为简单，只需要匹配到服务器的 IP 地址和相应的 Port 即可。具体描述如下：

（1）用服务器的 IP 地址和端口号实例化 Socket 对象。

（2）调用 connect 方法，连接到服务器上。

（3）获得 Socket 上的流，把流封装进 BufferedReader/PrintStream 的实例，以进行读写

（4）利用 Socket 提供的 getInputStream 和 getOutputStream 方法，通过 IO 流对象，向服务器发送数据流

关闭打开的流和 Socket。

```
    //客户端链接
    try{
    Socket s = new Socket("127.0.0.1",9999);
    bf = new BufferedReader(new InputStreamReader(s.getInputStream()));
    ps = new PrintStream(s.getOutputStream());
    new Thread(this).start();
    }catch(Exception ex){}
}
```

Part2:单连接、多线程

上述方法一个很致命缺点，这个连接没有开辟多线程，也就是说，服务器必须等待客户端发送来消息后才可以给一个回执消息，客户端操作过程也类似，但我们希望的应该是一个全双工的通信，因此必须开辟多线程。

开辟多线程的方法很简单，只需要继承 Runnable 接口即可，然后将 IO 操作添加至 run 函数中。

```
implements Runnable,
```

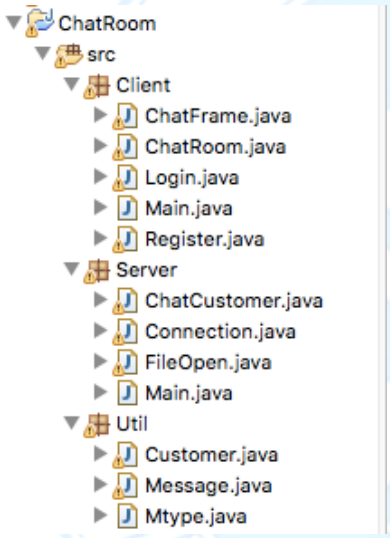
```
public void run()
{  try{
   while(true){

      String str = bf.readLine();
      jta.append(str + "\n");
}
   }catch (Exception ex) {}
            }
```

Part3:多连接、多线程

现在基本的通讯问题已经解决了，服务器和客户端可以像发短信一样互相全双工的发送信息了，但这还远远不够，因为我们不能是针对两个用户的应用，我们的应用应该可以在服务器转发来自不同用户的消息，并且实现私聊和群聊的区别，因此，我们需要做一个概要设计，之后会详细讲述每部分的内容。

如图，大体的设计架构如下，总共分为三个包：



（1）Client 包：与客户端相关的一切信息，其中 Login 类实现客户端到服务器的登陆，Register 类实现新用户信息的注册，ChatRoom 为聊天主界面，ChatFram 为私聊窗口。

（2）Server 包：完成服务器相关功能实现，FileOpen 用于识别登录用户的各项信息，由于时间紧迫，暂未连接数据库，采用的是文本文件存储用户内容，Connection 类用于开辟端口等待连接，同样采用多线程技术为每个链接进来的用户开辟线程，并保存所有已连接用户的信息，并保存他们的进程，ChatCustomer 类用于保存连接用户的全部

信息，并完成登陆、注册消息的处理以及聊天内容、文件等转发操作。

（3）Util 包：为服务器客户端通用内容，Customer 保留了每个登陆用户的信息，Message 定义了消息的类别、内容、发送方、接收方以及其他信息，Mtype 类定义各种用到的常量。

2.Util 包具体内容

（1）Mtype 类

定义各种常量，主要分两类：消息类别和 IP / Port

```
public class Mtype {
    public static final String LOGIN = "LOGIN";  //登陆
    public static final String REGISTER = "REGISTER";  //注册
    public static final String LOGINFALL = "LOGINFALL";  //登陆失败
    public static final String USERLIST = "USERLIST";    //用户列表
    public static final String REGISTERSUCCESS = "REGISTERSUCCESS";    //注册成功
    public static final String REGISTERFALL = "REGISTERFALL";     //注册失败
    public static final String MESSAGE = "MESSAGE";      //聊天消息
    public static final String LOGOUT = "LOGOUT";     //下线
    public static final String ALL = "ALL";     //所有人
    public static final String Error = "Error";      //命令出错
    public static final String FILE = "FILE";  //文件请求


    public static final String IP = "localhost";  //IP
    public static final int PORT = 6666;  //Port
}
```

（2）Message 类：

```
public class Message implements Serializable{
    //消息类型
    private String type;
    //消息内容
    private Object content;
        //发送方
    private String from;
        //接收方
    private String to;

    private String others;

    private int size;
    private String file;
```

如图，主要定义了消息类型、消息内容、发送方、接收方以及其他信息（主要用于文件传送），类中用 set 函数和 get 函数为消息赋值和得到消息内容。

（3）Customer 类：

```
public class Customer implements Serializable{
    private String account;
    private String password;
    private String name;

    public void setAccount(String account) {
        this.account = account;
    }
    public void setPwd(String password) {
        this.password = password;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getAccount() {
        return(this.account);

    }

    public String getPwd() {
        return(this.password);

    }

    public String getName() {
        return(this.name);
    }
}
```

同 Message 类，主要为得到已连接用户的各项基本信息。

3.Server 包

（1）Connection 类

由基本的 ServerSocket 功能扩展而来，主要为并行处理客户端发送过来的消息或者命令，并保存各个连接的客户线程。

```
public class Connection extends JFrame implements Runnable,Serializable{

   private ServerSocket serverSocket  = null;
   private Socket socket = null;
   private Vector<ChatCustomer> users = new Vector<ChatCustomer>();
   private Vector<Customer> online_users = new Vector<Customer>();

   public Connection() {

      this.setVisible(true);
      this.setTitle("当前在线"+online_users.size()+"人");
      this.setDefaultCloseOperation(EXIT_ON_CLOSE);
      this.setSize(200,200);

      try{
         System.out.println("no");
         serverSocket = new ServerSocket(Mtype.PORT);
         System.out.println("yes");
      }catch (Exception e) {
         JOptionPane.showMessageDialog(null, "服务器链接出错");
         e.printStackTrace();
      }


      new Thread(this).start();


   }
```

Vector 可以更好的处理并发功能，在客户端窗口顶端显示在线的人数

```
   public void run() {

      while (true)
      {
         try {
            socket = serverSocket.accept();
            ChatCustomer chatcus = new ChatCustomer(socket,this);

         } catch (Exception e) {
            JOptionPane.showMessageDialog(null, "服务器链接出错");
            try{
               serverSocket.close();
            }catch (Exception ex) {
               // TODO: handle exception
            }


         }
      }

   }


   public Vector<ChatCustomer> getUsers() {
      return users;
   }

   public Vector<Customer> getOnlineUsers() {
      return online_users;
   }

}
```

采用多线程，并行处理多个连接。并可以返回在线用户列表。

（2）ChartCustomer 类

该类主要是处理每个连接到服务器的客户端线程。
根据发送来的消息，可以分为：登陆消息、离线消息、注册消息、普通聊天消息、文件传送消息，以及应对错误错误消息格式的反馈。在此，如果一个线程断开，认为该用户已离线，发出离线消息给所有用户。

```
public void run() {
  while (flag) {

    try {
      Message message = (Message) ois.readObject();

    String type = message.getType();
    if (type.equals(Mtype.LOGIN)) {

      Login(message);

    } else if (type.equals(Mtype.REGISTER)) {

      Register(message);

    } else if (type.equals(Mtype.MESSAGE)||type.equals(Mtype.FILE)) {

      sendMessage(message, message.getTo());

    }
    else{
      ErrorCommand(message);
    }
    }
    catch (Exception e) {
      if(this.customer!=null)
      Logout();
    }
  }
```

对于每个消息，服务器都会解析消息的具体内容，对于登录的用户，需要判断该账号是否已经在线、用户名密码是否正确，如果成功登陆则为该用户保留线程资源的同时发送该用户上线的消息给所有在线用户；对于注册用户，注册时需要判断用户名是否已存在，存在则返回注册失败的消息，否则保留该用户的资料；处理文件消息和聊天消息在服务器端是相同的，只需要判断是发送给所有用户的还是发送给单独用户得即可，在服务器实现转发功能。

```
public void sendMessage(Message message,String to) throws Exception{
   for(ChatCustomer chatCustomer  :  server.getUsers())
   {
      if(chatCustomer.customer.getAccount().equals(this.customer.getAccount()))
         continue;
      if(chatCustomer.customer.getAccount().equals(to)||  to.equals(Mtype.ALL))
      {
         chatCustomer.oos.writeObject(message);
      }
   }
```

（3）FileOpen 类

保存用户资料

```
public class FileOpen implements Serializable{

   private static String filename = "Users.txt";
   private static Properties properties;

   static {
      properties = new Properties();
      FileReader fileReader = null;
      try {
         fileReader = new FileReader(filename);
         properties.load(fileReader);
      } catch (Exception e) {
         JOptionPane.showMessageDialog(null, "Error in File");
         System.exit(0);
      } finally {
         try {
            fileReader.close();
         } catch (Exception e2) {
            e2.printStackTrace();
         }

      }
   }
```

```
private static void listInfo() {

   PrintStream pStream = null;
   try {
      pStream = new PrintStream(filename);
      properties.list(pStream);
   } catch (Exception e) {
      JOptionPane.showMessageDialog(null, "Error in File");
      System.exit(0);

   } finally {
      try {
         pStream.close();
      } catch (Exception e2) {
         e2.printStackTrace();
      }

   }
}
```

```
//根据帐号返回登陆者的信息
public static Customer getCustomerByAccount(String Account) {
   Customer customer = null;
   String cusinfo = properties.getProperty(Account);
   if(cusinfo!=null)
   {
      customer = new Customer();
      String[] info = cusinfo.split("@#");
      customer.setAccount(Account);
      customer.setPwd(info[0]);
      customer.setName(info[1]);
   }

   return customer;

}



public static void insertCustomer(String account,String pwd,String name) {

   properties.setProperty(account, pwd+"@#"+name);
   listInfo();
}
```

4.Client 类

由于此部分代码多为界面设计，在此不一一罗列

（1）Login 类：

登陆界面，输入用户名密码登陆，登录成功会显示聊天主界面，登录失败会返回失败原因

（2）Register 类：

用户注册界面，注册失败会予以反馈

（3）ChartRoom 类

聊天主界面，可选择在线用户进行私聊或广播消息，同样可以给用户传送文件或给所有用户群发文件。

在系统消息一栏，用于显示各种系统消息，如：用户上线、下线提醒，新消息通知、文件传送提醒等。

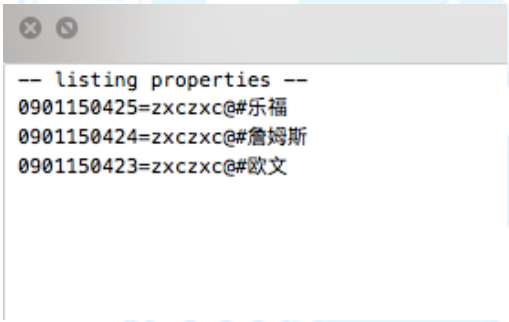
（4）ChatFrame 类

类似腾讯 QQ 的聊天界面，可以从本地同步聊天记录，并允许发送消息、接收消息。

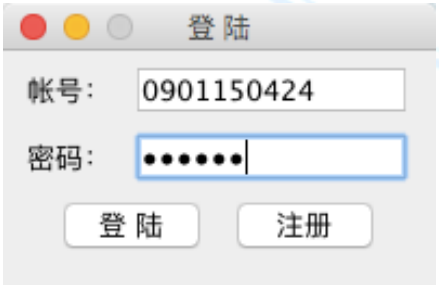
七、调试分析

注册界面如下，两次输入的密码必须一致，用户名、密码不得小于 6 位、昵称不能为空

服务器端保存的注册用户信息：



登录界面：



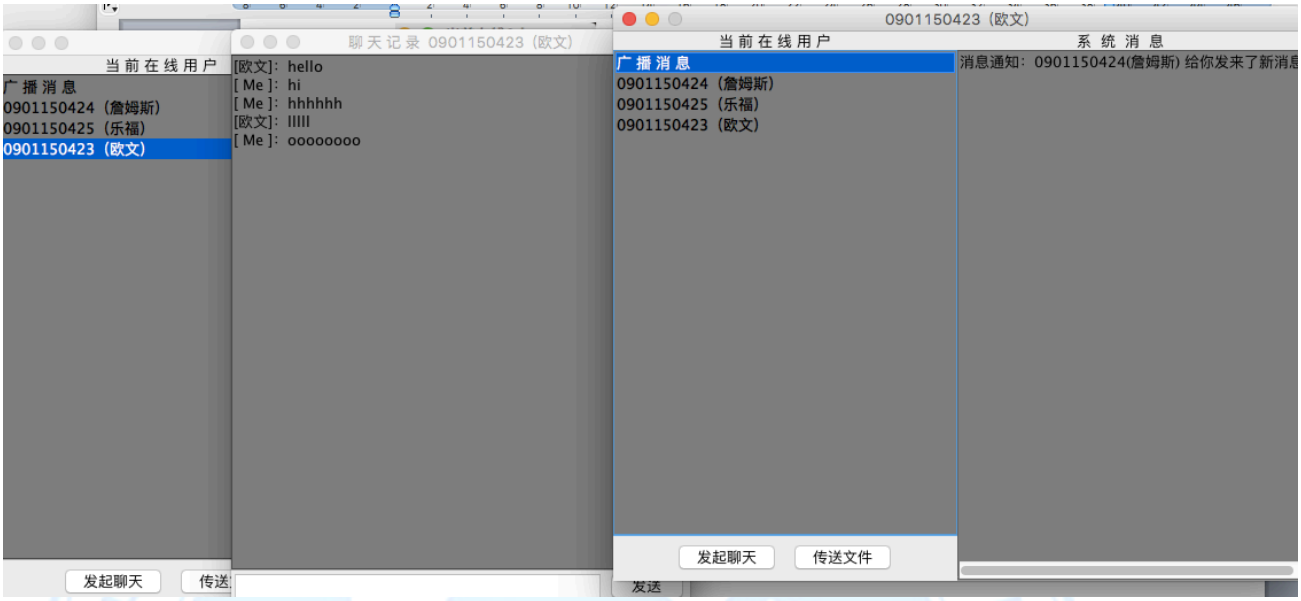
服务器会判断账号密码是否正确、是否当前用户已经在线

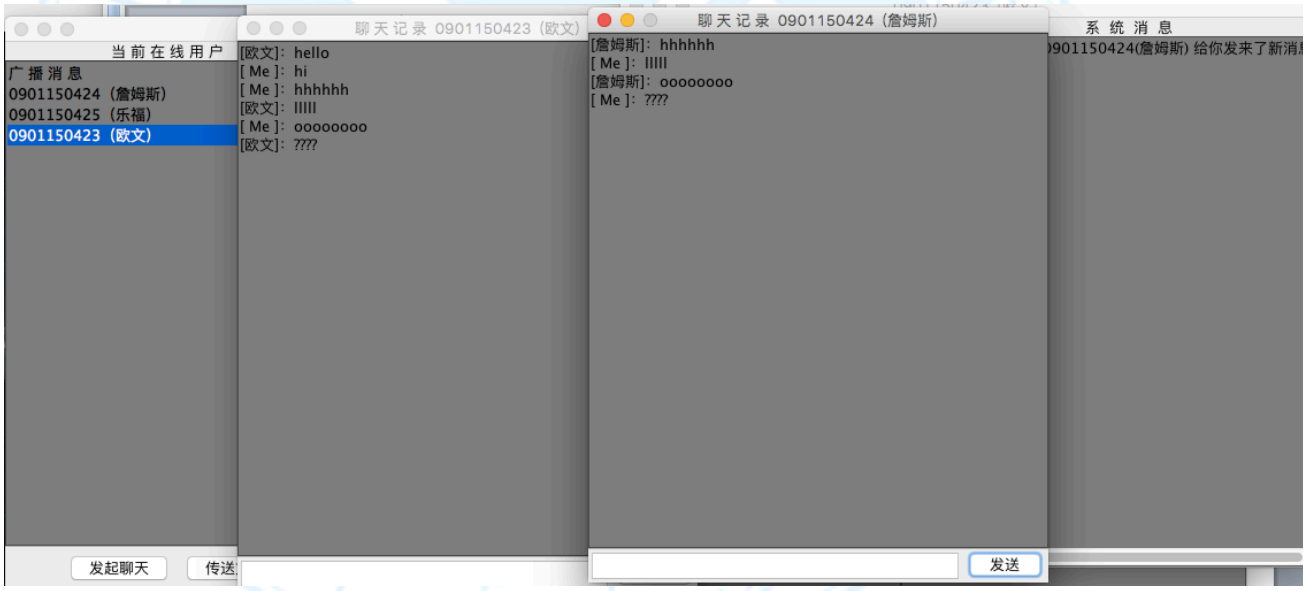
聊天室主界面：

用户上线会有提醒

客户端会显示在线人数

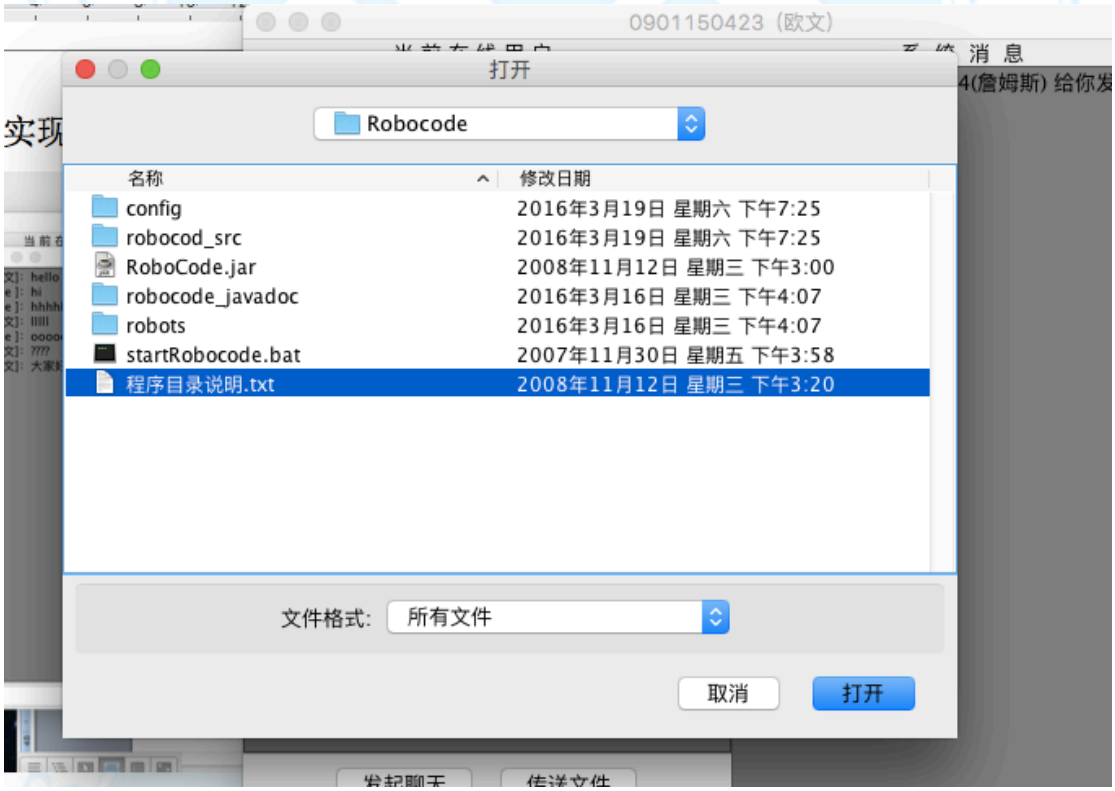
可以实现私聊功能

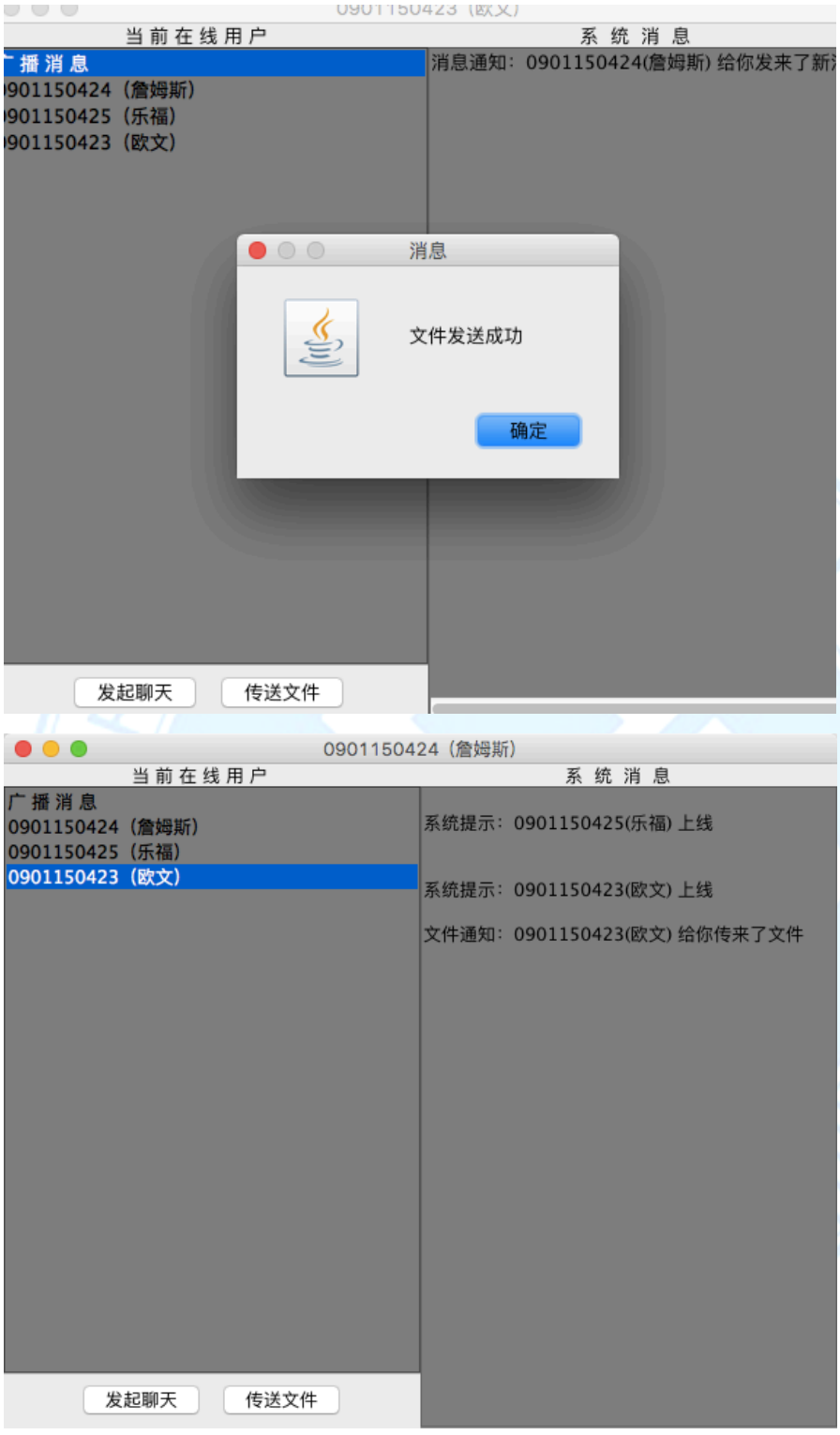

当前在线用户
广播消息
0901150424（詹姆斯）
0901150425（乐福）
0901150423（欧文）
聊天记录 0901150423（欧文）
[欧文]：hello
[Me]：hi
[Me]：hhhhhh
[欧文]：llllll
[Me]：oooooooo
0901150423（欧文）
系统消息
消息通知：0901150424(詹姆斯) 给你发来了新消息
发起聊天
传送文件
发送

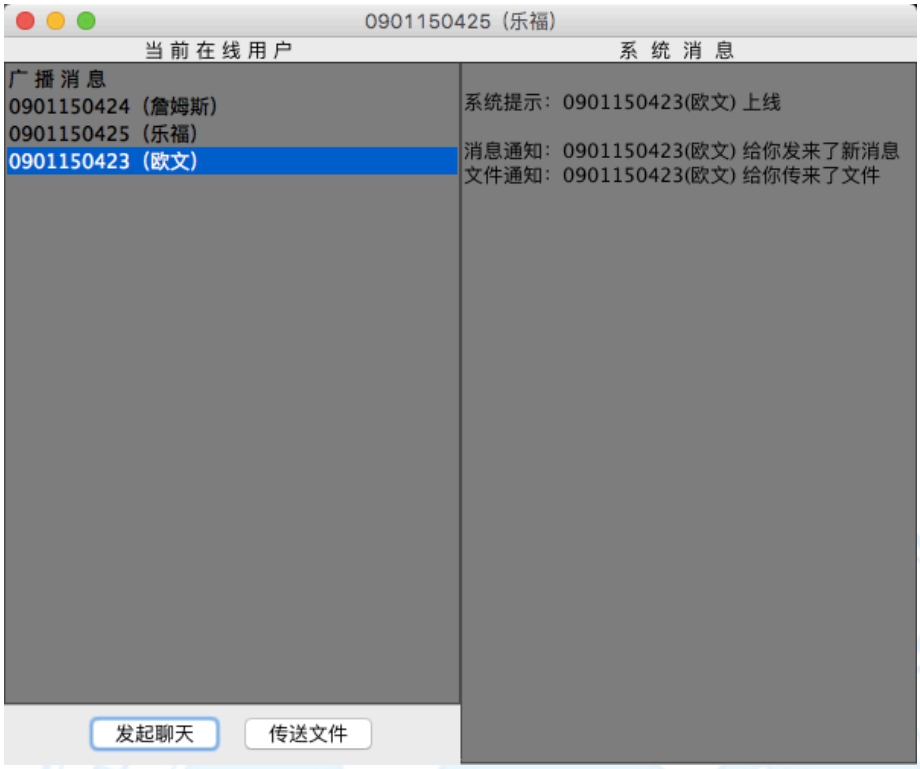

当前在线用户
广播消息
0901150424（詹姆斯）
0901150425（乐福）
0901150423（欧文）
聊天记录 0901150423（欧文）
[欧文]：hello
[Me]：hi
[Me]：hhhhhh
[欧文]：llllll
[Me]：oooooooo
[欧文]：????
聊天记录 0901150424（詹姆斯）
[詹姆斯]：hhhhhh
[Me]：llllll
[詹姆斯]：oooooooo
[Me]：????
系统消息
发起聊天
发送

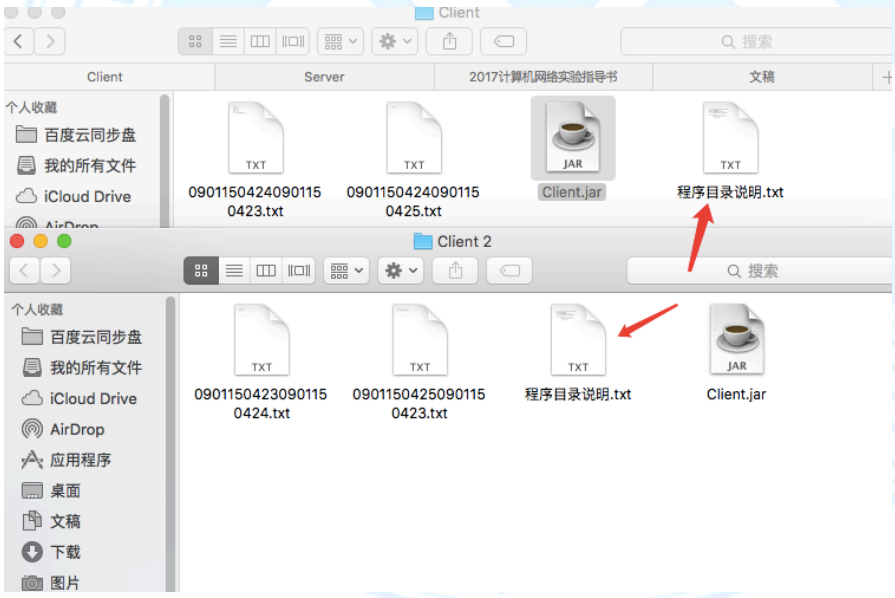
可以实现广播功能：

可以实现文件传送（以群发做示范）




当前在线用户
系统消息
播消息
901150424（詹姆斯）
901150425（乐福）
901150423（欧文）
消息通知：0901150424(詹姆斯) 给你发来了新
消息
文件发送成功
确定
发起聊天
传送文件
0901150424（詹姆斯）
当前在线用户
系统消息
广播消息
0901150424（詹姆斯）
0901150425（乐福）
0901150423（欧文）
系统提示：0901150425(乐福) 上线
系统提示：0901150423(欧文) 上线
文件通知：0901150423(欧文) 给你传来了文件
发起聊天
传送文件





可以看到所有人都收到了文件（除了本人）

离线有提醒：

聊天记录缓存在本地：