|  |  |  |
| --- | --- | --- |
| 南农大  **Java程序设计课程作业报告**  XH2 | | |
|  | 题 目: | 基于Socket实现网络聊天室 |
|  | 姓 名: |  |
|  | 学 院: | 人工智能学院 |
|  | 专 业: | 计算机科学与技术 |
|  | 班 级: |  |
|  | 学 号: |  |
|  | 指导教师: |  |
|  |  |  |
| 2020年 12月31日 | | |

# 设计思路与实现内容

**1.设计思路**

本聊天室基于图形化界面基本库java.awt及Swing组件实现了图形化界面，利用socket套接字实现了不同进程之间的通讯。利用多线程模拟实现并发的进程。同时连接mysql，实现了用户登陆与用户注册功能。

**1）图形化界面的设计**

界面类继承JFrame，并在类中定义相关组件及一些面板,在面板中设置布局，将组件加入其中，再将面板加入整个框架中，并利用对话框进行提示，便设计好了一个图形化界面。

**2）socket实现两个不同程序通过网络通信**

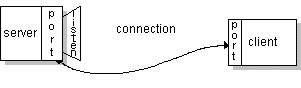
当客户端或服务器需要使用网络通信时，必须首先发出socket聊天室调用，请求操作聊天室为其创建一个socket，这个调用的实际效果是请求操作聊天室把网络通信所需要的一些聊天室资源分配给应用进程。此后，应用进程所进行的网络操作都必须使用这个套接字描述符（socket）。

TCP提供了一个可靠的，点对点的客户端-服务器应用通道，想要在TCP上进行传输，客户端程序与服务器程序需要建立彼此的连接。客户端和服务器都是对与连接绑定的socket进行读和写来传输数据。

每一个应用进程对应一个socket进行网络通信。两个进程之间相互通讯,然后其中一个必须提供一个固定的位置，而另一个则只需要知道这个固定的位置。并去建立两者之间的联系，然后完成数据的通讯就可以了。这里提供固定位置的通常称为服务器,而建立联系的通常叫做客户端。



本聊天室基于上述原理，为每个客户端分配一个套接字socket，服务器使用serversocket，与客户端建立连接。服务器serversocket的accept方法产生的socket与客户端的socket一一对应，形成可靠的连接。客户端之间通信通过socket输入输出流，交付给服务器端，在经过服务器的存储转发，发给另一个与之通信的客户端。服务端ServerSocket,创建时连带端口号，而客户端Socket,包括ip,端口号。



**3）利用多线程模拟并发通信进程**

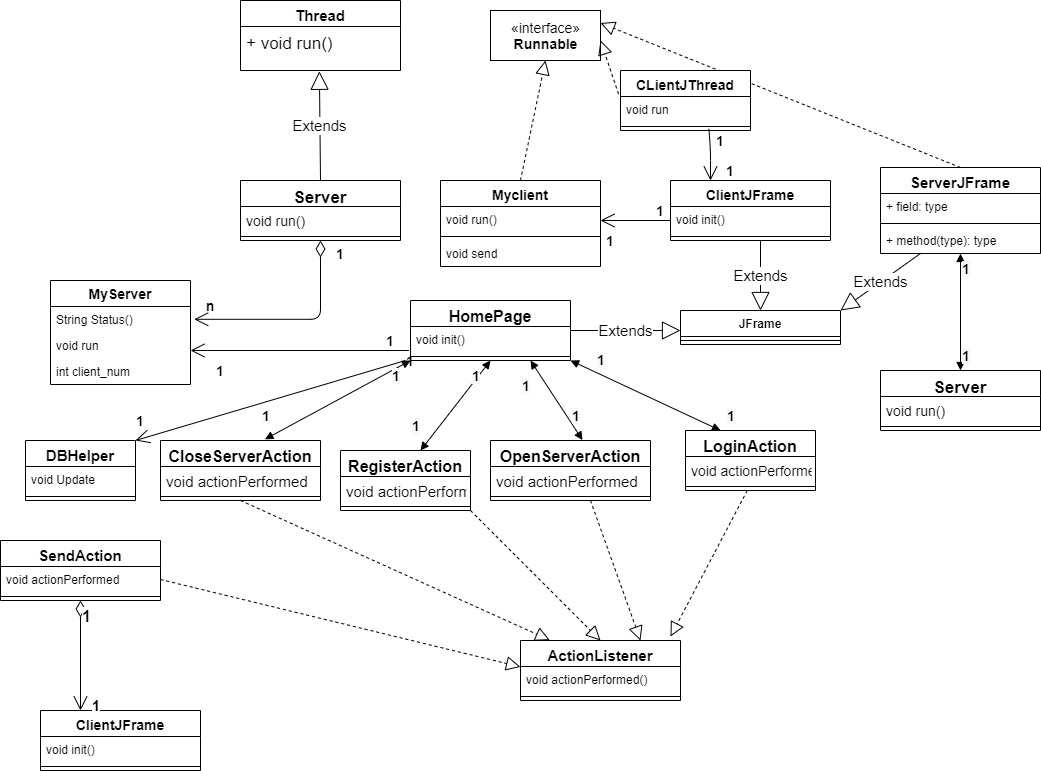
为了实现客户进程之间的通信，本聊天室使用多线程进行模拟。客户端之间的通信实际是客户进程与客户端进程之间的通信。为了实现实时的通信，使用多线程对客户通信进程进行仿真和模拟。模拟了进程之间的并发执行。

**4）与数据库相连实现用户管理**

在实现客户端之间通信之后，本聊天室与数据库相连接，建立用户表，对来到此聊天室的客户进行认证，只有先注册后，才能进行通信。

连接的数据库是Mysql8.0.

**5）系统UML图**



**2.实现内容**

**1）图形化界面的实现**

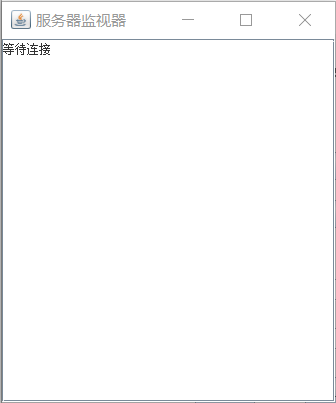
本聊天室的图形化界面如下：



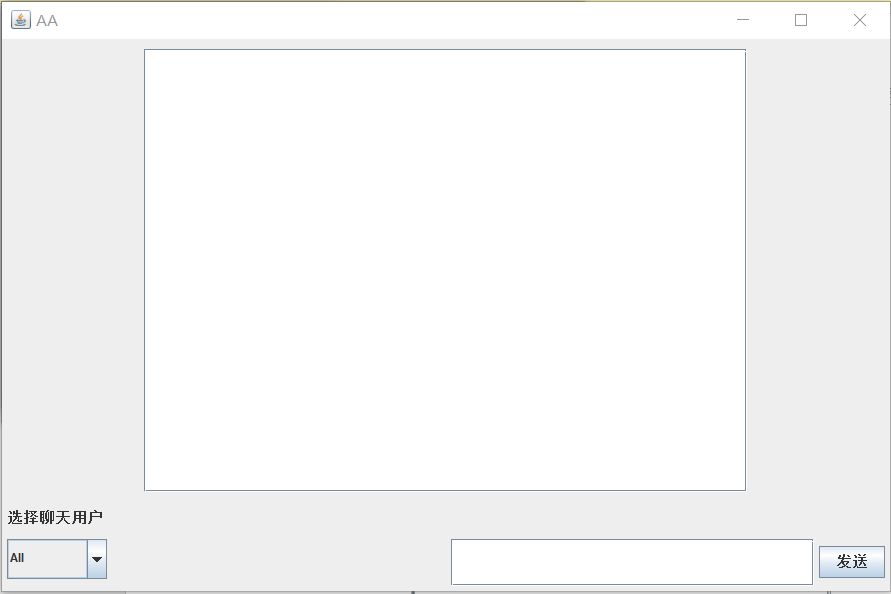
主页



打开服务器成功提示框



服务器监视器

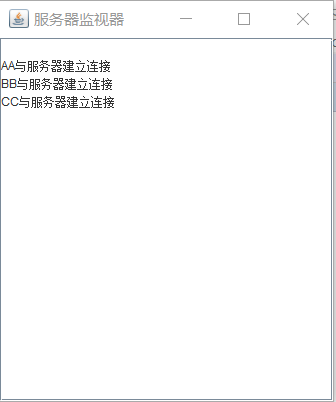


用户聊天界面

**2）实现客户端之间的通信**

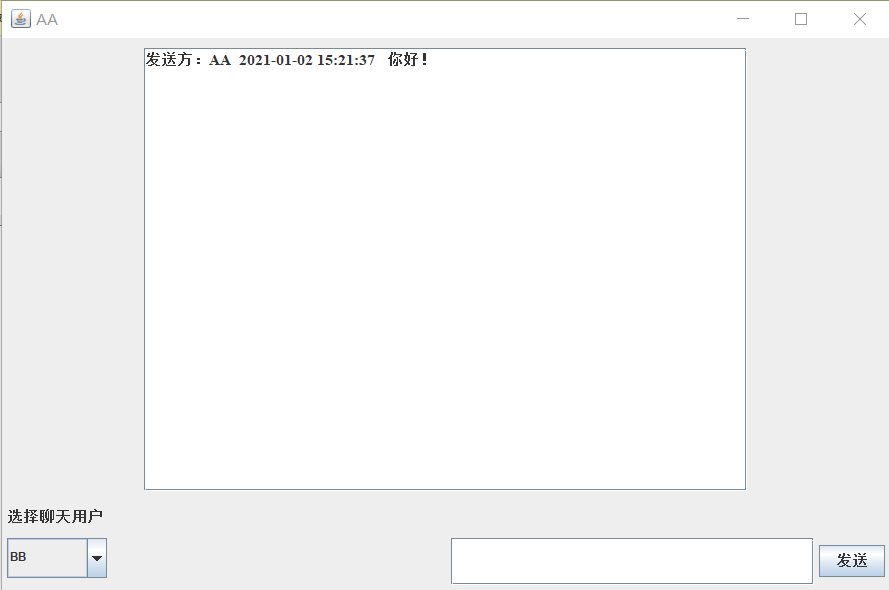
本聊天聊天室实现了客户端之间的通信，及客户端一方发送消息聊天室内所有客户都可接收到消息的功能。

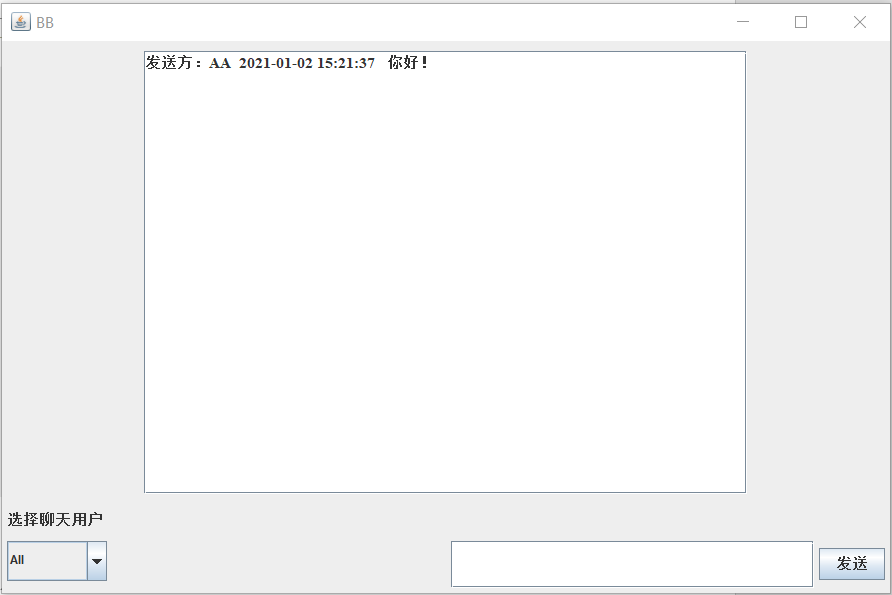
①客户进入聊天室时，服务器监视器中的情况。



②、客户端一对一的通信

发送方AA，将发送“你好”给BB，AA和BB界面都会显示相应的信息。

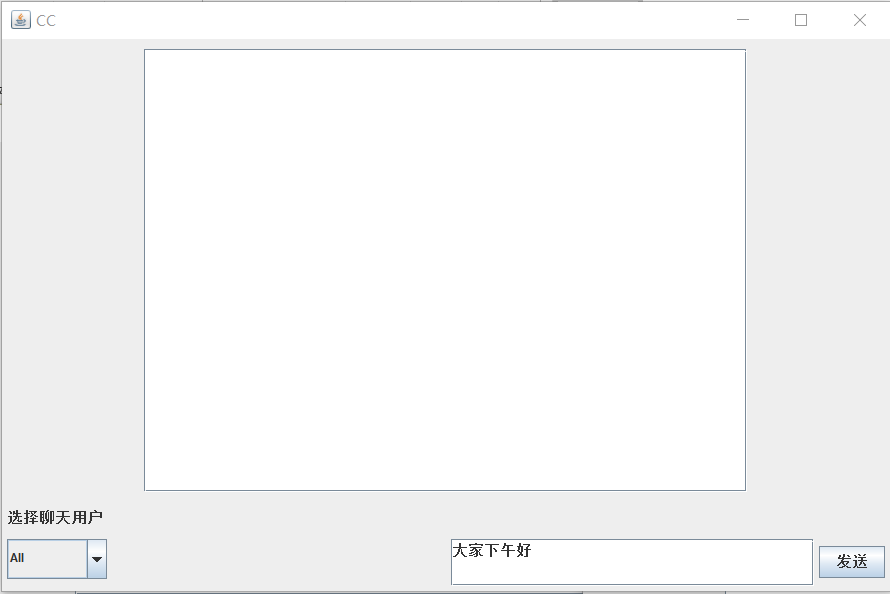




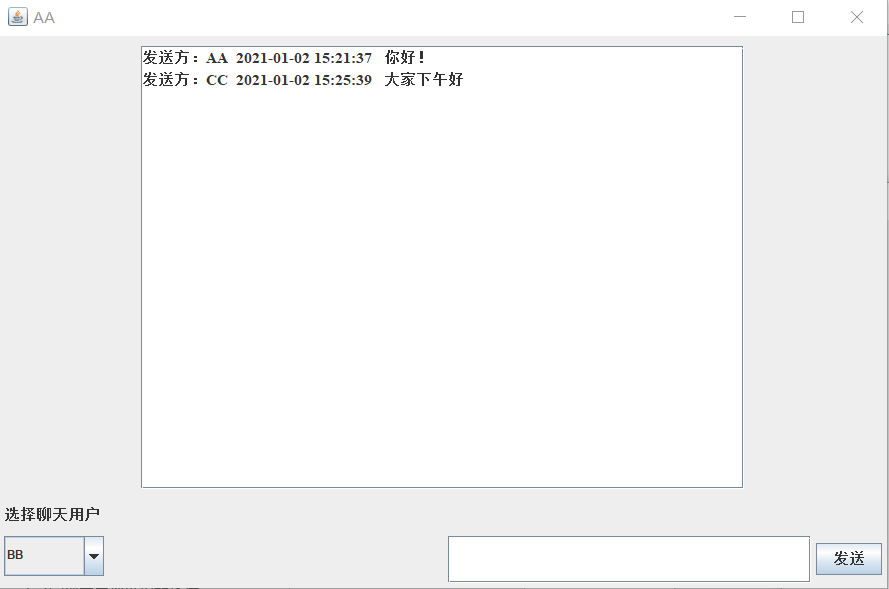
③、群聊机制

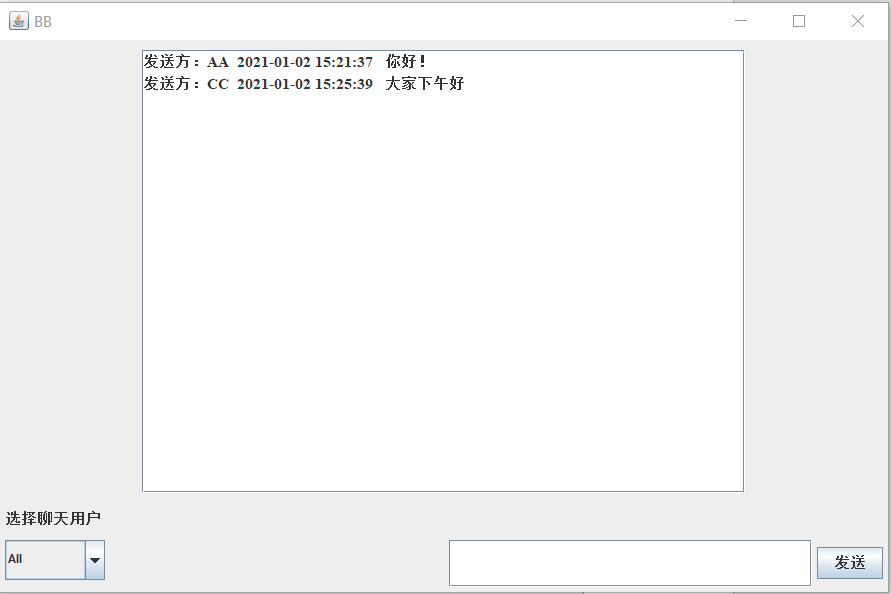
一个客户可以将消息发送给在聊天室内的所有客户，如CC客户端选择聊天用户为All,那么在聊天室所有客户都将收到消息。

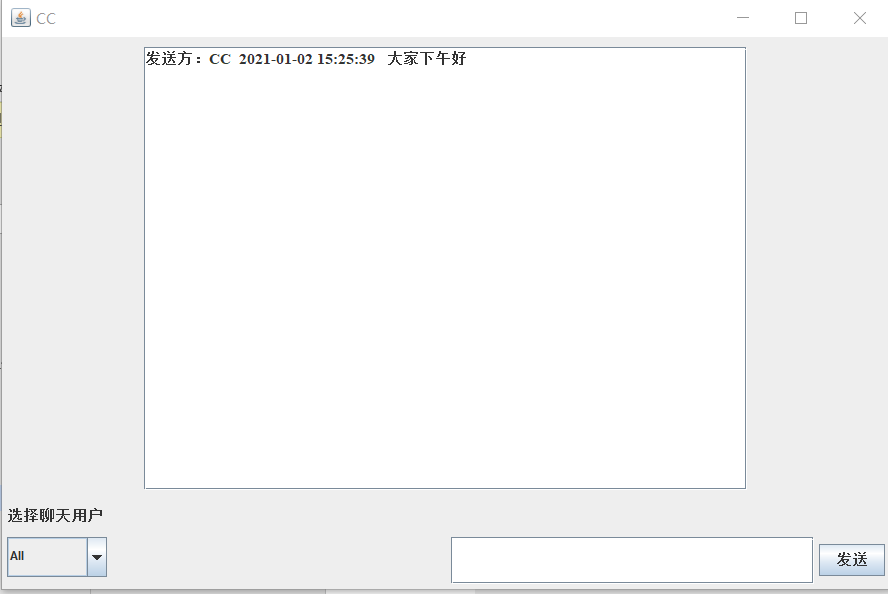
用户CC选择聊天用户为All.



接收方界面如图：



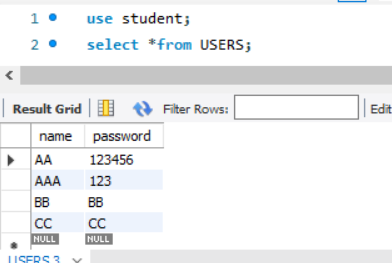




**3）用户的注册和登陆功能**

本聊天室与mysql数据库相连，实现了用户的注册及登陆功能，便于用户之间的管理，及认证用户身份，当用户进入聊天室时，需要登陆，当客户不是此聊天室的用户时，需要进行注册，使之成为聊天室的用户。当用户登陆时，密码不正确也不能进入聊天室，实现了用户的认证功能。

数据库中对应的表：



# 二、核心代码及运行结果展示

**1.图形化界面的编写**

本聊天室使用java.awt类库及swing组件实现了用户图形化界面及组件事件响应功能。

**2.使用线程、socket实现客户端进程的通信**

由上述分析可知，客户端之间的通信实际上是客户进程之间的通信，本聊天室使用线程模拟进程，实现客户端之间的实时通信。由于本聊天室基于socket进行通信。服务器端使用serversocket嵌套字与客户端socket进行建立连接，由于seversocket方法中的accept方法可以与客户端的socket建立连接，但是当没有客户端的socket与他建立连接时，他便进行阻塞，不符合服务器一直处于接收消息状态。当每次有个客户进行通信时，服务器端就开个线程为之服务。因此，服务器端也是利用多线程进行模拟实现。

**2.1客户端代码的实现**

①、客户端内含有唯一socket,及客户端用户的名字，及字符串txt表示从服务器接收到的消息。

**public** **class** MyClient **implements** Runnable{

Socket socket;

String name;

String txt="";

}

②、客户端在创建时，初始化自己的套接字，与服务器相连接并将自己的名字发送给服务器，便于显示在服务器监视器上。

**public** MyClient(String n)

{

**this**.name=n;

**try** {//将自己的名字发送给服务器，服务器建立线程

**this**.socket=**new** Socket("127.0.0.1",5432);

PrintStream out=**new** PrintStream(**this**.socket.getOutputStream());

out.println(name);

}

**catch**(Exception e)

{

System.***out***.println(name+"用户出了问题");

}}

运行结果如图：



③、当用户在聊天界面选择发送信息的用户并按下发送按钮时，调用客户端的发送功能函数，将对方的名字及通信的内容输入到socket流中，发送给服务器。发送消息时，会将目的地的名字和聊天内容当成一个字符串发送过去，这时需要进行特殊处理，我将用户名与消息内容之间加了一个到引号。到引号之前的字符串表示目的地，到引号之后的内容表示聊天内容。

**public** **void** send(String str,String des)

{

**try**

{

PrintStream out=**new** PrintStream(**this**.socket.getOutputStream(),**true**);

out.println(des+"`"+"发送方："+name+" "+str);//目的地dest用倒引号进行分割

System.***out***.println(name+"发送给"+des+" "+str);

System.***out***.println("发送信息成功");

}

**catch**(Exception e)

{

System.***out***.println(name+"客户发送信息出现问题");

}

}

④、接收功能，因为客户是被动接收消息的，不知何时服务器端会将消息发来，故用线程进行轮询操作，即一直在查询服务器端是否有消息传来。

**public** **void** run()

{

System.***out***.println(name+"接收消息");

**try**

{

**while**(**true**)

{

**try**

{

Thread.*sleep*(500);

}**catch**(InterruptedException e)

{

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*");

e.printStackTrace();

}

String line=**null**;

Scanner in=**new** Scanner(socket.getInputStream());

in.useDelimiter("\n");

**if**(socket.getInputStream().available()!=0)//缓冲区还有东西，否则会阻塞自己

{

**while**(socket.getInputStream().available()!=0)//要判断缓冲区有没有东西

{

line=in.nextLine();

txt=txt+line+"\r"+"\n";

}

System.***out***.println(name+"接收到 "+txt);

}

}

}

**catch**(Exception e)

{

System.***out***.println("接受信息出错");

}

}

**2.2服务器代码的实现**

服务器端使用线程实现，当接收到用户端的连接时，连接成功时，创建一个新的线程为其服务。并将其名字作为主键，和socket加入到map数据结构中，便于将消息发送给指定用户。

Server作为服务器的父线程为客户端的socket创建线程。

**public** **void** run()

{

**try**

{

**this**.serversocket=**new** ServerSocket(5432);

System.***out***.println("服务器正常启动!!");

**while**(**true**)

{

System.***out***.println("我是大线程");

Socket ss=**this**.serversocket.accept(); //建立连接

String name=(**new** MyServer(ss,**this**.SocketMap)).getname(); //建立为此客户传输消息的线程

**if**(text.equals("等待连接"))

{

text="";

}

**try**

{

text=text+"\n"+name+"与服务器建立连接";

}

**catch**(Exception ev)

{

System.***out***.println("生成子线程出错！");

}

}

}

**catch**(Exception e)

{

System.***out***.println("服务器出错！！！");

}

}

MyServer作为为客户服务的子线程，主要的作用有当这个线程创建时，将自己的名字和socket加入到数据结构Map中，注意这个Map是从父线程当中传来的，故这相当于每个客户都加入到一个Map中，便于索引查找。

同时还有存储、找到对应的客户进行转发功能。

①、创建时，读入所服务的客户端的名字，并将名字和socket一起加入数据结构Map。

**public** MyServer(Socket s,HashMap<String,Socket>So)

{

**super**();

**this**.socket=s;

*count*++;

**try**

{

Scanner in=**new** Scanner(socket.getInputStream());

in.useDelimiter("\n");

name=in.nextLine();

}

**catch**(Exception e)

{

System.***out***.println("创建输入流出错");

}

**this**.SocketMap=So;

So.put(name, s);//将自己加进去

**this**.start();

}

②、存储功能

**while**(socket.getInputStream().available()!=0)

{

line=in.nextLine();

txt+=line+"\r"+"\n";//txt+=line+"\n";

}

**int** index=txt.indexOf("`");

String othername=txt.substring(0,index);

//获取目的端用户名字

System.***out***.println("othername:"+othername);

**int** length=txt.length();

txt=txt.substring(index+1,length);//将聊天内容提取出来

根据客户端发送的信息，提取出来目的地的名称与聊天内容，即进行存储操作。

③、转发。根据目的地的名称，在Map中找到对应的socket，进行输出即转发。当目的地为“All”时，每个用户都将收到消息，否则只有一个用户收到相应的消息。

/\*不是发送给All\*///转发

**if**(!othername.equals("All"))

{

Socket i=**this**.findIndex(othername);//找到目的端的socket

**if**(i!=**null**)

{

PrintStream out=**new** PrintStream(i.getOutputStream(),**true**); //输入流

out.println(txt);//将消息发送给对方

}

}

**else**//是发送给All

{

Iterator<String>set=**this**.SocketMap.keySet().iterator();//建立键值集合

**while**(set.hasNext())

{

String key=set.next();

**if**(!key.equals(name))//不为自己时，需要建立流

{

Socket ss=**this**.SocketMap.get(key);

PrintStream out=**new** PrintStream(ss.getOutputStream(),**true**);

out.println(txt);

}}}

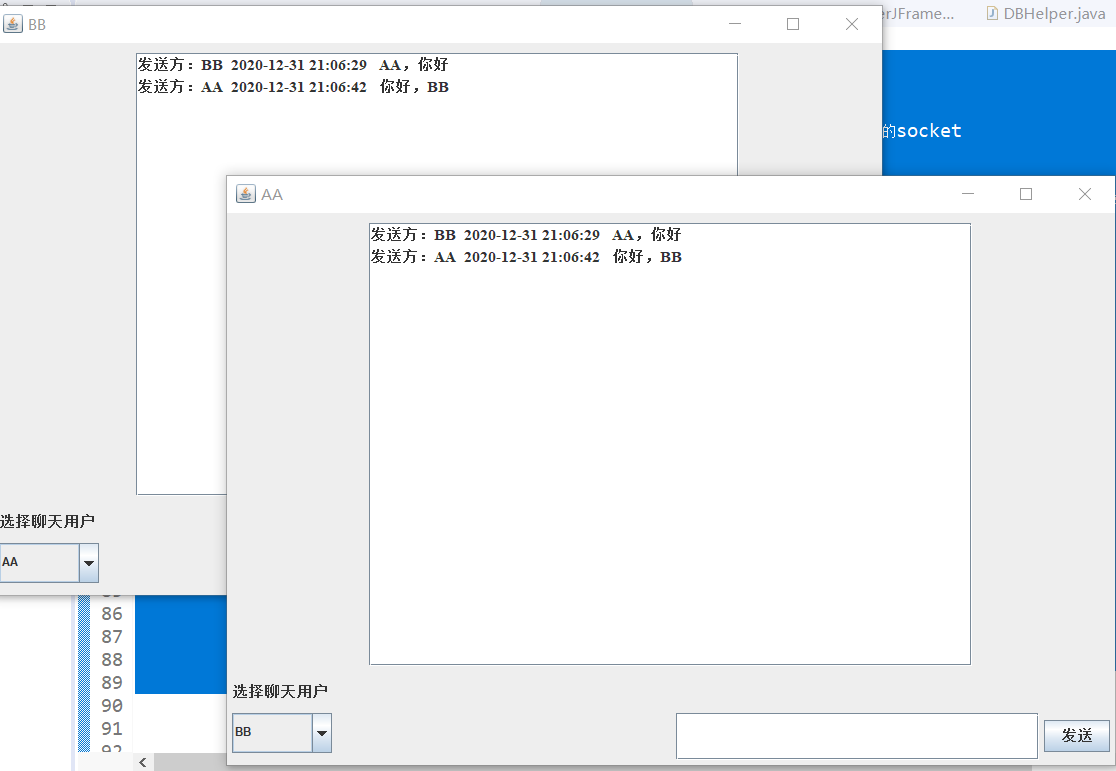
System.***out***.println(name+"服务器将发送给"+othername+txt);

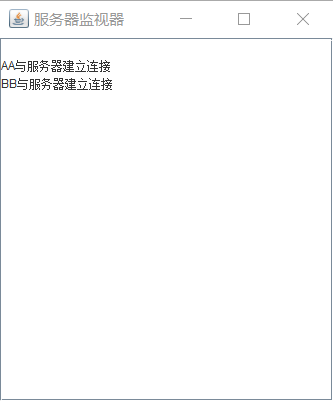
PrintStream outx=**new** PrintStream(**this**.socket.getOutputStream(),**true**);

outx.println(txt);//发送给自己

System.***out***.println(name+"服务器将发送给自己"+txt);

**2.3结果展示**





**3.与数据库相连接的操作**

本系统连接的是MySql数据库，在student表中有一张用户表，进行用户的注册和登录功能。

**3.1连接数据库**

**public** DBHelper(String sql) {

**try** {

Class.*forName*(***name***);//指定连接类型

conn = DriverManager.*getConnection*(***url***,***user***,***password***);//获取连接

System.***out***.println("数据库连接成功"+conn);

stmt=conn.createStatement();

} **catch** (Exception e) {

e.printStackTrace();

System.***out***.println(e.getMessage());

}

}

**3.2用户的注册和登陆功能**

①新用户在进入聊天室时，需要注册，即将用户名及密码加入到数据库中。

用户界面的RegisterAction中进行插入操作。

String sql="INSERT INTO USERS(name,password) values"+"('"+name+"',"+"'"+passwd+"'"+");";//插入语句

**this**.home.helper.eUpdate(sql);

**DBhelper类执行插入语句。**

**public** **int** eUpdate(String sql)//执行插入语句、更新语句、删除

{**int** rowCount=0;

**try** {

System.***out***.println("执行插入语句"+conn);

rowCount=stmt.executeUpdate(sql);//执行SQL 更新语句，包括Update, Delete和Insert语句}

**catch**(SQLException e)

{

System.***err***.println("发生异常"+e.getMessage());

}

**return** rowCount;}

②用户在登陆时，需要在数据库中查找，是否有该用户的信息，以及用户名所对应的密码是否正确，在用户表中，用户名作为主键。

String na=**this**.home.TextField.getText(); //获得用户名

String password=**this**.home.TextField1.getText();//获得密码

ResultSet rs=**this**.home.helper.select("select \*from USERS where name='"+na+"'");//获取数据库中的用户

**while**(rs.next())//从数据库中遍历名字

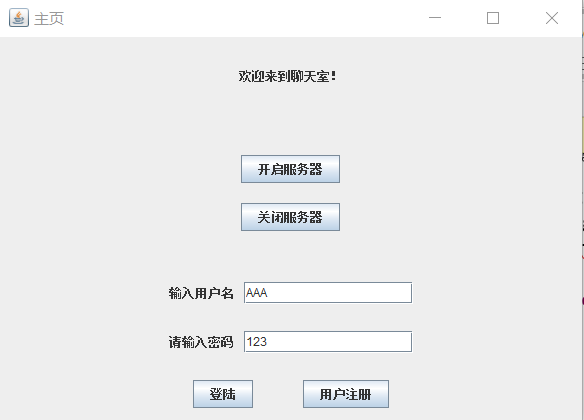
{

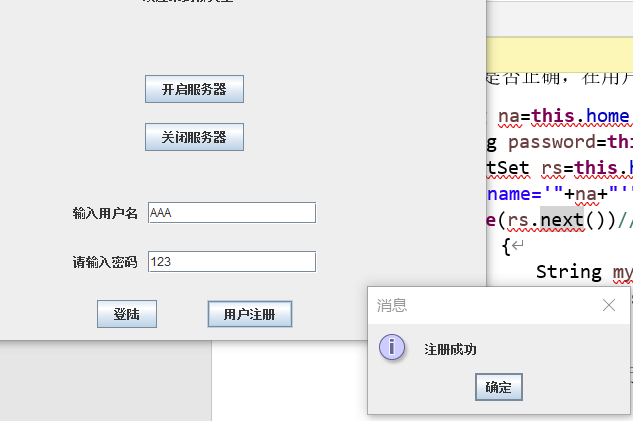
String myname=rs.getNString("name");//获取姓名

String passwd=rs.getNString("password");//获取密码}

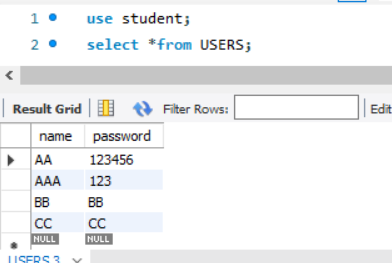
**3.3结果展示**

①进入聊天室之前，先进行用户注册

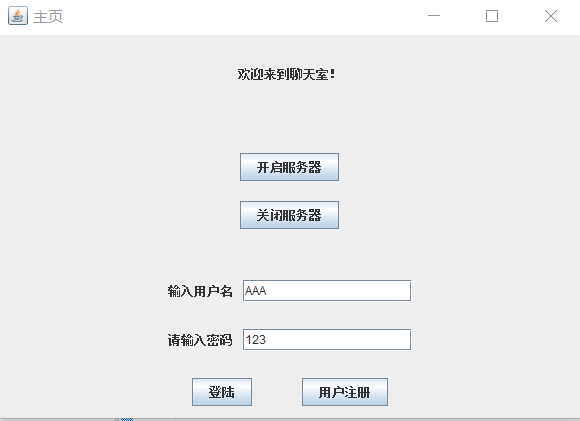


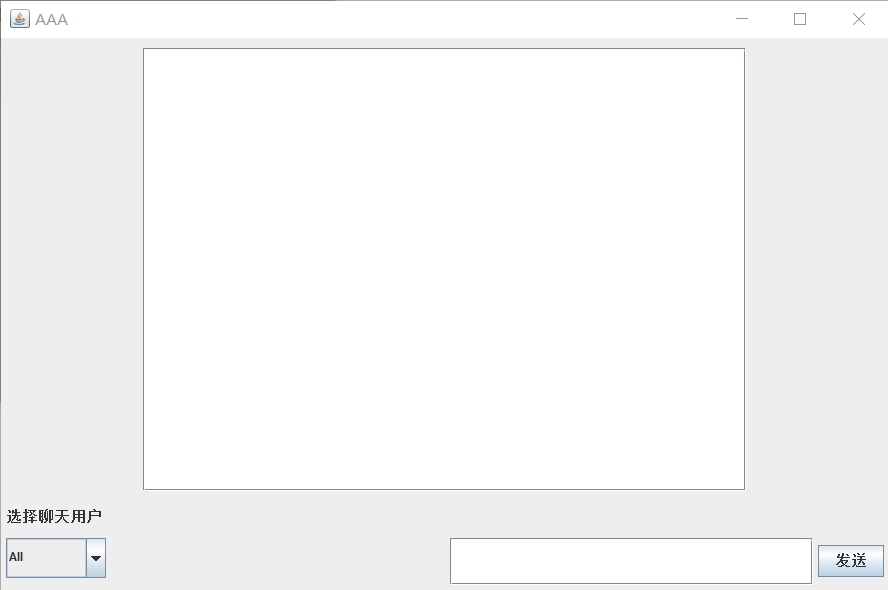


数据库中有相应的数据。

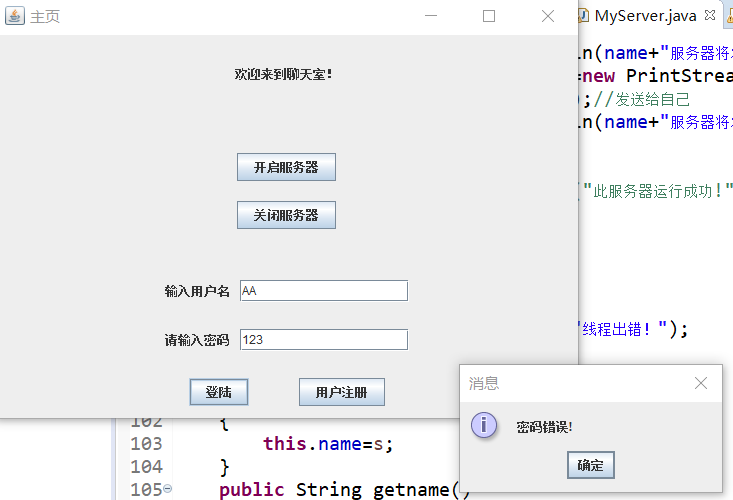


②、用户登陆时，输入正确的密码则进入聊天界面





③、输入错误的密码，则会弹出密码错误弹窗。



# 三、心得体会

**1.使用socket编程，对网络通信和线程有了更深的认识**

本次聊天室是基于socket之上，使用TCP/IP协议进行通信，进行可靠的连接。让我明白，客户端的socket和服务器中serversocket.accept建立的是一对一的联系，有点像点对点协议。除此之外，还明白了socket是通过输入输出流进行数据的传输。在实验过程之前，不明白为什么服务器要用多线程为客户服务，在实验时，尝试在服务器建立socket数组与客户端的socket相对应，但是发现Serversocket.accept在没有接受到客户端socket连接时，会将自己给阻塞了，无法进行后面的操作，当建立连接后，除非将此客户端socket结束即执行close()操作，才能接受到新的socket的连接。为了解决这个麻烦，于是我开始利用多线程，每次接受到客户端的一个连接，即新开一个线程为其服务，这样主线程就不会阻塞了，继续进行检测有无连接的操作。

在实验中还遇到另种会使线程阻塞的情况，即服务器线程run()函数中的循环是while(in.readLine())，其中in是socket的输入流的Reader，这样会使线程阻塞，原因是readLine()方法在进行读取一行时，只有遇到回车(\r)或者换行符(\n)才会返回读取结果，这就是“读取一行的意思”，重要的是readLine()返回的读取内容中并不包含换行符或者回车符；并且，当realLine()读取到的内容为空时，并不会返回 null，而是会一直阻塞，只有当读取的输入流发生错误或者被关闭时，readLine()方法才会返回null。

为了解决这个问题，我在while循环中，使用这个条件：socket.getInputStream().available()!=0，即判断输入输出流中是否有字节执行操作，这个操作不会进行阻塞。

**2.使用数据结构Map，对Map有了更深刻的认识**

在服务器进行转发的时候，我通过对目的地的查找，找出相应的socket，进行通信。最初我的想法是建立两个ArrayList，一个放客户端的名字，另一个放对应的socket，他们的下标形成一种映射关系，但是我发现Map这个对象容器里存放的是一种映射关系，相当于一张表。

在对Map查找过程中，先要建立键值的Set容器，然后根据键值获得相应的数据。与数据库中的关系表有异曲同工之妙。

**3.感悟**

在此次编程过程中，深刻体会到java的对底层函数的封装及面向对象之间相互调用是通过传参，即传递指针来使用的。在服务器主线程中定义了一个Map，为了将每个客户端的socket及其名字加进去，每个子线程也定义了一个Map，当创建子线程的时候，将这个Map作为参数传进去，当调用他们构造函数时，将其名字和socket加入，这就相当于加入一张表中。在本次实验中，我还连接了数据库，将对数据库的操作封装到一个类中，使用时，只用调用类中函数，这也是在数据库课设中没有试过的。在数据库课设中，只有用到sql语句时，连接数据库，并执行sql.但本次实验是通过执行类中的函数对数据库进行操作，很便利，容易修改！在做完实验时，内心还是很激动骄傲的！