



**计算机网络综合实验**

**实验报告**

题 目:**利用Socket编写聊天程序**

学 院: 网络空间安全与计算机学院

专业班级： 17计科二班

学 号： 20171101189

姓 名： 周朗睿

指导教师： 张虹

2019年 11 月21 日

**一、实验目的**

1. 基于socket实现简单的聊天程序

2. 通过socket程序的编写、调试，掌握socket编程的基本方法，了解客户端与服务器端的特点，掌握并熟悉运用socket类与serversocket类。

**二、实验内容**

1.学习Socket编程使用的类和接口，着重掌握基于TCP协议的Socket

2.编写服务器端和客户端的应用程序

3.实现服务器端和客户端的交互

**三、实验结果**

1、服务器端

（1）服务器界面



(2)重要代码

**- Server类**

A、服务器激活

**private** **void** execute(ActionEvent evt)

{

**int** port = Integer.*parseInt*(**this**.getJTextField().getText());

**new** ServerConnection(**this**, port).start();

}

B、添加窗口监听器，并设置异常处理方法

// 添加窗口监听器

**this**.addWindowListener(**new** WindowAdapter()

{

@Override

**public** **void** windowClosing(WindowEvent e)

{

// 添加异常处理

**try**

{

Collection<ServerMessageThread> cols = Server.**this**.map.values();

String messageXML = XMLUtil.*constructCloseServerWindowXML*();

**for**(ServerMessageThread smt : cols)

{

smt.sendMessage(messageXML);

}

}

**catch**(Exception ex)

{

ex.printStackTrace();

}

**finally**

{

System.*exit*(0);

}

}

});

**-ServerConnection类**

**A．连接服务器的方法**

**public** ServerConnection(Server server, **int** port)

{

**try**

{

**this**.server = server;

**this**.serverSocket = **new** ServerSocket(port);

// 下面两行代码不能放到上面一行代码之前

// 因为服务器启动时有可能会抛出异常

// 如果放在上面，则发生异常时也会导致label上的文本发生变化

// 这与实际情况不符

**this**.server.getJLabel2().setText("运行");

**this**.server.getJButton().setEnabled(**false**);

}

**catch**(Exception ex)

{

ex.printStackTrace();

JOptionPane.*showMessageDialog*(**this**.server, "端口号被占用！", "警告", JOptionPane.***ERROR\_MESSAGE***);

}

}

B．处理登录信息的方法

@Override

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

{

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

{

**try**

{

Socket socket = **this**.serverSocket.accept();

InputStream is = socket.getInputStream();

OutputStream os = socket.getOutputStream();

**byte**[] buf = **new** **byte**[5000];

**int** length = is.read(buf);

//客户端发来的连接信息（包括了用户名）

String loginXML = **new** String(buf, 0, length);

// 从客户端登录数据中提取出用户名信息（username）

String username = XMLUtil.*extractUsername*(loginXML);

String loginResult = **null**;

// 判断用户是否登录成功

**boolean** isLogin = **false**;

// 判断用户名是否重复

// 用户名重复

**if**(**this**.server.getMap().containsKey(username))

{

loginResult = "failure";

}

// 用户名不重复

**else**

{

loginResult = "success";

isLogin = **true**;

}

String xml = XMLUtil.*constructLoginResultXML*(loginResult);

os.write(xml.getBytes());

// 如果用户登录成功，启动线程

**if**(isLogin)

{

// 准备创建新的线程，用于处理用户的聊天数据，每一个连接上的用户都会对应一个该线程

ServerMessageThread serverMessageThread = **new** ServerMessageThread(**this**.server, socket);

//将用户名及与之对应的线程对象 放到Map中

**this**.server.getMap().put(username, serverMessageThread);

//更新用户列表（服务器端+客户端）

serverMessageThread.updateUserList();

serverMessageThread.start();

}

}

**catch**(Exception ex)

{

ex.printStackTrace();

}

}

}

**-ServerMessageThread类**

A．更新用户列表的方法

**public** **void** updateUserList()

{

//获得用户名的集合

Set<String> users = **this**.server.getMap().keySet();

//构造向客户端发送的在线用户列表数据xml

String xml = XMLUtil.*constructUserList*(users);

String str = "";

**for**(String user : users)

{

str += user + "\n";

}

//首先更新服务器端的用户列表

**this**.server.getJTextArea().setText(str);

Collection<ServerMessageThread> cols = **this**.server.getMap().values();

//遍历与每一个客户端对应的线程，向每一个客户端发送在线用户列表

**for**(ServerMessageThread smt : cols)

{

smt.sendMessage(xml);

}

}

B．向客户端发送数据

**public** **void** sendMessage(String message)

{

**try**

{

os.write(message.getBytes());

}

**catch**(Exception ex)

{

ex.printStackTrace();

}

}

C．处理聊天数据

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

{

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

{

**try**

{

**byte**[] buf = **new** **byte**[5000];

**int** length = **this**.is.read(buf);

//客户端发来的消息

String xml = **new** String(buf,0,length);

**int** type = Integer.*parseInt*(XMLUtil.*extractType*(xml));

// 聊天数据

**if**(CharacterUtil.***CLIENT\_MESSAGE*** == type)

{

//用户名（谁发来的消息）

String username = XMLUtil.*extractUsername*(xml);

//聊天的文本内容

String content = XMLUtil.*extractContent*(xml);

//构造向所有客户端发送的消息

String message = username + " : " + content;

//向所有客户端发送的XML聊天数据

String messageXML = XMLUtil.*constructServerMessageXML*(message);

Map<String, ServerMessageThread> map = **this**.server.getMap();

Collection<ServerMessageThread> cols = map.values();

**for**(ServerMessageThread smt : cols)

{

//向XML聊天数据发送给每一个客户端

smt.sendMessage(messageXML);

}

}

// 关闭客户端窗口

**else** **if**(CharacterUtil.***CLOSE\_CLIENT\_WINDOW*** == type)

{

String username = XMLUtil.*extractUsername*(xml);

//获得待删除用户所对应的线程对象

ServerMessageThread smt = **this**.server.getMap().get(username);

//构造出向客户端确认关闭的XML信息

String confirmationXML = XMLUtil.*constructCloseClientWindowConfirmationXML*();

//向客户端发送任意一条确认信息

smt.sendMessage(confirmationXML);

// 从用户列表的Map中将该用户去除

**this**.server.getMap().remove(username);

// 更新在线用户列表

**this**.updateUserList();

**this**.is.close();

**this**.os.close();

**break**; // 结束该线程

}

}

**catch**(Exception ex)

{

}

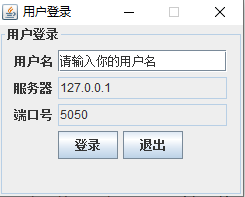
}

}

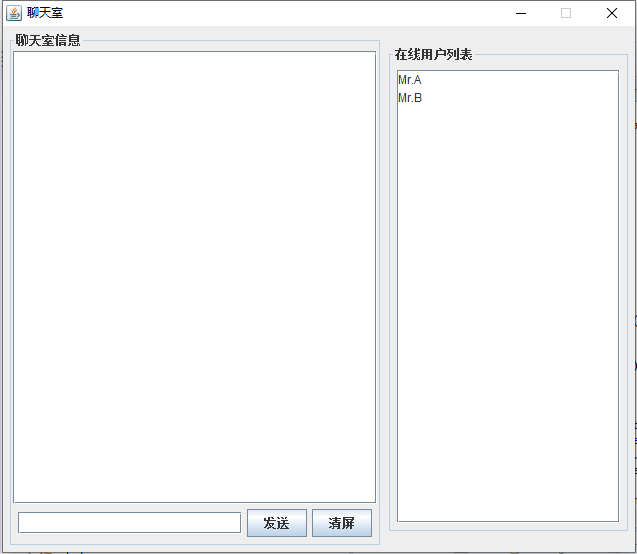
2、客户端

（1）客户端界面

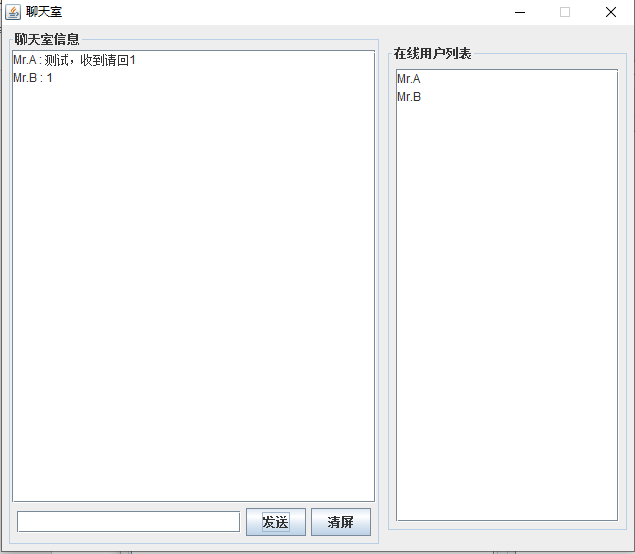
-登录界面



聊天界面



测试结果



（2）重要代码

**-Client类**

A．登录方法

**private** **void** login(ActionEvent event) {

String username = **this**.username.getText();

String hostAddress = **this**.hostAddress.getText();

String port = **this**.port.getText();

ClientConnection clientConnection = **new** ClientConnection(**this**,

hostAddress, Integer.*parseInt*(port), username);

**if** (clientConnection.login()) {

clientConnection.start();

} **else** {

JOptionPane.*showMessageDialog*(**this**, "用户名重复或服务器未运行！", "错误",

JOptionPane.***INFORMATION\_MESSAGE***);

}

}

**-ChatClient类**

A.添加监听器

jButton1.addActionListener(**new** ActionListener()

{

@Override

**public** **void** actionPerformed(ActionEvent e)

{

ChatClient.**this**.sendMessage(e);

}

});

**this**.addWindowListener(**new** WindowAdapter()

{

@Override

**public** **void** windowClosing(WindowEvent e)

{

**try**

{

ChatClient.**this**.clientConnection.sendMessage("client closed", "5");

}

**catch**(Exception ex)

{

ex.printStackTrace();

}

}

});

B．发送聊天数据的方法

**private** **void** sendMessage(ActionEvent event)

{

// 用户聊天的数据

String message = **this**.jTextField.getText();

// 清空聊天数据

**this**.jTextField.setText("");

// 向服务器端发送聊天数据

**this**.clientConnection.sendMessage(message, "2");

}

-ClientConnection方法

A．连接服务器的方法

**public** ClientConnection(Client client, String hostAddress, **int** port, String username)

{

**this**.client = client;

**this**.hostAddress = hostAddress;

**this**.port = port;

**this**.username = username;

//连接服务器

**this**.connect2Server();

}

// 连接服务器，由构造方法调用

**private** **void** connect2Server()

{

**try**

{

**this**.socket = **new** Socket(**this**.hostAddress, **this**.port);

**this**.is = **this**.socket.getInputStream();

**this**.os = **this**.socket.getOutputStream();

}

**catch**(Exception ex)

{

ex.printStackTrace();

}

}

B．用户登录，向服务器端传送用户名

// 返回true表示登录成功

// 返回false表示登录失败

**public** **boolean** login()

{

**try**

{

String xml = XMLUtil.*constructLoginXML*(**this**.username);

System.***out***.println(xml.toString());

os.write(xml.getBytes()); // 向服务器端发送用户的登录信息（其中包含了用户名）

**byte**[] buf = **new** **byte**[5000];

**int** length = is.read(buf); // 读取服务器端的响应结果，判断用户是否登录成功

String loginResultXML = **new** String(buf, 0, length);

String loginResult = XMLUtil.*extractLoginResult*(loginResultXML);

// 登录成功

**if**("success".equals(loginResult))

{

//打开聊天室主窗口

**this**.chatClient = **new** ChatClient(**this**);

**this**.client.setVisible(**false**);

**return** **true**;

}

// 登录失败

**else**

{

**return** **false**;

}

}

**catch**(Exception ex)

{

ex.printStackTrace();

}

**return** **false**;

}

C．发送聊天数据

**public** **void** sendMessage(String message, String type)

{

**try**

{

**int** t = Integer.*parseInt*(type);

String xml = **null**;

//客户端向服务器端发送聊天数据

**if**(CharacterUtil.***CLIENT\_MESSAGE*** == t)

{

xml = XMLUtil.*constructMessageXML*(**this**.username, message);

}

//客户端向服务器端发送关闭窗口的数据

**else** **if**(CharacterUtil.***CLOSE\_CLIENT\_WINDOW*** == t)

{

xml = XMLUtil.*constructCloseClientWindowXML*(**this**.username);

}

//向服务器端发送数据

**this**.os.write(xml.getBytes());

}

**catch**(Exception ex)

{

ex.printStackTrace();

}

}

D．更新客户端

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

{

**try**

{

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

{

**byte**[] buf = **new** **byte**[5000];

**int** length = is.read(buf);

String xml = **new** String(buf, 0, length);

**int** type = Integer.*parseInt*(XMLUtil.*extractType*(xml));

//在线用户列表

**if**(type == CharacterUtil.***USER\_LIST***)

{

List<String> list = XMLUtil.*extractUserList*(xml);

String users = "";

**for**(String user : list)

{

users += user + "\n";

}

**this**.chatClient.getJTextArea2().setText(users);

}

// 服务器端发来的聊天数据

**else** **if**(type == CharacterUtil.***SERVER\_MESSAGE***)

{

String content = XMLUtil.*extractContent*(xml);

**this**.chatClient.getJTextArea1().append(content + "\n");

}

// 关闭服务器端窗口

**else** **if**(type == CharacterUtil.***CLOSE\_SERVER\_WINDOW***)

{

JOptionPane.*showMessageDialog*(**this**.chatClient, "服务器端已关闭，程序将退出！", "信息", JOptionPane.***INFORMATION\_MESSAGE***);

System.*exit*(0); //客户端退出

}

// 服务器端确认关闭客户端窗口

**else** **if**(type == CharacterUtil.***CLOSE\_CLIENT\_WINDOW\_CONFIRMATION***)

{

**try**

{

**this**.getSocket().getInputStream().close();

**this**.getSocket().getOutputStream().close();

**this**.getSocket().close();

}

**catch**(Exception ex)

{

}

**finally**

{

System.*exit*(0);//退出客户端程序

}

}

}

}

**catch**(Exception ex)

{

ex.printStackTrace();

}

}

**四、分析及反思**

此次实验是对socket的一个应用，让我更深地了解了socket函数的作用及使用方法。通过这次实验能够感受到网络在实际当中的运用，使我对网络编程有了更深的概念。同时让我发现自己知识结构的欠缺。虽然基本功能已经实现，但是还是存在很多问题尚待解决。

实验过程中遇到了很多问题，例如：程序只能通信一次。刚开始我以为建立的socket套接口只要建立一次并更新连接地址就可以多次使用。当初不知道问题存在的在于这，在程序加入很多错误报告代码，发现在客户端再次connect的时候返回负值。在网上查看很多相关代码之后，发现问题所在，把建立socket的代码写进客户端的循环体中，每次通信都重新建立socket套接口就可以了。