面向对象程序设计（Java）

实验报告

实验七 聊天网络程序设计实验

|  |  |
| --- | --- |
| 学院名称 | 计算机科学与技术学院 |
| 专业班级 | 信安1802 |
| 学生姓名 | 侯天铭 |
| 学号 | 5120182377 |
| 指导教师 | 刘涌 |

2019年 5月

**一、实验目的**

练习基于Socket的网络访问方法，理解TCP/IP传输层网络程序的设计思想与实现方法。

**二、实验内容**

设计通过网络进行聊天的Java应用程序，实现如下功能：

（1）聊天两端具有相同的界面，包括能输入消息的文本框、发送消息按钮、消息列表文本框；

（2）任意一端均可以实时发送和接收消息，并在自身的消息列表中显示消息记录。

**三、概要设计**

1. **需求分析过程**

**用户对界面的需求：**作为一个聊天软件，应具备的最基本成分：输入发送消息的文本框；接收消息同时也是显示历史聊天记录的文本框；以及一个发送按钮。

**用户对使用过程的需求：**用户在使用过程中，首先会把需要发送的消息输入到用以发送消息的文本框中，然后或是按下回车，或是鼠标点击“发送”按钮将消息发送给对方。并且于此同时，输入框中的内容即刚刚完成发送的内容应被清空，接收消息文本框中应显示出用户自己刚刚发送的消息。并且在双方互相发送多条消息后，能够通过上下拉动文本框来查看历史聊天记录。

1. **总体设计方案**
2. **界面设计**

首先是在getContentPane中设置了一个JTextField和一个JButton组件，其中JTextField用于输入需要发送的内容，JButton为一个“发送”按钮，实现发送功能。然后再添加一个JScrollPane容器，并在其中填充一个JTextArea用以接收对方发送给用户的消息，同时也是一个显示历史聊天记录的文本框。（使用JScrollPane是考虑到双方互发消息过多时，能够上下拉动查看历史消息）

1. **事件设计**

一个基本的聊天工具，实现的功能无非就是消息的发送与接收。而从用户使用的角度出发，消息的接收显然不需要用户亲自动手，自然也就不是我们所需设计的事件。因此消息“发送”为唯一需要设计的事件，其完成的功能也就是将文本框中输入的文本发送给对方。只不过考虑到用户的偏好，同时设置了textFiled的actionPerformed事件以及一个“发送”按钮，使得用户既能通过按下回车发送消息，又能通过点击“发送”按钮发送消息。

**四、详细设计**

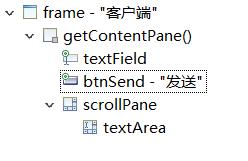
**1、类的设计**

**（1）客户端：**首先把客户端的窗体界面设计完毕。然后为了使后续的消息发送接收等功能不影响窗体的正常显示，我们在主方法中的run()方法里面仅仅是完成窗体的初始化与显示，将后面的语句均放到主方法内，run()方法外，也就是使其分属两个不同的线程。作为客户端而言，要与服务端通信，首先需要创建一个Socket对象：Socket client = new Socket(host,port)，两个传入参数分别指明主机名（IP）以及开放端口号（通信服务运行的端口）。此后Socket对象client便是客户端与服务端之间通信的一个桥梁，其可以使用OutputStream向服务端发送信息，也可以使用InputStream接收服务端传输过来的信息。在此先使用client.getInputStream打开输入流获取服务端的消息，为了解决中文乱码问题，我在此使用InputStreamReader is = new InputStreamReader(client.getInputStream());然后使用BufferedReader包装一下。之后就能使用BufferedReader对象br来获取服务端传输的消息了。接着，由于服务端传输消息的频率（也就是聊天对象发送的消息的频率）是不可能确定的，因此我们只能使用一个while(true)来无限循环获取消息。如果服务端没有消息传过来，自然什么也不做，但只要有消息过来，便将其显示在聊天记录文本框上。

**（2）服务端：**服务端类的设计与客户端大同小异。客户端首先是要创建Socket对象，而服务端需要先创建一个ServerSocket对象：ServerSocket sserver = new ServerSocket(port)，传入参数指定服务端通信服务开启的端口。然后使用该对象的accept方法，返回一个新的Socket对象：Socket server = sserver.accept();同理，该server也就成为了服务端与客户端之间通信的桥梁。往后同客户端，还是使用server.getInputStream()获取输入流，然后一个while(true)死循环，循环获取客户端发送的消息并显示在消息记录上。

**2、界面设计**

客户端程序与服务端程序的界面完全相同。使用BorderLayout，首先在getContentPane中上下分别放置一个JTextField（输入消息）和JButton（“发送”按钮）组件，然后在中央放置一个JScrollPane容器（添加滚动条），在容器内部加入一个JTextArea（接收消息，显示聊天记录）。层次结构如下：



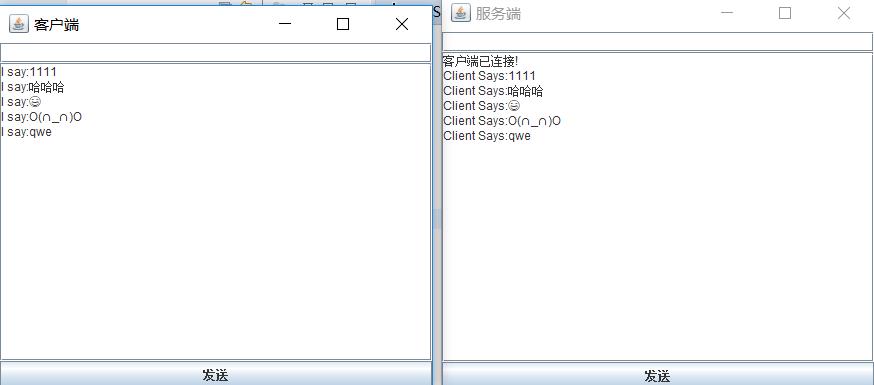
1. **事件设计**

客户端和服务端都具有两个事件：textField的actionPerformed事件（主要是监听回车），以及一个button“发送”按钮事件，两个事件的功能相同，即发送当前textField中的内容。用户在输入框中输入想要发送的消息，然后点击“发送”按钮，或是直接按下回车键，均可将文本框中的文本发送出去。由于两个事件实现的功能完全相同，代码也一致，因此以下就当作一个事件来分析了。

1. **客户端：**当用户点击“发送”按钮，或是按下回车键的时候会触发此事件。首先，利用Socket对象client来获取输出流，并将返回的流传给PrintStream类的构造方法来创建一个PrintStream输出流对象ps：PrintStream ps = new PrintStream(client.getOutputStream()); 接着，利用ps.println()方法便能将textField中的内容写入到输出流：ps.println(textField.getText()); 然后服务端那边通过server.getInputStream()便能创建一个输入流来接收客户端发送过去的消息，然后显示出来即可。然后考虑到qq等聊天软件的特性，此时客户端发送的消息服务端能够接收并显示在服务端的接收消息文本框中，但是客户端这边并没有显示，因此还需要将这一段文本显示在客户端的textArea中：textArea.append(“I say:”+textField.getText()+“\n”); 然后将textField中的内容清空：textField.setText(“”); 。
2. **服务端：**服务端与客户端基本相同，只不过Socket对象变为了server，于是使用server.getOutputStream()来获取输出流。往后还是使用ps.println(textField.getText())将textField中的文本写入到输出流，然后客户端那边就能通过client.getInputStream()来获取输入流，进而读取服务端发送的消息了。最后将发送的内容显示在服务端的消息记录上，并清空消息输入文本框。

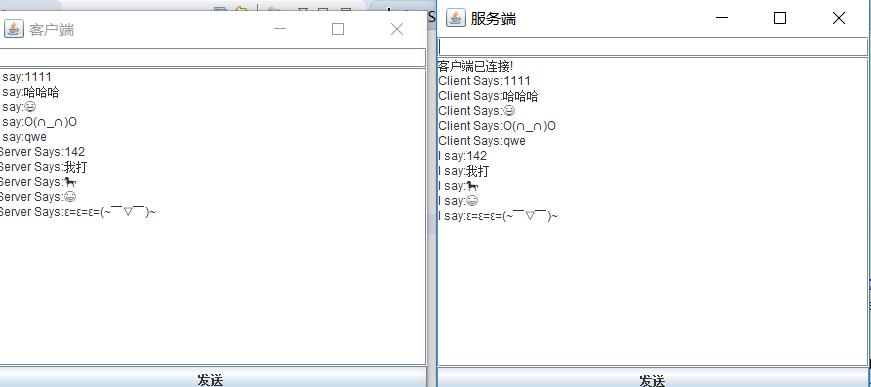
**五、实验结果**

**测试1、客户端向服务端发送消息：**

****

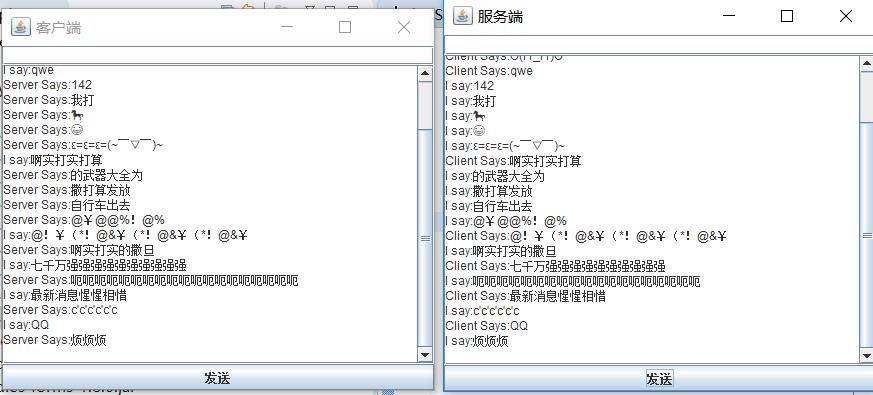
可以看到，客户端向服务端发送数字，汉字，表情，符号，英文字母都能被服务端正常接收并显示出来。

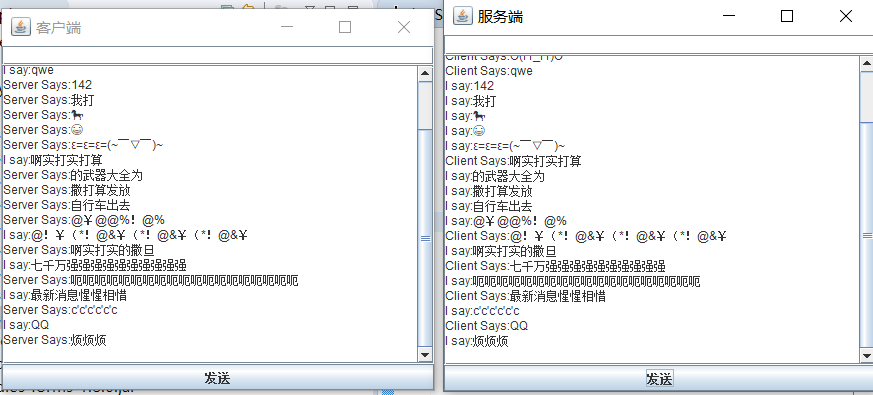
**测试2：服务端向客户端发送消息：**

****

同样的，服务端也能向客户端发送任意形式的文本内容，并且客户端能够正确接收并显示在聊天界面上。

**测试3：双方多次发送消息：**

****

可以看到，当双方多次进行消息往来导致消息过多时，聊天界面会出现滚动条使得用户能够上下拉动查看聊天记录。

**结论：**客户端能够与服务端完成网络连接，并实现实时的发送消息、接收消息功能，并且双端收发的消息内容从字符到汉字不受限制。因此，可以说本次设计的客户端程序与服务端程序实现了一个聊天软件的基本功能。

**六、讨论与分析**

**对课程知识点的理解：**

本次实验的目的是：练习基于Socket的网络访问方法，理解TCP/IP传输层网络程序的设计思想与实现方法。关于Socket到底是个什么东西，老师上课讲过，我也上网查过。经过这次实验，我认为Socket就是一个双向的通信通道，就java程序来说，通信的每一端都需要有一个Socket对象，比如本次的server和client分别是为服务端和客户端的Socket对象。在创建了这两个对象之后，客户端与服务端之间就能通过这两个Socket对象来进行通信了。具体是：客户端利用client的getOutputStream()方法输出的内容，服务端能够通过server的getInputStream()来接收；同理，服务端利用server.getOutputStream()输出信息，客户端便利用client.getInputStream()来接收信息。

**对设计过程的理解：**

就设计过程来说，还是采用“从外观到功能”的顺序。不管设计的是什么程序，首先总得有一个界面，于是首先添加两个文本框，一个用来输入发送的消息，一个用来显示接收的消息，外加一个“发送”按钮。然后再来考虑这两个文本框，以及这个按钮应该实现什么样的功能，才进一步去设计相应的事件，写相应的代码。当客户端与服务端打开后，首先应该完成网络连接，而此时用户并没有进行任何操作，所以创建连接显然不能写作事件，而是在主方法里完成。同时，接收对方消息也不能算是事件，因此同样写在主方法，从程序启动便开始读取，只要读取到发送过来的消息便显示出来。然后对于方法而言，只有一个“发送”，由于连接已建立，因此事件代码直接获取输出流，输出信息即可。

**实验结果的分析：**

首先，就像上面说过的，客户端与服务端之间能够实现实时聊天，完成了聊天程序的基本功能的设计。但是我又想，到此为止，该程序还只能通过输入文字或符号之类进行文本类的聊天，若是用户想要发送图片怎么办？这就还需要进行改进。所以目前为止，该程序还未能实现图片的发送与接收。

**需要引起注意的问题：**

**问题一**就是上面说过的，该程序还不能进行图片的发送与接收，只限于文本聊天。

**问题二**就是对于输入消息框，我们知道像qq这样的聊天软件，每一行输入的文本是有限的，超过了就会自动换到下一行，但是该程序并不能。

**问题三**其实和问题二很像，虽然本程序已经实现了当聊天记录过多时出现滚动条方便上下拖动，但同样有一个换行问题。如果说用户发送的一条消息过长，那不仅是在用户的输入框内连成一行，在发送出来后显示在消息记录里还是连成了一行，不方便查看。

**七、附录：关键代码**

**1、客户端与服务端建立连接+接收服务端消息：**

**try** {

*client* = **new** Socket("127.0.0.1",1111);//客户端创建Socket对象（与服务端建立连接）

InputStreamReader isr = **new** InputStreamReader(*client*.getInputStream());//获取输入流

BufferedReader br = **new** BufferedReader(isr);//采用缓冲流包装

String s;

s = br.readLine();

**while**(**true**) { //循环读取信息

*textArea*.append("Server Says:"+s+"\n");

s = br.readLine();

}

} **catch** (UnknownHostException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**2、客户端“发送”事件：**

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

**try** {

*ps* = **new** PrintStream(*client*.getOutputStream());//获取输出流

*ps*.println(textField.getText());//输出信息

*textArea*.append("I say:"+textField.getText()+"\n");//将输出信息显示在客户端消息记录里

textField.setText("");//将文本框中已输出内容清空

} **catch** (UnknownHostException e1) {

// **TODO** Auto-generated catch block

e1.printStackTrace();

} **catch** (IOException e1) {

// **TODO** Auto-generated catch block

e1.printStackTrace();

}

}

**3、服务端与客户端建立连接+接收客户端消息：  
try** {

ServerSocket sserver;

sserver = **new** ServerSocket(1111);//创建ServerSocket对象

*server* = sserver.accept();//建立连接，返回Socket对象

*textArea*.append("客户端已连接!\n");//客户端成功连接后提示

InputStreamReader isr = **new** InputStreamReader(*server*.getInputStream());//获取输入流

BufferedReader br = **new** BufferedReader(isr);//缓冲流包装

String s;

s = br.readLine();

**while**(**true**) {//循环读取消息

*textArea*.append("Client Says:"+s+"\n");

s = br.readLine();

}

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**4、服务端“发送”事件：**

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

**try** {

*ps* = **new** PrintStream(*server*.getOutputStream());//获取输出流

*ps*.println(textField.getText());//输出消息

*textArea*.append("I say:"+textField.getText()+"\n");//将输出消息显示在服务端消息记录里

textField.setText("");//将文本框中已输出内容清空

} **catch** (UnknownHostException e1) {

// **TODO** Auto-generated catch block

e1.printStackTrace();

} **catch** (IOException e1) {

// **TODO** Auto-generated catch block

e1.printStackTrace();

}

}