|  |  |  |
| --- | --- | --- |
| 南农大  **Java大作业实验报告**  XH2 | | |
|  | 题 目: | 基于Socket通讯原理的网络聊天室通讯 |
|  | 姓 名: | 董自经 学号: 19222126 |
|  | 学 院: | 人工智能学院 |
|  | 专 业: | 计算机科学与技术 |
|  | 班 级: | 计科221 |
|  | 指导教师: | 谢忠红 职称： 副教授 |
| 年 月 日 | | |

目录

[**基于Socket通讯原理的网络聊天室通讯** 1](#_Toc169456908)

[一、阶段二实验目标 1](#_Toc169456909)

[二、实验原理 1](#_Toc169456910)

[（一）Socket连接 1](#_Toc169456911)

[（二）输入输出流 1](#_Toc169456912)

[（三）线程处理 1](#_Toc169456913)

[三、实验内容 2](#_Toc169456914)

[四、实验结果 2](#_Toc169456915)

[五、实验不足 3](#_Toc169456916)

[六、实验小结 4](#_Toc169456917)

[七、源代码 4](#_Toc169456918)

**基于Socket通讯原理的网络聊天室通讯**

计科221董自经

一、阶段二实验目标

利用Socket通讯原理，实现服务器和客户端之间发送信息和接收信息的简单信息通讯，需要设计服务器和客户端的界面，服务器和客户端为一对一模式。

二、实验原理

要实现客户端和服务器端之间相互的通信，需要建立Socket连接，建立连接之后Server和Client都会产生一个Socket对象，两个界面可以通过这个对象实现文本信息交换。同时建立线程对象实现多服务器和多用户同时对接的目标。

（一）Socket连接

在网络通信中，Socket连接是实现客户端和服务器端通信的关键机制。通过使用Socket，客户端和服务器可以建立一个双向通信的数据通道。Socket提供了一种抽象层，使得不同主机上的应用程序能够通过网络进行数据交换。客户端和服务器通过互相知晓的IP地址和端口号来建立连接。一旦Socket连接建立成功，客户端和服务器端即可互相发送和接收数据。

（二）输入输出流

Socket对象中包含了输入流和输出流，这两种流分别用于从客户端读取数据和向客户端发送数据。通过输入流，服务器可以接收来自客户端的请求、指令或数据；而通过输出流，服务器可以将处理结果、响应或其他信息发送回客户端。输入输出流使得Socket连接实现了灵活的双向数据传输，为客户端和服务器端之间的通信提供了便利的手段。

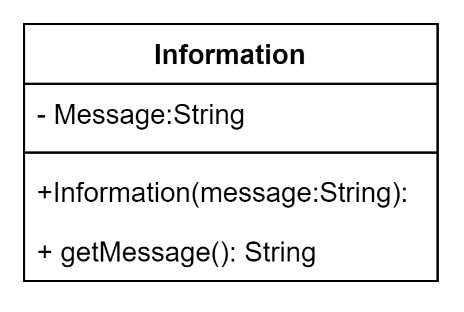
（三）线程处理

当服务器端接收到客户端的连接请求时，通常会将客户端的Socket对象传递给一个单独的线程进行处理。每个客户端连接对应一个线程，这样服务器端可以同时处理多个客户端的请求，而不会因为一个客户端的阻塞操作而影响其他客户端的连接。使用线程处理客户端连接可以实现并发处理，提高服务器的性能和响应速度。

三、实验内容

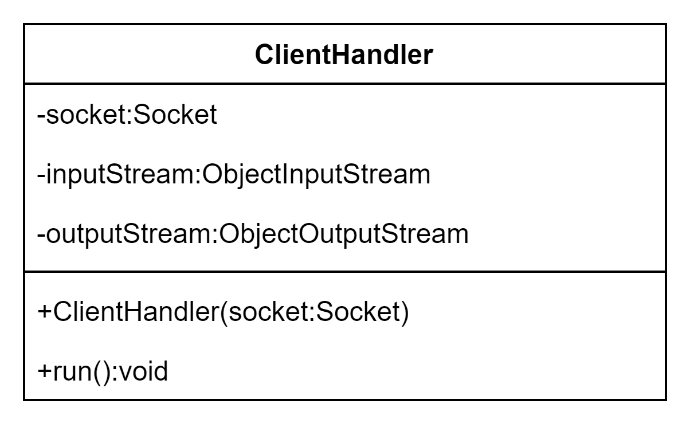
实验中建立了四个.java文件，分别是ServerFrame.java，ClientFrame.java，ClientHandler.java，Information.java其功能以及函数如下所示：

在Information类中主要是对输入的类型做了定义，只能够输入字符串类型数据



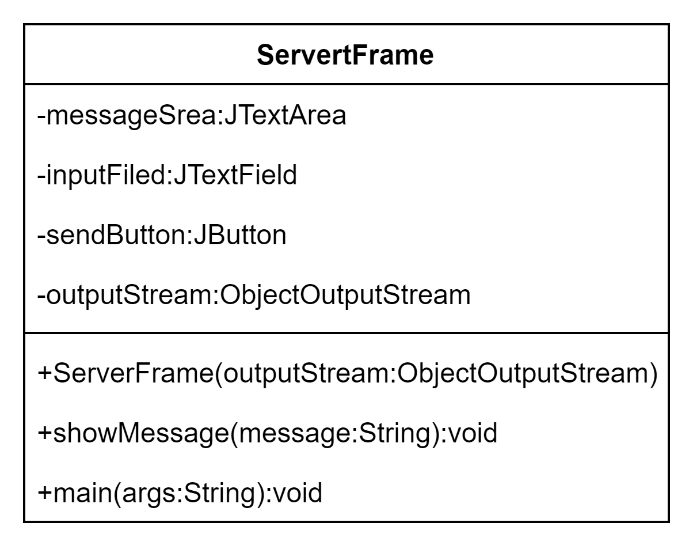
图一：类Information

在ClientHandler类中主要实现了多线程的功能



图二：类ClientHandler

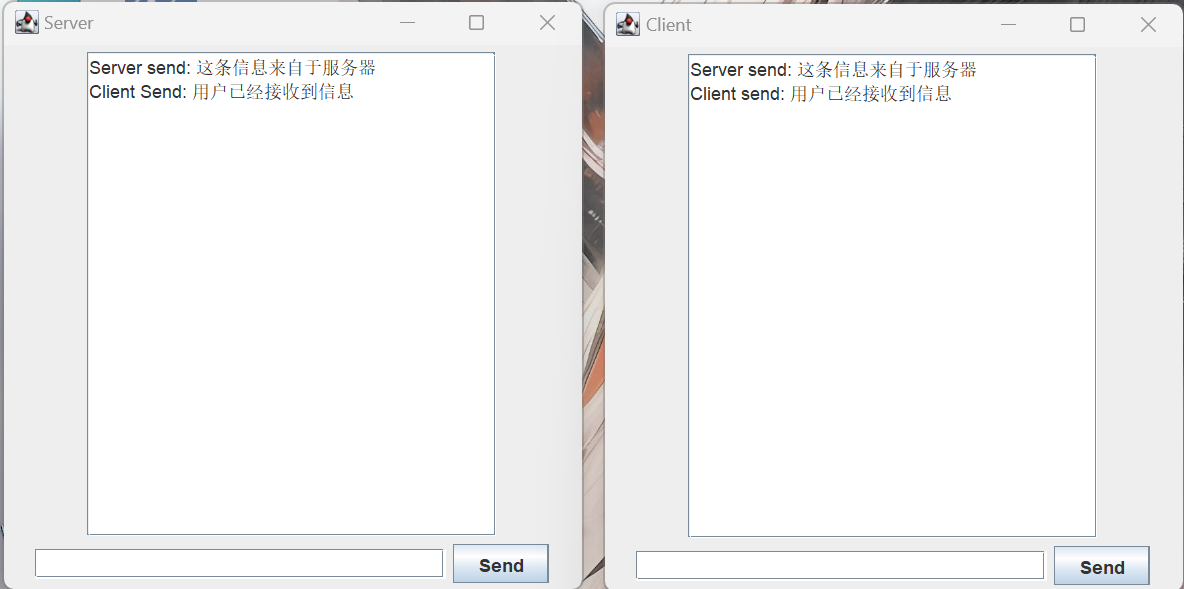
ClientFrame和ServerFrame实现了图形化界面创建和基本的通讯建立



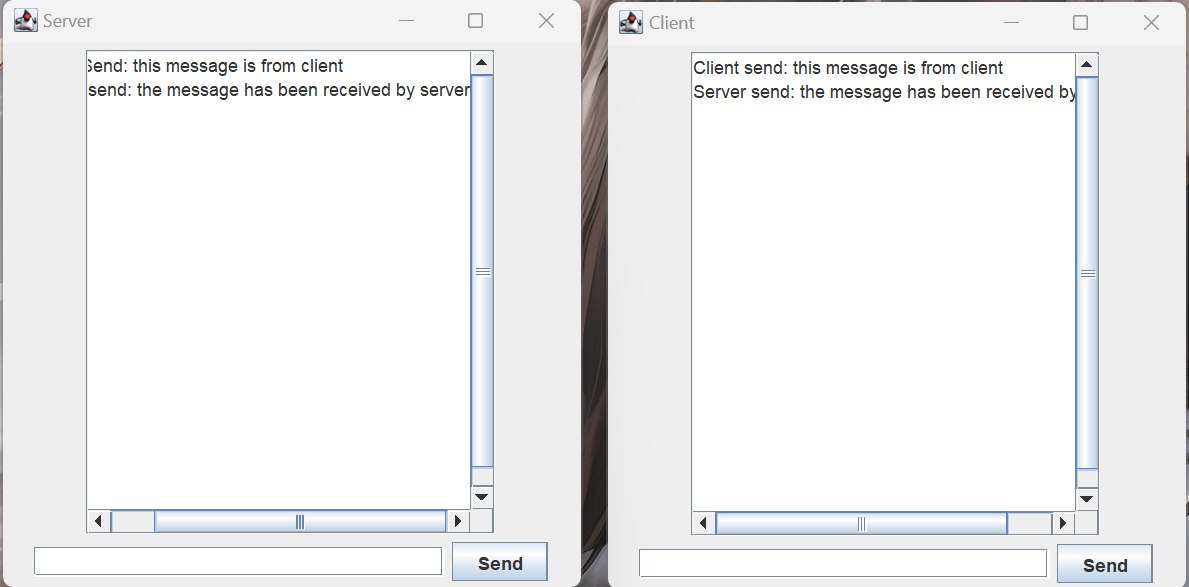
图三：类ClientFrame 图四：类ServerFrame

四、实验结果

实验时先运行ServerFrame.java文件建立起服务器的端口，之后再运行ClientFrame.java文件实现客户端。同时还可以多次运行ClientFrame.java文件，系统会自动为后续客户端建立服务器端口。具体运行结果如下图：



图五：运行结果1



图六：运行图五时多线程工作

五、实验不足

实验未能实现一个服务器向多个用户发送消息。

实验中的server均由ClientHandler.java文件产生，一个关闭后所有server服务器均关闭。

六、实验小结

通过Java中的Socket类，可以实现服务器和客户端之间的通信。服务器端可以监听指定端口，等待客户端发起的连接请求；而作为客户端则可以通过指定服务器的IP地址和端口号，发起连接请求。一旦连接建立成功，客户端和服务器端就可以相互发送消息。

在Java中，使用Socket类创建Socket对象来建立连接。服务器端通过ServerSocket类监听指定端口，接受作为客户端的连接请求，并为每个连接创建一个新线程进行处理。使用Socket类连接到服务器，并通过输入输出流来进行数据读写操作。

通过Java的Socket类实现服务器和客户端通信，涉及Socket连接的建立、输入输出流的数据传输、线程处理等多个步骤。这种通信方式非常灵活高效，适用于各种网络应用场景，为客户端和服务器端之间的实时数据交换提供了方便的解决方案。

七、源代码

Information.java

package week16;

import java.io.Serializable;

public class Information implements Serializable {

private String message;

public Information(String message) {

this.message = message;

}

public String getMessage() {

return message;

}

}

ServerFrame.java

package week16;

import javax.swing.\*;

import java.io.IOException;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.net.ServerSocket;

import java.net.Socket;

public class ServerFrame extends JFrame {

private JTextArea messageArea;

private JTextField inputField;

private JButton sendButton;

private ObjectOutputStream outputStream;

public ServerFrame() {}

public ServerFrame(ObjectOutputStream outputStream) {

this.outputStream = outputStream;

setTitle("Server");

setSize(400, 400);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

messageArea = new JTextArea(20, 30);

JScrollPane scrollPane = new JScrollPane(messageArea);

inputField = new JTextField(30);

sendButton = new JButton("Send");

JPanel panel = new JPanel();

panel.add(scrollPane);

panel.add(inputField);

panel.add(sendButton);

sendButton.addActionListener(e -> {

String message = inputField.getText();

showMessage("Server send: " + message);

try {

outputStream.writeObject(new Information(message));

outputStream.flush();

} catch (Exception ex) {

ex.printStackTrace();

}

inputField.setText("");

});

add(panel);

}

public void showMessage(String message) {

messageArea.append(message + "\n");

}

public static void main(String[] args) {

try {

ServerSocket serverSocket = new ServerSocket(12345);

while (true) {

Socket socket = serverSocket.accept();

Thread clientThread = new Thread(new ClientHandler(socket));

clientThread.start();

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

ClientFrame.java

package week16;

import javax.swing.\*;

import java.io.IOException;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.net.Socket;

public class ClientFrame extends JFrame {

private JTextArea messageArea;

private JTextField inputField;

private JButton sendButton;

private ObjectOutputStream outputStream;

public ClientFrame(ObjectOutputStream outputStream) {

this.outputStream = outputStream;

setTitle("Client");

setSize(400, 400);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

messageArea = new JTextArea(20, 30);

JScrollPane scrollPane = new JScrollPane(messageArea);

inputField = new JTextField(30);

sendButton = new JButton("Send");

JPanel panel = new JPanel();

panel.add(scrollPane);

panel.add(inputField);

panel.add(sendButton);

sendButton.addActionListener(e -> {

String message = inputField.getText();

showMessage("Client send: " + message);

try {

outputStream.writeObject(new Information(message));

outputStream.flush();

} catch (Exception ex) {

ex.printStackTrace();

}

inputField.setText("");

});

add(panel);

}

public void showMessage(String message) {

messageArea.append(message + "\n");

}

public static void main(String[] args) {

try {

Socket socket = new Socket("localhost", 12345);

System.out.println("Connected to server");

ObjectOutputStream outputStream = new ObjectOutputStream(socket.getOutputStream());

ObjectInputStream inputStream = new ObjectInputStream(socket.getInputStream());

ClientFrame clientFrame = new ClientFrame(outputStream);

clientFrame.setVisible(true);

Thread serverResponseThread = new Thread(() -> {

try {

while (true) {

Information info = (Information) inputStream.readObject();

clientFrame.showMessage("Server send: " + info.getMessage());

}

} catch (IOException | ClassNotFoundException e) {

e.printStackTrace();

}

});

serverResponseThread.start();

} catch (IOException e) {

e.printStackTrace();

}

}

}

ClientHandler.java

package week16;

import java.io.IOException;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.net.Socket;

class ClientHandler implements Runnable {

private Socket socket;

private ObjectInputStream inputStream;

private ObjectOutputStream outputStream;

public ClientHandler(Socket socket) {

this.socket = socket;

try {

inputStream = new ObjectInputStream(socket.getInputStream());

outputStream = new ObjectOutputStream(socket.getOutputStream());

} catch (IOException e) {

e.printStackTrace();

}

}

@Override

public void run() {

try {

ServerFrame serverFrame = new ServerFrame(outputStream);

serverFrame.setVisible(true);

while (true) {

Information info = (Information) inputStream.readObject();

serverFrame.showMessage("Client Send: " + info.getMessage());

outputStream.flush();

}

} catch (IOException | ClassNotFoundException e) {

e.printStackTrace();

} finally {

try {

if (inputStream != null) {

inputStream.close();

}

if (outputStream != null) {

outputStream.close();

}

if (socket != null) {

socket.close();

}

} catch (IOException ex) {

ex.printStackTrace();

}

}

}

}