目录

1 实验目的与要求

1.1 实验目的

1. 掌握 C++、JAVA 或 Python 等集成开发环境编写网络程序的方法
2. 掌握客户/服务器（C/S）应用的工作方式
3. 学习网络中进程之间通信的原理和实现方法
4. 要求本机既是客户端又是服务器端

1.2 实验要求

网络编程是通过使用套接字来达到进程间通信目的的编程，Socket 编程是网络编程的主流工具，Socket API 是实现进程间通信的一种编程设施，也是一种为进程间提供底层抽象的机制，提供了访问下层通信协议的大量系统调用和相应的数据结构。本实验利用 Socket API 编写网络通信程序。

编写一个基于 socket 的简易聊天程序。所编写的程序应具有如下功能：

1. 具有点对点通信功能，任意客户端之间能够发送消息
2. 具有群组通信功能，客户端能够向组内成员同时发送消息，其他组成员不能收到
3. 具有广播功能，客户端能够向所有其他成员广播消息

2 实验原理与实验内容

2.1 实验原理

2.1.1 点对点通信功能

实现网络点对点通讯程序的关键步骤就是实现信息在网络中的发送和接收。数据接收使用的是 Socket，数据发送使用的是 NetworkStream。

实现代码如下：

```
TcpListener tlListen1 = new TcpListener ( 8889 ) ;
  //侦听端口号
  tlListen1.Start ( ) ;
  Socket skSocket = tlListen1.AcceptSocket ( ) ;
  //接受远程计算机的连接请求，并获得用以接收数据的Socket实例
  EndPoint tempRemoteEP = skSocket.RemoteEndPoint  ;
  //获得远程计算机对应的网络远程终结点
  while (  true )
  {
      Byte [] byStream = new Byte[80] ;
      //定义从远程计算机接收到数据存放的数据缓冲区
      int i = skSocket.ReceiveFrom  ( byStream   , ref
          tempRemoteEP  ) ;
      //接收数据，并存放到定义的缓冲区中
      string sMessage = System.Text.Encoding.UTF8.GetString
          ( byStream  ) ;
      //以指定的编码，从缓冲区中解析出内容
      MessageBox.Show ( sMessage ) ;
      //显示传送来的数据
  }
```

利用 Socket 来接收信息

```
TcpClient tcpc = new TcpClient (  "10.138.198.213"  , 8888
    ) ;
 //对IP地址为“10.138.198.213”的计算机的8888端口提出连接申请
NetworkStream tcpStream = tcpc.GetStream ( ) ;
//如果连接申请建立，则获得用以传送数据的数据流
string sMsg = "您好，见到您很高兴" ;
StreamWriter reqStreamW = new StreamWriter ( tcpStream  ) ;
//以特定的编码往向数据流中写入数据 ,默认为UTF8编码
reqStreamW.Write ( sMsg  ) ;
//将字符串写入数据流中
reqStreamW.Flush ( ) ;
```

```
//清理当前编写器的所有缓冲区，并使所有缓冲数据写入基础流
```

利用 NetworkStream 来发送信息

2.1.2 群组通信功能

组播编程需要 UDP，有两个类支持组播网络编程 Socket 和 UdpClient。一台计算机要加入某一个组，然后接收发往这个组的信息。Socket 类要调用 SetSocketOption 函数加入和离开某一个组。UdpClient 类有直接的加入和离开某个组的成员函数可以调用。而向某个组发信息，则没有什么特殊的，只需把发送数据的目的地址设为组播地址就可以了。

实现代码如下：

```
1     Socket s = new Socket(AddressFamily.InterNetwork, SocketType
          .Dgram, ProtocolType.Udp);
      IPEndPoint iep = new IPEndPoint(IPAddress.Parse("224.0.0.1")
          , 3000);
3     EndPoint ep = (EndPoint)iep;
byte[] b = Encoding.ASCII.GetBytes("just a test!");
5     s.SendTo(b, ep);
      s.Close();
```

发送端

```
      Socket s = new Socket(AddressFamily.InterNetwork, SocketType
          .Dgram, ProtocolType.Udp);
2     IPEndPoint iep = new IPEndPoint(IPAddress.Any, 3000);
      EndPoint ep=(EndPoint)iep;
4     s.Bind(iep);
      s.SetSocketOption(SocketOptionLevel.IP,SocketOptionName.
          AddMembership,new MulticastOption(IPAddress.Parse("
          224.0.0.1")));
6     byte[]b=new byte[1024];
      s.ReceiveFrom(b,ref ep);
8     string test;
```

```
    test = System.Text.Encoding.ASCII.GetString(b);
10  Console.WriteLine(test);
    s.Close();
12  Console.ReadKey();
```

接收端

2.1.3 广播功能

此功能和组播功能实现类似，只要在发送端获得子网中 IP 广播地址发送即可。

实现代码如下：

```
    // 广播模式(自动获得子网中的IP广播地址)
2   broadcastIpEndPoint = new IPEndPoint(IPAddress.Broadcast,
        3000);
```

广播

2.2 实验内容

编写一个基于 socket 的简易聊天程序。所编写的程序应具有如下功能：

1. 具有点对点通信功能，任意客户端之间能够发送消息
2. 具有群组通信功能，客户端能够向组内成员同时发送消息，其他组成员不能收到
3. 具有广播功能，客户端能够向所有其他成员广播消息

3 实验具体设计实现及结果

3.1 实验具体设计实现

本实验使用 Java 语言编程，将 client 和 server 分别封装成一个类，分别在各自的主函数中调用，通过 socket 进行通信。

3.1.1 ChatServer 服务器端实现

ChatServer 类继承 JFrame 类，实现服务器端的图形化界面，在本实验中仅展示各客户端连接情况以及客户端发送的点对点消息、群组消息、广播消息。

ChatServer 的窗体实现如下：

```
    public ChatServer() {
2       super("聊天室服务器端");

4       chatBox = new JTextArea();
        chatBox.setEditable(false);
6
        JScrollPane jsp = new JScrollPane(chatBox);
8       jsp.setPreferredSize(new Dimension(400, 300));
        chatBox.setFont(new Font("宋体", Font.PLAIN, 12));
10
        setLayout(new BorderLayout());
12      add(jsp, BorderLayout.CENTER);

14      setDefaultCloseOperation((JFrame.DISPOSE_ON_CLOSE));
        pack();
16      setVisible(true);
    }
```

ChatServer 的窗体实现

ChatServer 的 main 函数作为程序入口。在 main 方法中，创建一个

ServerSocket 对象，并绑定到指定的服务器端口和本地地址，在本实验中选取的服务器端口（SERVER_PORT）是 12345

```
ServerSocket serverSocket = new ServerSocket(SERVER_PORT, 0,
    InetAddress.getByName("localhost"));
```

ServerSocket 绑定

然后，在主程序中创建了 ChatServer 对象，用于显示服务器端的图形化界面。之后，通过一个 while 循环，不断接受客户端的连接请求。每当有客户端请求连接时，accept () 方法就会返回一个 Socket 对象，该对象与客户端的 Socket 对象相连，通过该 Socket 对象就可以与客户端进行通信，表示服务器端与客户端之间的通信链路已经建立，并创建了一个新的线程来处理该客户端的请求。

```
SwingUtilities.invokeLater(new Runnable() {
        @Override
        public void run() {
            new ChatServer();
        }
    });

    while (true) {
        //serverSocket.accept()方法是一个阻塞调用，意味着程序会在此处暂停，直到有客户端请求连接才会继续执行。
        Socket clientSocket = serverSocket.accept();

        System.out.println("New client connected in " + clientSocket);
        //创建了一个新的线程clientThread，并将其与客户端的连接clientSocket关联。
        // 通过start()方法启动线程后，线程将执行ClientHandler对象中定义的任务
        Thread clientThread = new Thread(new ClientHandler(clientSocket));
```

```
            clientThread.start();
        }
```

服务器端接受客户端连接

main 函数部分实现完成

定义了 ClientInfo 类用于存储客户端的信息，包括客户端的套接字（Socket），客户端昵称（nickname）和组名（groupName）。

类的具体实现如下：

```
    private static class ClientInfo {
        private Socket socket;//客户端端口
        private String nickname;//客户端昵称
        private String groupName;//客户端组名

        public ClientInfo(Socket socket, String nickname, String
            groupName) {
            this.socket = socket;
            this.nickname = nickname;
            this.groupName = groupName;
        }

        public Socket getSocket() {
            return socket;
        }

        public String getNickname() {
            return nickname;
        }

        public String getGroupName() {
            return groupName;
        }

        public Socket getSocketByNickname(String nickname) {
            for (ClientInfo clientInfo : clientSockets) {
```

```
                if (clientInfo.getNickname().equals(nickname)) {
                    return clientInfo.getSocket();
                }
            }
            return null;
        }
    }
```

ClientInfo 类的实现

在定义了 ClientInfo 类的基础上定义列表存储在线的客户端

定义 ClientHandler 类：ClientHandler 类是一个实现了 Runnable 接口的内部类，用于处理每个用户端的连接和消息收发。

在 ClientHandler 类中定义了服务器窗口追加消息的方法 appendMessage ()，用于将消息追加到服务器窗口的聊天记录中，实现方法如下：

```
    public void appendMessage(String message) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
            chatBox.append(message + "\n");
            }
        });
    }
```

appendMessage 方法的实现

对 run () 进行了重写，首先获取客户端的输入流和输出流，进行客户端的初始化，将客户端信息记录在 clientSockets 列表中，实现代码如下：

```
    Scanner scanner = new Scanner(clientSocket.getInputStream(),
        "UTF-8");//获取客户端的请求
    writer = new PrintWriter(new OutputStreamWriter(clientSocket
        .getOutputStream(), "UTF-8"), true);//向客户端发送信息

    writer.println("欢迎来到JIANG的聊天室！");
```

```
6       writer.println("请输入您的昵称：");
        Name = scanner.nextLine();
8       writer.println("欢迎" + Name + "进入聊天室");

10      System.out.println("client " + clientSocket + ":" + Name);
        appendMessage("client " + clientSocket + ":" + Name);
12
        writer.println("请输入您要加入的群组：");
14      GroupName = scanner.nextLine();
        writer.println("欢迎加入群组 '" + GroupName + "' ");
16
        System.out.println("Client " + clientSocket + " joined group
            :" + GroupName);
18      appendMessage("Client " + clientSocket + " joined group:" +
            GroupName);

20      ClientInfo clientInfo = new ClientInfo(clientSocket, Name,
            GroupName);
        clientSockets.add(clientInfo);
```

获取输入输出流及初始化

然后进行发送消息的处理，在消息分类时，根据消息的前缀进行分类：

```
1       while (true) {
            String message = scanner.nextLine();
3           if (message.startsWith("@p")) {
                handlePersonToPersonMessage(message);
5           } else if (message.startsWith("@g")) {
                handleGroupMessage(message, GroupName);
7           } else if (message.startsWith("@b")) {
                handleBroadcastMessage(message);
9           } else {
                System.out.println("Received message from " + Name +
                    ":" + message);
11              sendToAllClients("用户昵称" + Name + ":" + message);
```

```
            appendMessage("用户昵称" + Name + ":" + message);
        }
    }
```

消息分类

@*p* 表示点对点发送消息，在进行消息处理的时候，将消息分割成三段，以空格作为分割标志，第一段为消息前缀，第二段为消息接收者的昵称，第三段为消息内容；

```
    private void handlePersonToPersonMessage(String message) {
        String[] parts = message.split(" ", 3);//将消息切成三
            片，分别是报头、接收者及消息内容
        if (parts.length == 3) {
            String receiver = parts[1];
            String p2pMessage = parts[2];

            System.out.println("Received Person-to-Person
                message from " + Name + " to " + receiver + ":" +
                 p2pMessage);
            sendPersonToPerson(receiver, "用户昵称 " + Name + "
                （私聊消息）: " + p2pMessage);
            sendPersonToPerson(Name, "用户昵称 " + Name + "（私
                聊消息）: " + p2pMessage);
            appendMessage("用户昵称 " + Name + "（私聊:" +
                receiver + "）: " + p2pMessage);
        }
    }
```

处理点对点消息

```
    private void sendPersonToPerson(String targetName, String
        message) {
        for (ClientInfo clientInfo : clientSockets) {
            if (clientInfo.getNickname().equals(targetName)) {
                try {
```

```
                    PrintWriter socketWriter = new PrintWriter(
                        new OutputStreamWriter(clientInfo.
                        getSocket().getOutputStream(), "UTF-8"),
                        true);
                    socketWriter.println(message);
                    break;
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
```

发送点对点消息

@g 表示群组发送消息，在进行消息处理的时候，将消息分割成两段，以空格作为分割标志，第一段为消息前缀，第二段为消息内容；

```
    private void handleGroupMessage(String message, String
        groupName) {
        String[] parts = message.split(" ", 2);
        if (parts.length == 2) {
            String groupMessage = parts[1];

            System.out.println("Received group message from " +
                Name + "(group:" + groupName + "):" +
                groupMessage);
            sendToGroup(groupName, "用户昵称" + Name + "（群组消
                息）：" + groupMessage);
            appendMessage("用户昵称" + Name + "（群组消息）：" +
                groupMessage);
        }
    }
```

处理群组消息

```
    private void sendToGroup(String groupName, String message) {
```

```
        for (ClientInfo clientInfo : clientSockets) {
            if (clientInfo.getGroupName().equals(groupName)) {
                try {
                    PrintWriter socketWriter = new PrintWriter(
                        new OutputStreamWriter(clientInfo.
                        getSocket().getOutputStream(), "UTF-8"),
                        true);
                    socketWriter.println(message);
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
```

发送群组消息

@*b* 表示广播发送消息，在进行消息处理的时候，将消息分割成两段，以空格作为分割标志，第一段为消息前缀，第二段为消息内容；

```
    private void handleBroadcastMessage(String message) {
        String[] parts = message.split(" ", 2);
        if (parts.length == 2) {
            String broadcastMessage = parts[1];

            System.out.println("Received broadcast message from
                " + Name + ":" + broadcastMessage);
            sendToAllClients("用户昵称 " + Name + "（广播消息）:
                " + broadcastMessage);
            appendMessage("用户昵称 " + Name + "（广播消息）:" +
                 broadcastMessage);
        }
    }
```

处理广播消息

```
    private void sendToAllClients(String message) {
```

```
        for (ClientInfo clientInfo : clientSockets) {
            try {//向所有客户端输出消息
                //通过clientInfo.getSocket().getOutputStream()获取客户端的输出流，
                //使用OutputStreamWriter和指定的字符编码（UTF-8）创建PrintWriter对象socketWriter，用于向客户端发送消息。
                //使用socketWriter.println(message)将message发送给客户端
                PrintWriter socketWriter = new PrintWriter(new OutputStreamWriter(clientInfo.getSocket().getOutputStream(), "UTF-8"), true);
                socketWriter.println(message);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
```

发送广播消息

在客户端断开连接的时候，将该客户端信息移除 clientSockets，并进行广播，告知所有用户该用户已经下线，实现代码如下：

```
    clientSockets.remove(clientSocket);
    System.out.println("Client " + clientSocket + " disconnected");
    sendToAllClients("Client " + clientSocket + " disconnected");
    appendMessage("Client " + clientSocket + " disconnected");
```

下线操作

至此，ChatServer 类的主体功能已经实现

3.1.2 ChatClientGUI 客户端实现

ChatClientGUI 类继承 JFrame 类，实现客户端的图形化界面，在本实验中仅展示客户端发送的点对点消息、群组消息、广播消息。

ChatClientGUI 的窗体实现如下：

```
    public ChatClientGUI() {
2       super("聊天室客户端");

4       chatBox = new JTextArea();
        chatBox.setEditable(false);//设置为不可编辑
6
        JScrollPane jsp = new JScrollPane(chatBox);
8       jsp.setPreferredSize(new Dimension(400, 300));
        chatBox.setFont(new Font("宋体", Font.PLAIN, 12));
10
        messageField = new JTextField();
12      messageField.addActionListener(new ActionListener() {
            @Override
14          public void actionPerformed(ActionEvent e) {
                sendMessage(messageField.getText());
16          }
        });
18
        JButton send = new JButton("发送信息");
20      send.addActionListener(new ActionListener() {
            @Override
22          public void actionPerformed(ActionEvent e) {
                sendMessage(messageField.getText());
24          }
        });
26
        JPanel input = new JPanel();
28      input.setLayout(new BorderLayout());
        input.add(messageField, BorderLayout.CENTER);
30      input.add(send, BorderLayout.EAST);
```

```
32          setLayout(new BorderLayout());
            add(jsp, BorderLayout.CENTER);
34          add(input, BorderLayout.SOUTH);

36          setDefaultCloseOperation((JFrame.DISPOSE_ON_CLOSE));
            pack();
38          setVisible(true);

40          connectToServer();

42      }
```

ChatClientGUI 窗体实现

在窗体实现过程中，添加了消息输入框、发送按钮的活动侦听，分别用于消息获取和发送。在构造函数中，还调用了 connectToServer () 方法，用于连接服务器端。

connectToServer () 方法用于连接服务器端。通过 socket 连接服务器端，获取服务器端的输入输出流，用于消息的收发。

```
    private void connectToServer() {
2       try {
            //通过socket连接服务器端口
4           socket = new Socket(SERVER_HOST, SERVER_PORT);
            writer = new PrintWriter(new OutputStreamWriter(
                socket.getOutputStream(), "UTF-8"), true);
6           scanner = new Scanner(new InputStreamReader(socket.
                getInputStream(), "UTF-8"));

8           //初始欢迎
            String welcomeMessage = scanner.nextLine();
10          appendMessage(welcomeMessage);

12          Thread messageListener = new Thread(new
                MessageListener());//创建一个新的线程来监听消息
```

```
            messageListener.start();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

connectToServer 方法实现

在 connectToServer () 方法中，通过创建一个新的线程，将 MessageListener 的一个实例放入线程来监听消息。

其中 MessageListener 类是一个实现了 Runnable 接口的内部类，重写了 run () 方法，用于监听消息。在 run () 方法中，通过 scanner 获取服务器端的消息，然后将消息追加到客户端的聊天记录中。

```
    private class MessageListener implements Runnable {
        @Override
        //使用一个无限循环来持续监听消息。
        // 通过scanner.nextLine()从输入流中读取下一行消息，并将该消息传递给appendMessage方法，以将其追加到聊天区域。
        public void run() {
            try {
                while (true) {
                    String message = scanner.nextLine();
                    appendMessage(message);
                }
            } catch (Exception e) {
                e.printStackTrace();
            } finally {
                try {
                    socket.close();
                    ;
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
```

```
23      }
```

MessageListener 类的实现

在 main 函数中，创建了 ChatClientGUI 对象，用于显示客户端的图形化界面。

```
    public static void main(String[] args) {
2       SwingUtilities.invokeLater(new Runnable() {
            @Override
4           public void run() {
                new ChatClientGUI();
6           }
        });
8   }
```

main 函数

至此，ChatClientGUI 类的主体功能已经实现

3.2 程序流程图

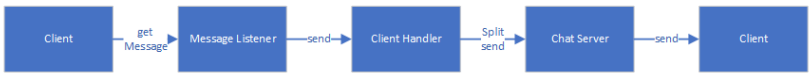


图 1: 程序流程图

3.3 实验结果

3.3.1 启动服务器端

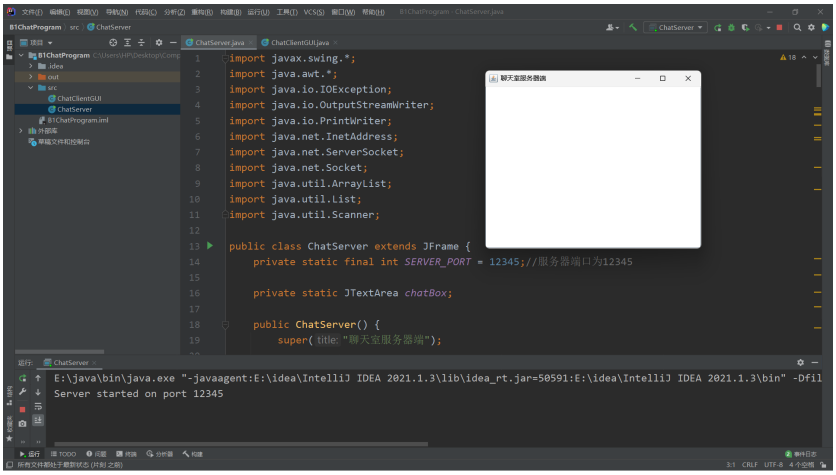
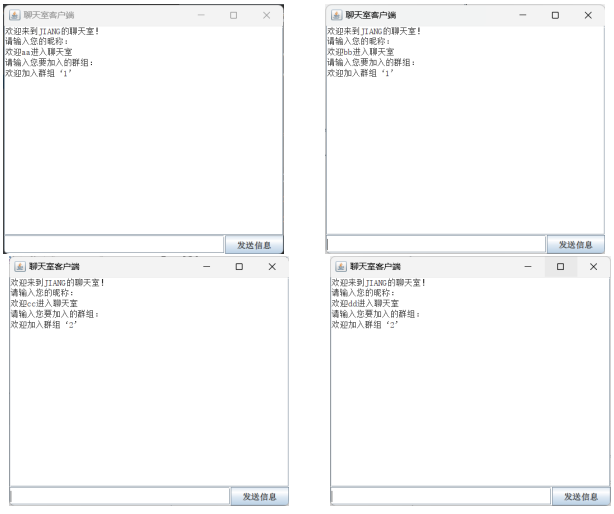


图 2: 启动服务器端

3.3.2 初始化客户端



3.3.3 服务器监听 1

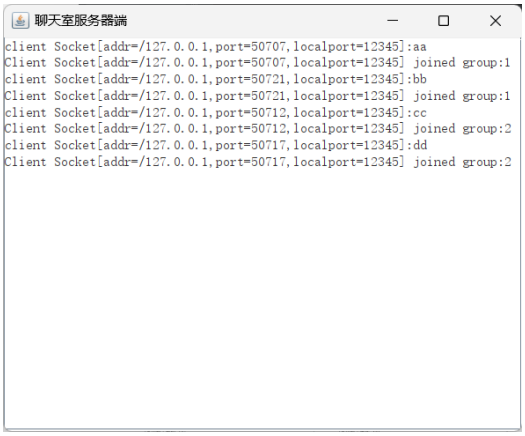
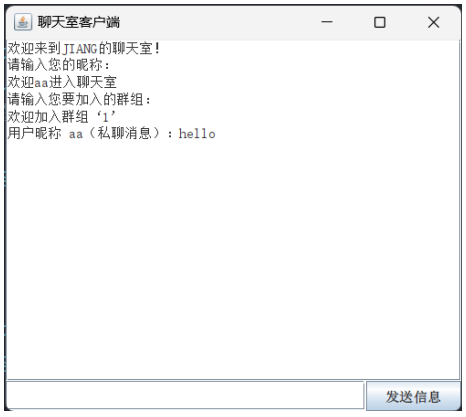
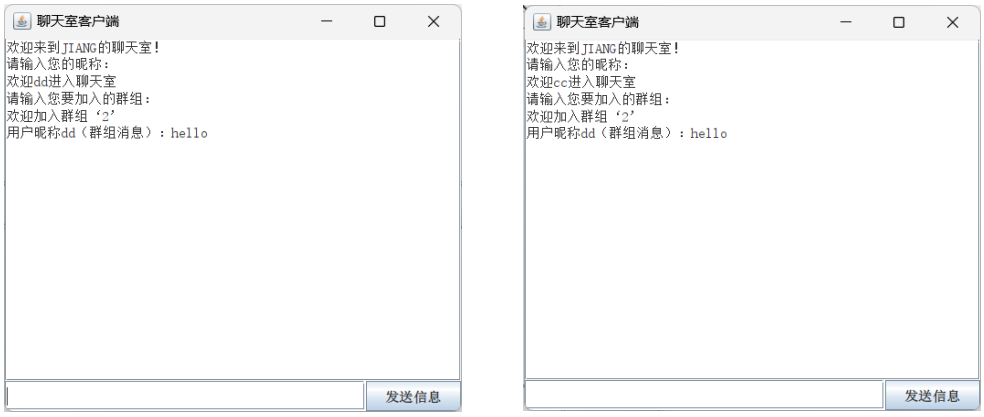


图 3: 服务器监听 1

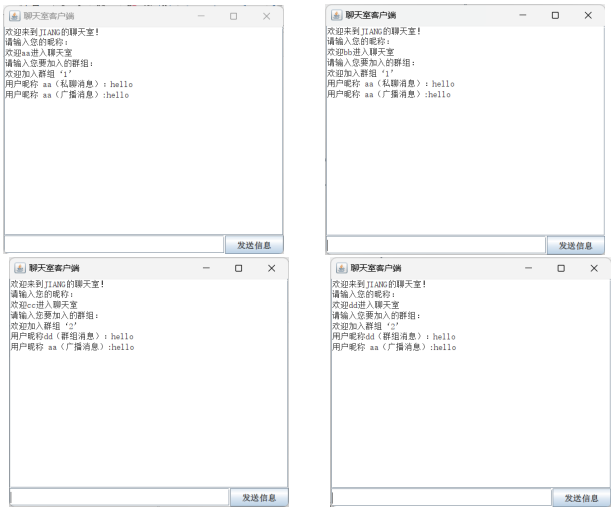
3.3.4 点对点通信



3.3.5 群组通信



3.3.6 广播通信



3.3.7 服务器监听 2

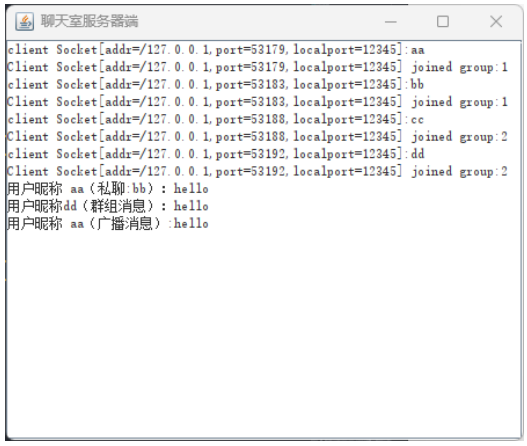


图 4: 服务器监听 2

4 实验设备与实验环境

1. 编程语言：Java

2. 编程环境：IDEA，windows11 操作系统

5 实验总结

本次实验是实现一个简单的聊天系统，包括服务器和客户端程序的开发。通过实验，我掌握了以下内容：

Socket 编程：学习了如何使用 Socket 建立客户端与服务器之间的通信连接，并进行数据的发送和接收。

GUI 设计：使用 Swing 库创建了简单的用户界面，包括聊天记录显示区域和消息输入框，提供了基本的交互功能。

多线程编程：为了实现同时处理多个客户端的连接和消息交互，使用了多线程编程。每个客户端连接都在独立的线程中进行处理，避免了阻塞其他客户端的情况。

在实验过程中，遇到了一些困难和挑战：

网络通信理解：对于网络通信的概念和原理需要有一定的了解，包括服务器端的监听和客户端的连接，以及数据的发送和接收。

多线程同步：在处理多个客户端连接和消息交互时，需要注意线程之间的同步和互斥，避免数据竞争和不一致的情况。

GUI 设计和事件处理：创建用户界面并处理用户的输入和操作需要熟悉 Swing 库的使用，并理解事件驱动的编程模型。

网络编程是实现分布式系统和实时通信的关键技术之一，了解并掌握 Socket 编程对于开发网络应用程序至关重要。多线程编程可以提高程序的并发性和响应性，但也带来了线程安全性的考虑，需要注意同步和互斥机制的使用。GUI 设计需要结合用户需求和交互模式，简洁明了的界面和良好的用户体验是提升应用程序质量的重要方面。在开发实时通信系统时，需要考虑网络延迟和消息丢失等因素，设计合适的机制来保证消息的可靠性和实时性。

通过实验，我加深了对网络通信和多线程编程的理解，并提升了问题解决能力和编程技巧。本次实验让我在实践中掌握了聊天系统的基本原理和实现方式，同时也面对了一些挑战和难题。通过不断的学习和实践，我逐渐提升了对网络编程和多线程编程的理解和应用能力。

6 附录

6.1 ChatServer.java 源代码

```
import javax.swing.*;
import java.awt.*;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class ChatServer extends JFrame {
    private static final int SERVER_PORT = 12345;//服务器端口为12345

    private static JTextArea chatBox;

    public ChatServer() {
        super("聊天室服务器端");

        chatBox = new JTextArea();
        chatBox.setEditable(false);

        JScrollPane jsp = new JScrollPane(chatBox);
        jsp.setPreferredSize(new Dimension(400, 300));
        chatBox.setFont(new Font("宋体", Font.PLAIN, 12));

        setLayout(new BorderLayout());
        add(jsp, BorderLayout.CENTER);

```

```
            setDefaultCloseOperation((JFrame.DISPOSE_ON_CLOSE));
32          pack();
            setVisible(true);
34      }

36      private static class ClientInfo {
            private Socket socket;//客户端端口
38          private String nickname;//客户端昵称
            private String groupName;//客户端组名
40
            public ClientInfo(Socket socket, String nickname,
                String groupName) {
42              this.socket = socket;
                this.nickname = nickname;
44              this.groupName = groupName;
            }
46
            public Socket getSocket() {
48              return socket;
            }
50
            public String getNickname() {
52              return nickname;
            }
54
            public String getGroupName() {
56              return groupName;
            }
58
            public Socket getSocketByNickname(String nickname) {
60              for (ClientInfo clientInfo : clientSockets) {
                    if (clientInfo.getNickname().equals(nickname
                        )) {
62                      return clientInfo.getSocket();
                    }
```

```
64                  }
                    return null;
66              }
            }
68
            private static List<ClientInfo> clientSockets = new
                ArrayList<>();
70
            private static class ClientHandler implements Runnable {
72              private Socket clientSocket;
                private PrintWriter writer;
74
                public String Name;
76              public String GroupName;

78              public ClientHandler(Socket clientSocket) {
                    this.clientSocket = clientSocket;
80              }

82              public void appendMessage(String message) {
                    SwingUtilities.invokeLater(new Runnable() {
84                      @Override
                        public void run() {
86                          chatBox.append(message + "\n");
                        }
88                  });
                }
90
                @Override
92              public void run() {
                    try {
94                      Scanner scanner = new Scanner(clientSocket.
                            getInputStream(), "UTF-8");//获取客户端的
                            请求
                        writer = new PrintWriter(new
```

```
        OutputStreamWriter(clientSocket.
        getOutputStream(), "UTF-8"), true);//向客
        户端发送信息
96
                writer.println("欢迎来到JIANG的聊天室！");
98
                writer.println("请输入您的昵称：");
100             Name = scanner.nextLine();
                writer.println("欢迎" + Name + "进入聊天室")
                    ;
102
                System.out.println("client " + clientSocket
                    + ":" + Name);
104             appendMessage("client " + clientSocket + ":"
                     + Name);

106             writer.println("请输入您要加入的群组：");
                GroupName = scanner.nextLine();
108             writer.println("欢迎加入群组 '" + GroupName
                    + "' ");

110             System.out.println("Client " + clientSocket
                    + " joined group:" + GroupName);
                appendMessage("Client " + clientSocket + "
                    joined group:" + GroupName);
112
                ClientInfo clientInfo = new ClientInfo(
                    clientSocket, Name, GroupName);
114             clientSockets.add(clientInfo);

116             //进行消息发送方式的分离
                while (true) {
118                 String message = scanner.nextLine();
                    if (message.startsWith("@p")) {
120                     handlePersonToPersonMessage(message)
```

```
                    ;
                } else if (message.startsWith("@g")) {
                    handleGroupMessage(message,
                        GroupName);
                } else if (message.startsWith("@b")) {
                    handleBroadcastMessage(message);
                } else {
                    System.out.println("Received message
                        from " + Name + ":" + message);
                    sendToAllClients("用户昵称" + Name +
                        ":" + message);
                    appendMessage("用户昵称" + Name + ":
                        " + message);
                }
            }
        } catch (IOException e) {//捕获输入输出操作过程
            中可能发生的异常情况，例如文件读写错误、网络
            连接问题
            e.printStackTrace();
        } finally {
            clientSockets.remove(clientSocket);
            System.out.println("Client " + clientSocket
                + " disconnected");
            sendToAllClients("Client " + clientSocket +
                " disconnected");
        }
    }

    private void handlePersonToPersonMessage(String
        message) {
        String[] parts = message.split(" ", 3);//将消息
            切成三片，分别是报头、接收者及消息内容
        if (parts.length == 3) {
            String receiver = parts[1];
            String p2pMessage = parts[2];
```

```
146                 System.out.println("Received Person-to-
                        Person message from " + Name + " to " +
                        receiver + ":" + p2pMessage);
                    sendPersonToPerson(receiver, "用户昵称 " +
                        Name + "（私聊消息）: " + p2pMessage);
148                 sendPersonToPerson(Name, "用户昵称 " + Name
                        + "（私聊消息）: " + p2pMessage);
                    appendMessage("用户昵称 " + Name + "（私聊:"
                         + receiver + "）: " + p2pMessage);
150             }
            }
152
            private void handleGroupMessage(String message,
                String groupName) {
                String[] parts = message.split(" ", 2);
154             if (parts.length == 2) {
                    String groupMessage = parts[1];

158                 System.out.println("Received group message
                        from " + Name + "(group:" + groupName + "
                        ):" + groupMessage);
                    sendToGroup(groupName, "用户昵称" + Name + "
                         （群组消息）: " + groupMessage);
160                 appendMessage("用户昵称" + Name + "（群组消
                        息）: " + groupMessage);
                }
162         }

164         private void handleBroadcastMessage(String message)
                {
                String[] parts = message.split(" ", 2);
166             if (parts.length == 2) {
                    String broadcastMessage = parts[1];
168
```

```
                System.out.println("Received broadcast
                    message from " + Name + ":" +
                    broadcastMessage);
170             sendToAllClients("用户昵称 " + Name + "（广
                    播消息）:" + broadcastMessage);
                appendMessage("用户昵称 " + Name + "（广播消
                    息）:" + broadcastMessage);
172         }
        }
174
        private void sendToAllClients(String message) {
176         for (ClientInfo clientInfo : clientSockets) {
                try {//向所有客户端输出消息
178                 //通过clientInfo.getSocket().
                        getOutputStream()获取客户端的输出流，
                    //使用OutputStreamWriter和指定的字符编码
                        （UTF-8）创建PrintWriter对象
                        socketWriter，用于向客户端发送消息。
180                 //使用socketWriter.println(message)将
                        message发送给客户端
                    PrintWriter socketWriter = new
                        PrintWriter(new OutputStreamWriter(
                        clientInfo.getSocket().
                        getOutputStream(), "UTF-8"), true);
182                 socketWriter.println(message);
                } catch (IOException e) {
184                 e.printStackTrace();
                }
186         }
        }
188
        private void sendToGroup(String groupName, String
            message) {
190         for (ClientInfo clientInfo : clientSockets) {
                if (clientInfo.getGroupName().equals(
```

```
                    groupName)) {
192                 try {
                        PrintWriter socketWriter = new
                            PrintWriter(new
                            OutputStreamWriter(clientInfo.
                            getSocket().getOutputStream(), "
                            UTF-8"), true);
194                     socketWriter.println(message);
                    } catch (IOException e) {
196                     e.printStackTrace();
                    }
198             }
            }
200     }

202     private void sendPersonToPerson(String targetName,
            String message) {
            for (ClientInfo clientInfo : clientSockets) {
204             if (clientInfo.getNickname().equals(
                    targetName)) {
                    try {
206                     PrintWriter socketWriter = new
                            PrintWriter(new
                            OutputStreamWriter(clientInfo.
                            getSocket().getOutputStream(), "
                            UTF-8"), true);
                        socketWriter.println(message);
208                     break;
                    } catch (IOException e) {
210                     e.printStackTrace();
                    }
212             }
            }
214     }
}
```

```
216
        public static void main(String[] args) {
218         SwingUtilities.invokeLater(new Runnable() {
                @Override
220             public void run() {
                    new ChatServer();
222             }
            });
224
            try {
226             //创建一个ServerSocket对象，将其绑定到指定的服务
                    器端口和本地地址
                //ServerSocket是java实现服务器端的套接字，监听指
                    定的端口，接收客户端连接请求，并与客户端进行
                    通信
228             //SERVER_PORT是服务器绑定端口，为12345
                //backlog设为0表示使用默认值，即系统根据具体实现
                    来选择一个合适的默认队列长度
230             //InetAddress.getByName("localhost")获取本机IP
                ServerSocket serverSocket = new ServerSocket(
                    SERVER_PORT, 0, InetAddress.getByName("
                    localhost"));
232
                System.out.println("Server started on port " +
                    SERVER_PORT);
234
                while (true) {
236                 //serverSocket.accept()方法是一个阻塞调用，
                        意味着程序会在此处暂停，直到有客户端请求
                        连接才会继续执行。
                    Socket clientSocket = serverSocket.accept();
238
                    System.out.println("New client connected in
                        " + clientSocket);
240                 //创建了一个新的线程clientThread，并将其与客
```

```
                    户端的连接clientSocket关联。
                // 通过start()方法启动线程后，线程将执行
                    ClientHandler对象中定义的任务
242             Thread clientThread = new Thread(new
                    ClientHandler(clientSocket));
                clientThread.start();
244         }
        } catch (IOException e) {
246         e.printStackTrace();
        }
248 }
}
```

ChatServer.java 源代码

6.2 ChatClientGUI.java 源代码

```
1   import javax.swing.*;
    import java.awt.*;
3   import java.awt.event.ActionEvent;
    import java.awt.event.ActionListener;
5   import java.io.*;
    import java.net.*;
7   import java.util.*;

9   public class ChatClientGUI extends JFrame {
        private static final String SERVER_HOST = "localhost";//
            服务器地址
11      private static final int SERVER_PORT = 12345;//服务器端
            口

13      private Socket socket;//客户端套接字
        private PrintWriter writer;//输出流
15      private Scanner scanner;
```

```
17          private JTextArea chatBox;//聊天框
            private JTextField messageField;//消息输入框
19
            public ChatClientGUI() {
21              super("聊天室客户端");

23              chatBox = new JTextArea();
                chatBox.setEditable(false);//设置为不可编辑
25
                JScrollPane jsp = new JScrollPane(chatBox);
27              jsp.setPreferredSize(new Dimension(400, 300));
                chatBox.setFont(new Font("宋体", Font.PLAIN, 12));
29
                messageField = new JTextField();
31              messageField.addActionListener(new ActionListener()
                    {
                    @Override
33                  public void actionPerformed(ActionEvent e) {
                        sendMessage(messageField.getText());
35                  }
                });
37
                JButton send = new JButton("发送信息");
39              send.addActionListener(new ActionListener() {
                    @Override
41                  public void actionPerformed(ActionEvent e) {
                        sendMessage(messageField.getText());
43                  }
                });
45
                JPanel input = new JPanel();
47              input.setLayout(new BorderLayout());
                input.add(messageField, BorderLayout.CENTER);
49              input.add(send, BorderLayout.EAST);
```

```
51              setLayout(new BorderLayout());
                add(jsp, BorderLayout.CENTER);
53              add(input, BorderLayout.SOUTH);

55              setDefaultCloseOperation((JFrame.DISPOSE_ON_CLOSE));
                pack();
57              setVisible(true);

59              connectToServer();

61          }

63          private void connectToServer() {
                try {
65                  //通过socket连接服务器端口
                    socket = new Socket(SERVER_HOST, SERVER_PORT);
67                  writer = new PrintWriter(new OutputStreamWriter(
                        socket.getOutputStream(), "UTF-8"), true);
                    scanner = new Scanner(new InputStreamReader(
                        socket.getInputStream(), "UTF-8"));
69
                    //初始欢迎
71                  String welcomeMessage = scanner.nextLine();
                    appendMessage(welcomeMessage);
73
                    Thread messageListener = new Thread(new
                        MessageListener());//创建一个新的线程来监听消
                        息
75                  messageListener.start();
                } catch (IOException e) {
77                  e.printStackTrace();
                }
79          }

81          private void sendMessage(String message) {//发送消息并清
```

```
            空输入框
            writer.println(message);
            ;
            messageField.setText("");
        }

        private void appendMessage(String message) {//在消息框中
            进行消息追加
            SwingUtilities.invokeLater(new Runnable() {
                @Override
                public void run() {
                    chatBox.append(message + "\n");
                }
            });
        }


        private class MessageListener implements Runnable {
            @Override
            //使用一个无限循环来持续监听消息。
            // 通过scanner.nextLine()从输入流中读取下一行消息，
                并将该消息传递给appendMessage方法，以将其追加到聊
                天区域。
            public void run() {
                try {
                    while (true) {
                        String message = scanner.nextLine();
                        appendMessage(message);
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                } finally {
                    try {
                        socket.close();
                        ;
```

```
113                     } catch (IOException e) {
                            e.printStackTrace();
115                     }
                    }
117             }

119         }

121         public static void main(String[] args) {
                SwingUtilities.invokeLater(new Runnable() {
123                 @Override
                    public void run() {
125                     new ChatClientGUI();
                    }
127             });
            }
129     }
```

ChatClientGUI.java 源代码