目录

[一、需求分析 1](#_Toc454350388)

[1.1 设计需求 1](#_Toc454350389)

[1.2 系统设计分析 1](#_Toc454350390)

[1.3 系统功能分析 2](#_Toc454350391)

[二、模块概要分析 3](#_Toc454350392)

[2.1 模块划分 3](#_Toc454350393)

[2.2 模块结构图 3](#_Toc454350394)

[2.3 模块功能 3](#_Toc454350395)

[三、模块详细设计 4](#_Toc454350396)

[3.1 客户端实现 4](#_Toc454350397)

[3.2 服务器实现 7](#_Toc454350398)

[3.3 公共类实现 11](#_Toc454350399)

[四、运行结果 13](#_Toc454350400)

[4.1 客户端进入聊天室 13](#_Toc454350401)

[4.2 聊天界面 13](#_Toc454350402)

[4.3 选择聊天类型 14](#_Toc454350403)

[4.4 人数动态刷新 17](#_Toc454350404)

[五、课程设计总结 18](#_Toc454350405)

[六、附录 19](#_Toc454350406)

[6.1 参考文献 19](#_Toc454350407)

[6.2 源代码 19](#_Toc454350408)

## 一、需求分析

### 1.1 设计需求

1．采用C/S模式，基于TCP协议编程的方式，使得各个用户通过服务器转发实现聊天的功能。

2．分为三大模块：客户端模块、服务器端模块和公共辅助类模块。

3. 客户端模块的主要功能：

1. 登陆功能：用户可以注册，然后选择服务器登入聊天室。
2. 显示用户：将在线用户显示在列表中。
3. 接收信息：能接收其他用户发出的信息。
4. 发送信息：能发出用户要发出的信息。

4. 服务器端模块的主要功能：

1) 检验登陆信息：检查登陆信息是否正确，并向客户端返回登陆信息，如信息正确。就允许用户登陆。

2) 显示在线状态：将该用户的状态发给各在线用户。

3) 转发聊天信息：将消息转发给所有在线的用户。

5. 公共辅助类模块的主要功能：

1) 定义完整的消息传递机制

2) 对消息转发的方式进行有效约束

3) 规定消息类型

### 1.2 系统设计分析

一．实现网络聊天室编程，关键在于Socket通信，程序的功能都是在Socket的基础上一层一层增加的。实现Socket通信的基本方法为以下4个步骤。

1. 客户端与服务器端分别实例化ServerSockot/Socket。
2. 打开连接到Socket的面向对象输入输出流。
3. 利用输入输出流按照TCP协议对Socket进行读写操作。
4. 关闭输入输出流和Socket。

我们要实现的功能都是在第3步对Socket的输入输出流做相应的操作：

ServerSocket.accept()

等待连接

Server

Close

OutputStream InputStream

通信过程

Socket(ip，port)

与服务器建立连接

Close

OutputStream InputStream

Client

二．涉及到多客户端并发访问，必须用线程进行控制，不同的处理线程为不同的客户服务，主线程只负责循环等待，处理线程负责网络连接，接受客户输入的信息，根据消息类型对消息转发。

### 1.3 系统功能分析

本系统基于TCP协议，采用客户端服务器模式，客户端可以并发访问服务器。采用了多线程技术，可以做到多人同时在线，提高了系统的并发能力，增强了系统的实用性。有简单的客户端界面，方便用户使用。采用的是TCP协议，这是可靠的，稳定的协议，保证了消息传递的稳定性和一致性。

本系统实现的功能有：

1. 客户端输入用户名登录
2. 服务端检验登陆信息
3. 客户端之间群聊
4. 客户端之间私聊
5. 动态刷新好友列表
6. 显示在线人数
7. 正确处理客户端退出
8. 客户端上下线提醒
9. 客户端文件透明传输

## 二、模块概要分析

### 2.1 模块划分

客户端：

Client类模块，ClientUI界面模块，ClientMsgType消息处理模块

服务器：

Server类模块，ServerHandler模块，ServerMsgType消息处理模块

公共类：

Message消息结构类，Common公共函数类，Config系统配置类

### 2.2 模块结构图

Server模块

客户端线程

ServerHandle模块

ServerHandle模块

消息转发

通信过程

建立连接

ServerMsgType模块

ClientUI模块

Client模块

消息展示

消息处理

ClientMsgType模块

### 2.3 模块功能

客户端：

1. Client类模块：与服务端建立连接，完成客户端登录，实现消息的群聊，私聊，用户列表更新，文件转换等功能。
2. ClientUI界面模块：客户端窗口界面，实现用户友好，方便用户使用，在窗口上有消息显示，用户列表，消息类型，在线人数等视图信息。
3. ClientMsgType消息处理模块：定义了常用的消息处理方法。

服务器：

1. Server类模块：启动监听，与客户端建立连接
2. ServerHandler模块：提供线程管理，实现多用户的管理。针对每一个用户发过来的消息，进行相应处理，再转发给客户。
3. ServerMsgType消息处理模块：对客户端发送的消息进行指定操作模式处理，只进行消息转发，涉及一对多和一对一两种模式。

公共类：

1. Message消息结构类：客户端和服务器传输的消息被定义为类，通过对Message对象的序列化传输。

2. Common公共函数类：提供文件序列化的处理以及常用的功能性函数。

3. Config系统配置类：包含连接IP及端口号的配置信息。

## 三、模块详细设计

### 3.1 客户端实现

1. Client类模块：关键模块，通过服务器IP地址与端口建立连接。客户端有套接字和输入输出流。首先输入用户名，然后包装成Message对象发送给服务器，之后进入主循环对来自服务器的消息进行处理。

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

**try** {

// 获取用户名

**while** (name == **null** || name.equals(""))

name = JOptionPane.*showInputDialog*("请输入名字").trim();

// 第一次运行，发送登陆消息

send(**new** Message("login", name, **null**, **null**));

// 显示界面

ui = **new** ClientUI(**this**, name + " 的客户端");

// 消息处理主循环

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

Message rcv = (Message) in.readObject();

// 服务器端发来的消息

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

**new** ClientMsgType(**this**, rcv).process();

}

} **catch** (Exception e) {

System.***out***.println(e.toString() + " 客户端退出");

}

}

2. ClientUI类模块：用户界面，所有的界面元素及相关的操作都在里面定义了方法，不论UI界面如何改变都不会影响到外部类。里面含有各种控件极其功能定义，还包括文件传输。主体代码如下：

**public** ClientUI(Client client, String winname) {

**super**(winname); // 继承父类的名字

setSize(600, 400);

**this**.client = client;

// 消息文本显示区域

msgArea = **new** JTextArea(400, 400);

msgArea.setEditable(**false**);

textAreaScrollPane = **new** JScrollPane(msgArea);

add(textAreaScrollPane, BorderLayout.***CENTER***);

// 发送消息区域

textFieldPanel.setLayout(**new** FlowLayout(0, 10, 10));

add(textFieldPanel, BorderLayout.***SOUTH***);

clientList = **new** JComboBox<String>(); // 在线用户列表

clientList.addItem("All");

textFieldPanel.add(clientList);

msgType = **new** JComboBox<String>(); // 聊天文本或者文件

msgType.addItem("chat");

msgType.addItem("file");

textFieldPanel.add(msgType);

msgField = **new** JTextField(20); // 输入消息文本

textFieldPanel.add(msgField);

btn = **new** JButton("发送"); // 发送消息按钮

btn.setMnemonic(KeyEvent.***VK\_ENTER***);

textFieldPanel.add(btn);

cntLabel = **new** JLabel("在线人数:1"); // 显示在线人数

textFieldPanel.add(cntLabel);

// 发送消息按钮监听器

btn.addActionListener(**new** ActionListener() {

@Override

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

String type = (String) msgType.getSelectedItem();

String title = "";

String content = getText().trim();

**byte**[] fbyte = **null**;

**if** (content.equals("")) {

JOptionPane.*showMessageDialog*(client.ui, "输入不能为空");

**return**;

}

**if** (type.equals("file")) {

// 获取客户端发送文件

JFileChooser dlg = **new** JFileChooser();

dlg.setDialogTitle("选择文件");

**int** result = dlg.showOpenDialog(client.ui);

**if** (result == JFileChooser.***APPROVE\_OPTION***) {

File file = dlg.getSelectedFile();

fbyte = Common.*file2Byte*(file);

title = file.getName();

} **else** {

**return**;

}

append("TO " + getName() + ": " + title + "\n");

}

append("TO " + getName() + ": " + content + "\n");

client.send(**new** Message(type, client.name, getName(), title, content, fbyte));

clear();

}

});

addWindowListener(**new** WindowAdapter() {

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

client.close();

System.*exit*(0);

}

});

setVisible(**true**);

}

3.ClientMsgType类模块：定义了客户端处理消息的方法。主体架构和服务区类似。这里我们只介绍处理文件的方法：

/\*\*

\* 文件处理 利用字节数组可以处理所有文件

\*/

**public** **void** file() {

String toAll = "[public]";

**if** (msg.getTo().equals(client.name))

toAll = "[private]";

client.ui.append(toAll + msg.getFrom() + ": " + msg.getContent() + "\n");

**int** confirm = JOptionPane.*showConfirmDialog*(client.ui,

"收到了来自" + msg.getFrom() + "的文件:" + msg.getTitle() + "，需要保存吗？");

**if** (confirm != JOptionPane.***YES\_OPTION***)

**return**;

// 获取保存路径

JFileChooser dlg = **new** JFileChooser();

dlg.setDialogTitle("选择保存路径");

dlg.setSelectedFile(**new** File(msg.getTitle()));

**int** result = dlg.showSaveDialog(client.ui);

**if** (result != JFileChooser.***APPROVE\_OPTION***)

**return**;

File file = dlg.getSelectedFile();

**if** (file.exists()) {

**int** copy = JOptionPane.*showConfirmDialog*(**null**, "是否要覆盖当前文件？", "保存", JOptionPane.***YES\_NO\_OPTION***,

JOptionPane.***QUESTION\_MESSAGE***);

**if** (copy == JOptionPane.***YES\_OPTION***) {

dlg.approveSelection();

}

} **else** {

dlg.approveSelection();

}

**if** (Common.*byte2File*(msg.getFile(), file)) {

JOptionPane.*showMessageDialog*(client.ui, "文件保存成功");

} **else** {

JOptionPane.*showMessageDialog*(client.ui, "文件保存失败");

}

}

### 3.2 服务器实现

1.Server类模块：创建ServerSocket,在指定端口监听，当有用户登录，接受套接字输入流，显示其登录信息，调用ServerHandler线程进行管理。

**public** Server(**int** port) {

**try** {

@SuppressWarnings("resource")

ServerSocket server = **new** ServerSocket(port);

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

Socket conn = server.accept(); // 建立客户端套接字

**new** Thread(**new** ServerHandler(conn)).start();

// 新建线程和客户端建立全双工通信

}

} **catch** (Exception e) {

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

}

}

2.ServerHandler模块：服务端的核心，其中包含客户端线程连接类，套接字及输入输出流的初始化。每个新连接的客户端被初始化为一个ServerHandler对象线程，首先进行的是客户端登陆合法化检验，通过后将客户端添加到服务器线程池并通知所有在线用户。之后采用一个无限循环来和该客户端进行双向通信。

代码中数据结构定义：

**private** ObjectInputStream in; // 输入流

**private** ObjectOutputStream out; // 输出流

**private** String name; // 用户名

**private** **static** ArrayList<ServerHandler> *clientList* = **new** ArrayList<ServerHandler>(); // 把用户线程放入对象数组

**private** **static** HashMap<String, ServerHandler> *clientMap* = **new** HashMap<>(); // 把用户名和连接线程关联便于查找

初始化代码：

**public** ServerHandler(Socket socket) {

**try** {

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

out = **new** ObjectOutputStream(socket.getOutputStream());

} **catch** (IOException e) {

System.***out***.println(e.toString() + " 服务端初始化失败");

}

}

核心代码：

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

**if** (!loginVerify())

**return**; // 登陆校验

**try** {

addClient(); // 客户端经过登陆检验 添加到列表

sendMessage(**new** Message("online", name, "All", "上线了!"));

// 广播用户列表

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

Message rcv = (Message) in.readObject(); // 等待客户端消息

sendMessage(rcv);

}

} **catch** (Exception e) {

System.***out***.println(e.toString() + " 客户端已退出");

} **finally** {

removeClient();

sendMessage(**new** Message("offline", name, "All", "下线了！"));

// 广播下线通知

}

}

登陆校验：

**private** Boolean loginVerify() {

**try** {

Message rcv = (Message) in.readObject();

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

**this**.name = rcv.getFrom();

**if** (*clientMap*.containsKey(name)) {

// 用户名重复

sendToClient(**this**, **new** Message("login", **null**, name, "用户名已存在"));

**return** **false**;

}

} **catch** (Exception e) {

System.***out***.println(e.toString() + "登陆检验失败");

**return** **false**;

}

**return** **true**;

}

消息广播：

把msg消息发送给每一个用户。其中*clientList*是ServerHandler的客户端容器类型，可以指向每一个用户。*clientMap* 是用户名到线程的映射。

**public** **void** sendAllClient(Message msg) {

String name = msg.getFrom();

ServerHandler sh = *clientMap*.get(name);

**synchronized** (*clientList*) {

**for** (ServerHandler client : *clientList*) {

**if** (client != sh) {

sendToClient(client, msg);

}

}

}

}

上线提醒：

sendClientList函数：更新用户列表，当有新用户上线时，对以前的用户的好友列表，加上这个新用户名字，对于这个新用户，加上所有用户的名字。通过将消息包装成Message对象进行数据交互。

**public** **synchronized** **void** sendClientList(Message msg) {

// 新上线的用户

ServerHandler newclient = *clientMap*.get(msg.getFrom());

**for** (ServerHandler client : *clientList*) {

**if** (client == newclient)

**continue**;

// 给新上线用户所有列表

sendToClient(newclient, **new** Message(msg.getType(), client.name, newclient.name, **null**));

// 更新老用户

sendToClient(client, **new** Message(msg.getType(), newclient.name, client.name, msg.getContent()));

}

}

3.ServerMsgType模块：定义了服务端对客户端发送消息的处理方式。消息包装成Message类，经过序列化后在服务器和客户端间传输。服务端根据消息的收发对象和类型进行转发。这里利用到Java的反射机制，根据消息类型获取相应处理函数。

**public** **class** ServerMsgType {

ServerHandler server;

Message msg;

**public** ServerMsgType(ServerHandler server, Message msg) {

**this**.server = server;

**this**.msg = msg;

}

**public** **void** process() {

**try** {

Method method = **this**.getClass().getDeclaredMethod(msg.getType());

method.invoke(**this**);

} **catch** (Exception e) {

System.***out***.println(e.toString() + " 使用了未定义的操作");

**try** {

Method method = **this**.getClass().getDeclaredMethod("chat");

method.invoke(**this**);

} **catch** (Exception e1) {

System.***out***.println(e1.toString() + " 默认处理失败");

}

}

}

**public** **void** chat() {

**if** (msg.getTo().equals("All")) {

// 给所有人发送消息

server.sendAllClient(msg);

} **else** {

// 发送消息给指定的人

server.sendToClient(msg);

}

}

**public** **void** online() {

server.sendClientList(msg);

}

**public** **void** offline() {

server.sendAllClient(msg);

}

**public** **void** login() {

server.sendToClient(msg);

}

}

### 3.3 公共类实现

1. Message模块：底层数据封装的核心。所有的消息全部通过封装成Message从而在服务器和客户端之间收发。主体内容如下：

**private** String type; // 消息类型

**private** String from; // 来源

**private** String to; // 目的

**private** String title; // 附件名

**private** String content; // 消息内容

**private** **byte**[] file; // 附件内容

其中系统已经定义的消息类型包括：

chat:聊天内容、online：客户端上线、offline：客户端下线、login：客户端登陆

每种类型在服务器和客户端的MsgType类中都有定义对应的处理函数。

From字段标识消息发送的来源，为客户端的用户名；

To字段标识消息发送的对象，为客户端的用户名，如果为All则是广播消息；

Title字段当发送文件时有效，为原文件名；

Content字段为消息的内容；

File字段为附件的字节数组。

把以上字段组装成类之后经过序列化就可以在客户端和服务器之间传输。

2. Common模块：定义了文件处理相关的函数，这里我们把文件作为字节数组处理，这样实现了文件的透明化传输。

文件转化为字节数组：

**public** **static** **byte**[] file2Byte(File file) {

**byte**[] buffer = **new** **byte**[***DEFAULT\_BUFFER\_SIZE***];

**int** len = 0;

**try** {

FileInputStream fin = **new** FileInputStream(file);

len = fin.read(buffer);

fin.close();

} **catch** (Exception e) {

System.***out***.println(e.toString() + " 文件无法转化成字节");

}

**byte**[] ret = **new** **byte**[len];

**for** (**int** i = 0; i < len; i++)

ret[i] = buffer[i];

**return** ret;

}

字节数组转化为文件：

**public** **static** **boolean** byte2File(**byte**[] in, File out) {

**try** {

FileOutputStream fout = **new** FileOutputStream(out);

fout.write(in);

fout.close();

} **catch** (Exception e) {

System.***out***.println(e.toString() + " 字节转换失败");

**return** **false**;

}

**return** **true**;

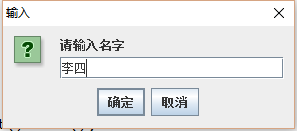
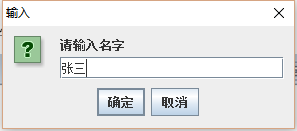
}

3.Config模块：主要包括IP和port的配置。

## 四、运行结果

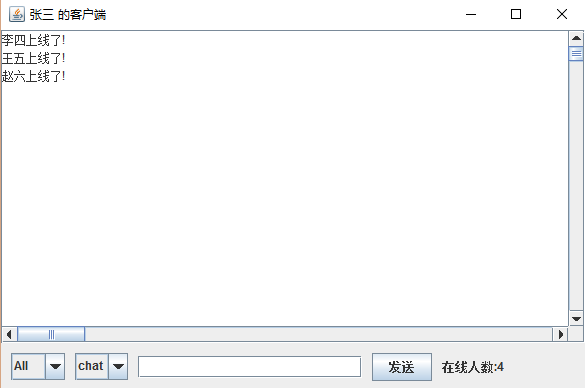
### 4.1 客户端进入聊天室

在客户端输入名字进入聊天室，如果名字重复会弹框提示。



### 4.2 聊天界面

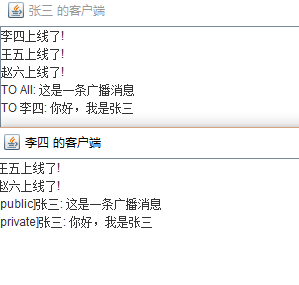
聊天界面显示客户端名字，好友列表，在线人数，消息类型等信息。



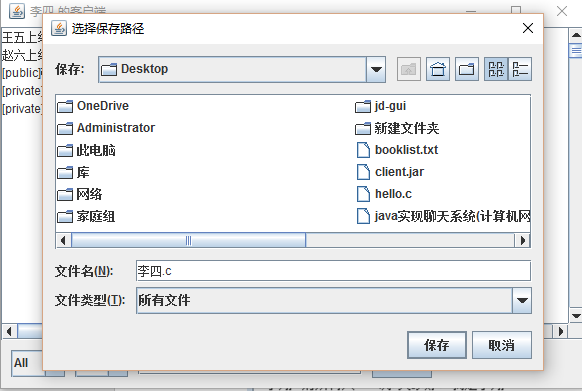
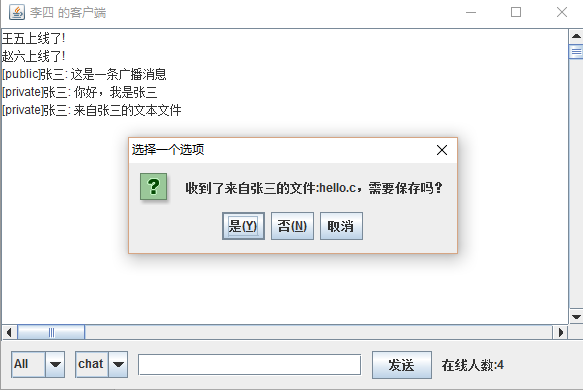
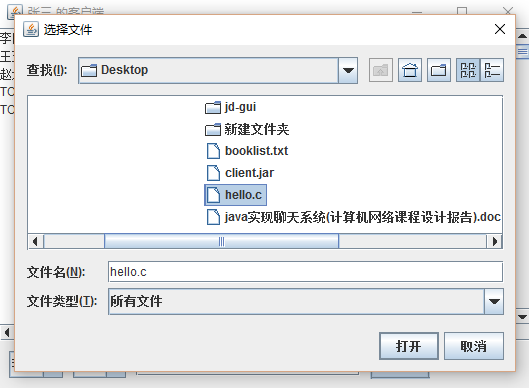
### 4.3 选择聊天类型

#### 4.3.1 聊天类型为chat

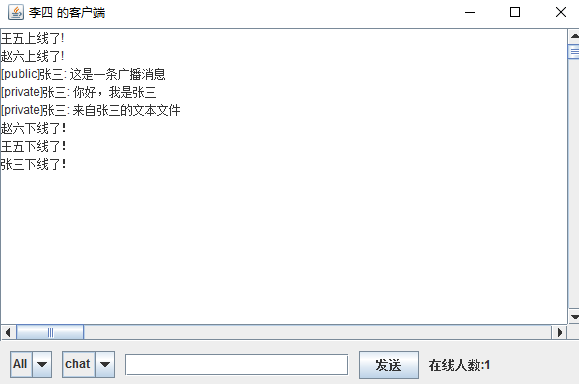
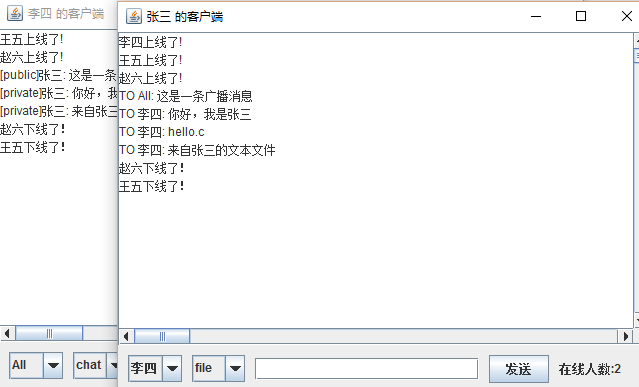
分为群聊和私聊：群聊是选择All，私聊是选择具体的用户名。



#### 4.3.2 聊天类型为file

发送文件，可以输入描述信息，客户端可以选择是否接收文件。

### 4.4 人数动态刷新



## 五、课程设计总结

本次课设用到了很多知识，比如Java程序设计中的一些高级功能。刚开始写的聊天程序很简单，只能实现客户端和服务器的双向通信，多个客户端会发生阻塞。之后发现原来并没有实现并发的访问。然后用线程实现客户端的处理就可以多个客户端同时在线并发响应了。但后来发送消息的过程中不断发现其实每个消息都是有规律的，他们可以用一个统一的接口来定义，这样就不用去考虑消息本身的细节，而是把重点放在核心代码上。于是我利用Java对象的序列化，把消息类进行统一为Message类，然后用对象输入输出流在客户端和服务器之间传输。但一个很显然的问题是如何处理消息，我之前想的是判断消息类型，但是条件判断的分支太多，并且这样不利于内容的扩充和维护。于是我想到了Java的一个很高级的功能：反射。

课程设计在这种不断学习与摸索中差不多完成了，这次课程设计的不足:

1. UI界面过于简陋，因为时间比较紧，所以没有细心设计。
2. 客户端和服务器之间消息传输的安全性检验没有涉及。
3. 文件传输的交互过于简单，应该让客户端主动选择是否接收。
4. 没有用到数据库，应该把聊天信息存储下来。

对于自己能独立的完成课程设计，还是颇为满意的，至少努力没有白费，学到的知识也非常多，最重要的应该是理解了TCP协议的原理，并学会了简单的应用，JAVA编程水平得到了一定的提高，面向对象的思想比以往更清晰了。

## 六、附录

### 6.1 参考文献

《Java程序设计语言（第4版）》James Gosling 人民邮电出版社

《Java编程思想》Bruce Eckel 机械工业出版社

### 6.2 源代码

#### 6.2.1客户端代码

**//Client.java**

package client;

import java.io.\*;

import java.net.\*;

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

import common.\*;

public class Client extends Thread {

public ClientUI ui; // 客户端界面

public String name; // 用户名

private Socket csocket; // 客户端套接字

private ObjectInputStream in; // 输入流

private ObjectOutputStream out; // 输出流

public static void main(String[] args) {

new Client(Config.IP, Config.PORT);

}

public Client(String server, int port) {

try {

csocket = new Socket(server, port);

out = new ObjectOutputStream(csocket.getOutputStream());

in = new ObjectInputStream(csocket.getInputStream());

} catch (Exception e) {

System.out.println(e.toString() + " 客户端初始化失败");

this.close();

System.exit(0);

}

this.start();

}

public void run() {

try {

// 获取用户名

while (name == null || name.equals(""))

name = JOptionPane.showInputDialog("请输入名字").trim();

// 第一次运行，发送登陆消息

send(new Message("login", name, null, null));

// 显示界面

ui = new ClientUI(this, name + " 的客户端");

// 消息处理主循环

while (true) {

Message rcv = (Message) in.readObject();

// 服务器端发来的消息

System.out.println(rcv.toString());

new ClientMsgType(this, rcv).process();

}

} catch (Exception e) {

System.out.println(e.toString() + " 客户端退出");

}

}

/\*\*

\* 客户端发送消息给服务器

\*

\* @param msg

\* 消息对象

\*/

public void send(Message msg) {

try {

out.writeObject(msg);

out.flush();

} catch (Exception e) {

System.out.println(e.toString() + " 消息转换出错");

}

}

/\*\*

\* 关闭所有资源

\*/

public void close() {

try {

if (ui != null)

ui.dispose();

if (out != null)

out.close();

if (in != null)

in.close();

if (csocket != null)

csocket.close();

} catch (IOException e) {

System.out.println(e.toString() + " 资源无法释放");

}

}

}

/\*\*

\* 客户端界面类

\*

\* @author Administrator

\*

\*/

@SuppressWarnings("serial")

class ClientUI extends Frame {

JTextArea msgArea; // 显示消息文本

JTextField msgField; // 输入消息文本

JComboBox<String> clientList; // 在线用户列表

JComboBox<String> msgType; // 聊天文本或者文件

JButton btn; // 发送消息按钮

JLabel cntLabel; // 统计在线人数

JScrollPane textAreaScrollPane;

JPanel textFieldPanel = new JPanel();

Client client;

public ClientUI(Client client, String winname) {

super(winname); // 继承父类的名字

setSize(600, 400);

this.client = client;

// 消息文本显示区域

msgArea = new JTextArea(400, 400);

msgArea.setEditable(false);

textAreaScrollPane = new JScrollPane(msgArea);

add(textAreaScrollPane, BorderLayout.CENTER);

// 发送消息区域

textFieldPanel.setLayout(new FlowLayout(0, 10, 10));

add(textFieldPanel, BorderLayout.SOUTH);

clientList = new JComboBox<String>(); // 在线用户列表

clientList.addItem("All");

textFieldPanel.add(clientList);

msgType = new JComboBox<String>(); // 聊天文本或者文件

msgType.addItem("chat");

msgType.addItem("file");

textFieldPanel.add(msgType);

msgField = new JTextField(20); // 输入消息文本

textFieldPanel.add(msgField);

btn = new JButton("发送"); // 发送消息按钮

btn.setMnemonic(KeyEvent.VK\_ENTER);

textFieldPanel.add(btn);

cntLabel = new JLabel("在线人数:1"); // 显示在线人数

textFieldPanel.add(cntLabel);

// 发送消息按钮监听器

btn.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

String type = (String) msgType.getSelectedItem();

String title = "";

String content = getText().trim();

byte[] fbyte = null;

if (content.equals("")) {

JOptionPane.showMessageDialog(client.ui, "输入不能为空");

return;

}

if (type.equals("file")) {

// 获取客户端发送文件

JFileChooser dlg = new JFileChooser();

dlg.setDialogTitle("选择文件");

int result = dlg.showOpenDialog(client.ui);

if (result == JFileChooser.APPROVE\_OPTION) {

File file = dlg.getSelectedFile();

fbyte = Common.file2Byte(file);

title = file.getName();

} else {

return;

}

append("TO " + getName() + ": " + title + "\n");

}

append("TO " + getName() + ": " + content + "\n");

client.send(new Message(type, client.name, getName(), title, content, fbyte));

clear();

}

});

addWindowListener(new WindowAdapter() {

public void windowClosing(WindowEvent e) {

if (JOptionPane.OK\_OPTION == JOptionPane.showConfirmDialog(client.ui, "确定退出吗?")) {

client.close();

System.exit(0);

}

}

});

setVisible(true);

}

/\*\*

\* 获得输入框中的消息文本

\*

\* @return

\*/

public String getText() {

return msgField.getText().trim();

}

/\*\*

\* 设置cntLabel文本以更新在线用户数

\* @param txt

\*/

public void setText(String txt) {

cntLabel.setText(txt);

}

/\*\*

\* 获得选中的聊天对象的名字

\*/

public String getName() {

return (String) clientList.getSelectedItem();

}

/\*\*

\* 获得在线用户数

\*

\* @return

\*/

public int getClientCount() {

return clientList.getItemCount();

}

/\*\*

\* 添加在线用户

\*

\* @param name

\* 用户名

\*/

public void addClient(String name) {

clientList.addItem(name);

}

/\*\*

\* 删除在线用户

\*

\* @param name

\* 用户名

\*/

public void removeClient(String name) {

clientList.removeItem(name);

}

/\*\*

\* 清空输入框

\*/

public void clear() {

msgField.setText(" ");

}

/\*\*

\* 添加文本到消息显示区域

\*

\* @param txt

\* 消息文本

\*/

public void append(String txt) {

msgArea.append(txt);

}

}

**//ClientMsgType.java**

package client;

import java.io.File;

import java.lang.reflect.Method;

import javax.swing.JFileChooser;

import javax.swing.JOptionPane;

import common.\*;

public class ClientMsgType {

Client client;

Message msg;

public ClientMsgType(Client client, Message msg) {

this.client = client;

this.msg = msg;

}

/\*\*

\* 处理接收到的消息

\*/

public void process() {

try {

Method method = this.getClass().getDeclaredMethod(msg.getType()); // 反射获取指定方法

method.invoke(this);

client.ui.setText("在线人数:" + (client.ui.getClientCount())); // 动态更新在线人数

} catch (Exception e) {

System.out.println(e.toString());

}

}

/\*\*

\* 聊天消息处理

\*/

public void chat() {

String toAll = "[public]";

if (msg.getTo().equals(client.name))

toAll = "[private]";

client.ui.append(toAll + msg.getFrom() + ": " + msg.getContent() + "\n");

}

/\*\*

\* 文件处理 利用字节数组可以处理所有文件

\*/

public void file() {

String toAll = "[public]";

if (msg.getTo().equals(client.name))

toAll = "[private]";

client.ui.append(toAll + msg.getFrom() + ": " + msg.getContent() + "\n");

int confirm = JOptionPane.showConfirmDialog(client.ui,

"收到了来自" + msg.getFrom() + "的文件:" + msg.getTitle() + "，需要保存吗？");

if (confirm != JOptionPane.YES\_OPTION)

return;

// 获取保存路径

JFileChooser dlg = new JFileChooser();

dlg.setDialogTitle("选择保存路径");

dlg.setSelectedFile(new File(msg.getTitle()));

int result = dlg.showSaveDialog(client.ui);

if (result != JFileChooser.APPROVE\_OPTION)

return;

File file = dlg.getSelectedFile();

if (file.exists()) {

int copy = JOptionPane.showConfirmDialog(null, "是否要覆盖当前文件？", "保存", JOptionPane.YES\_NO\_OPTION,

JOptionPane.QUESTION\_MESSAGE);

if (copy == JOptionPane.YES\_OPTION) {

dlg.approveSelection();

}

} else {

dlg.approveSelection();

}

if (Common.byte2File(msg.getFile(), file)) {

JOptionPane.showMessageDialog(client.ui, "文件保存成功");

} else {

JOptionPane.showMessageDialog(client.ui, "文件保存失败");

}

}

/\*\*

\* 在线提醒处理

\*/

public void online() {

client.ui.addClient(msg.getFrom());

if (msg.getContent() != null)

client.ui.append(msg.getFrom() + msg.getContent() + "\n");

}

/\*\*

\* 下线提醒处理

\*/

public void offline() {

client.ui.removeClient(msg.getFrom());

if (msg.getContent() != null)

client.ui.append(msg.getFrom() + msg.getContent() + "\n");

}

/\*\*

\* 登陆出错处理

\*/

public void login() {

client.ui.setVisible(false);

JOptionPane.showMessageDialog(client.ui, msg.getContent());

System.out.println(msg.getContent());

client.close();

new Client(Config.IP, Config.PORT);

}

}

#### 6.2.2服务端代码

**//Server.java**

package service;

import java.io.\*;

import java.util.\*;

import common.\*;

import java.net.\*;

public class Server {

public Server(int port) {

try {

@SuppressWarnings("resource")

ServerSocket server = new ServerSocket(port);

while (true) {

Socket conn = server.accept(); // 建立客户端套接字

new Thread(new ServerHandler(conn)).start(); // 新建线程和客户端建立全双工通信

}

} catch (Exception e) {

System.out.println(e.toString());

}

}

public static void main(String[] args) {

new Server(Config.PORT);

}

}

class ServerHandler implements Runnable {

private ObjectInputStream in; // 输入流

private ObjectOutputStream out; // 输出流

private String name; // 用户名

private static ArrayList<ServerHandler> clientList = new ArrayList<ServerHandler>(); // 把用户线程放入对象数组

private static HashMap<String, ServerHandler> clientMap = new HashMap<>(); // 把用户名和连接线程关联便于查找

public ServerHandler(Socket socket) {

try {

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

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

} catch (IOException e) {

System.out.println(e.toString() + " 服务端初始化失败");

}

}

public void run() {

if (!loginVerify())

return; // 登陆校验

try {

addClient(); // 客户端经过登陆检验 添加到列表

sendMessage(new Message("online", name, "All", "上线了!")); // 广播用户列表

while (true) {

Message rcv = (Message) in.readObject(); // 等待客户端消息

sendMessage(rcv);

}

} catch (Exception e) {

System.out.println(e.toString() + " 客户端已退出");

} finally {

removeClient();

sendMessage(new Message("offline", name, "All", "下线了！")); // 广播下线通知

}

}

/\*\*

\* 登陆校验

\*

\* @return

\*/

private Boolean loginVerify() {

try {

Message rcv = (Message) in.readObject();

System.out.println(rcv.toString());

this.name = rcv.getFrom();

if (clientMap.containsKey(name)) {

// 用户名重复

sendToClient(this, new Message("login", null, name, "用户名已存在"));

return false;

}

} catch (Exception e) {

System.out.println(e.toString() + "登陆检验失败");

return false;

}

return true;

}

/\*\*

\* 添加用户到列表

\*

\* @return

\*/

private Boolean addClient() {

if (clientMap.containsKey(name))

return false;

clientList.add(this);

clientMap.put(name, this);

return true;

}

/\*\*

\* 移除当前用户

\*

\* @return

\*/

private Boolean removeClient() {

clientList.remove(this);

clientMap.remove(name);

return true;

}

/\*\*

\* 发送消息

\*

\* @param msg

\* @throws IOException

\*/

public void sendMessage(Message msg) {

new ServerMsgType(this, msg).process();

}

/\*\*

\* 发送消息给所有用户,排除name

\*

\* @param msg

\*/

public void sendAllClient(Message msg) {

String name = msg.getFrom();

ServerHandler sh = clientMap.get(name);

synchronized (clientList) {

for (ServerHandler client : clientList) {

if (client != sh) {

sendToClient(client, msg);

}

}

}

}

/\*\*

\* 发送消息给指定用户

\*

\* @param msg

\*/

public void sendToClient(Message msg) {

String name = msg.getTo();

ServerHandler client = clientMap.get(name);

sendToClient(client, msg);

}

/\*\*

\* 发送消息给指定用户线程

\*

\* @param client

\* @param msg

\*/

public void sendToClient(ServerHandler client, Message msg) {

try {

client.out.writeObject(msg);

client.out.flush();

System.out.println("send to client: " + msg);

} catch (IOException e) {

System.out.print(e.toString() + " 服务端发送消息失败");

// client.interrupt();

}

}

/\*\*

\* 更新用户列表

\*

\* @param msg

\*/

public synchronized void sendClientList(Message msg) {

// 新上线的用户

ServerHandler newclient = clientMap.get(msg.getFrom());

for (ServerHandler client : clientList) {

if (client == newclient)

continue;

// 给新上线用户所有列表

sendToClient(newclient, new Message(msg.getType(), client.name, newclient.name, null));

// 更新老用户

sendToClient(client, new Message(msg.getType(), newclient.name, client.name, msg.getContent()));

}

}

}

**//ServerMsgType.java**

package service;

import java.lang.reflect.Method;

import common.\*;

public class ServerMsgType {

ServerHandler server;

Message msg;

public ServerMsgType(ServerHandler server, Message msg) {

this.server = server;

this.msg = msg;

}

public void process() {

try {

Method method = this.getClass().getDeclaredMethod(msg.getType());

method.invoke(this);

} catch (Exception e) {

System.out.println(e.toString() + " 使用了未定义的操作");

try {

Method method = this.getClass().getDeclaredMethod("chat");

method.invoke(this);

} catch (Exception e1) {

System.out.println(e1.toString() + " 默认处理失败");

}

}

}

public void chat() {

if (msg.getTo().equals("All")) {

// 给所有人发送消息

server.sendAllClient(msg);

} else {

// 发送消息给指定的人

server.sendToClient(msg);

}

}

public void online() {

server.sendClientList(msg);

}

public void offline() {

server.sendAllClient(msg);

}

public void login() {

server.sendToClient(msg);

}

}

#### 6.2.3 公共类

**//Message.java**

package common;

import java.io.Serializable;

public class Message implements Serializable, Cloneable {

private static final long serialVersionUID = -8104648599710335924L;

private String type; // 消息类型

private String from; // 来源

private String to; // 目的

private String title; // 附件名

private String content; // 消息内容

private byte[] file; // 附件内容

public Message() {

}

public Message(String type, String from, String to, String content) {

this.set(type, from, to, null, content);

}

public Message(String type, String from, String to, String title, String content, byte[] file) {

this.set(type, from, to, title, content);

this.setFile(file);

}

public void set(String type, String from, String to, String title, String content) {

this.type = type;

this.from = from;

this.to = to;

this.title = title;

this.content = content;

}

public Message clone() {

Message m = null;

try {

m = (Message) super.clone();

} catch (CloneNotSupportedException e) {

e.printStackTrace();

}

return m;

}

public String toString() {

return "type=" + type + " from=" + from + " to=" + to + " title=" + title + " content=" + content;

}

public String getType() {

return type;

}

public void setType(String type) {

this.type = type;

}

public String getFrom() {

return from;

}

public void setFrom(String from) {

this.from = from;

}

public String getTo() {

return to;

}

public void setTo(String to) {

this.to = to;

}

public String getTitle() {

return title;

}

public void setTitle(String title) {

this.title = title;

}

public String getContent() {

return content;

}

public void setContent(String content) {

this.content = content;

}

public byte[] getFile() {

return file;

}

public void setFile(byte[] file) {

this.file = file;

}

}

**// Commmon.java**

package common;

import java.io.\*;

public class Common {

private static final int DEFAULT\_BUFFER\_SIZE = 10 \* 1024 \* 1024; // 10M缓冲区

/\*\*

\* 将字节序列写入指定文件

\*

\* @param in

\* 待写入字节数组

\* @param out

\* 目标文件

\* @return 是否写入成功

\*/

public static boolean byte2File(byte[] in, File out) {

try {

FileOutputStream fout = new FileOutputStream(out);

fout.write(in);

fout.close();

} catch (Exception e) {

System.out.println(e.toString() + " 字节转换失败");

return false;

}

return true;

}

/\*\*

\* 将指定文件转化成字节数组

\*

\* @param file

\* 待转化文件

\* @return 转化后的字节数组

\*/

public static byte[] file2Byte(File file) {

byte[] buffer = new byte[DEFAULT\_BUFFER\_SIZE];

int len = 0;

try {

FileInputStream fin = new FileInputStream(file);

len = fin.read(buffer);

fin.close();

} catch (Exception e) {

System.out.println(e.toString() + " 文件无法转化成字节");

}

byte[] ret = new byte[len];

for (int i = 0; i < len; i++)

ret[i] = buffer[i];

return ret;

}

/\*\*

\* 复制指定文件到目标文件

\*

\* @param in

\* 待复制的源文件

\* @param out

\* 目标文件

\* @return

\*/

public static boolean file2File(File in, File out) {

try {

FileInputStream fin = new FileInputStream(in);

FileOutputStream fout = new FileOutputStream(out);

byte[] buffer = new byte[DEFAULT\_BUFFER\_SIZE];

int len = 0;

while ((len = fin.read(buffer)) != -1) {

fout.write(buffer, 0, len);

}

fin.close();

fout.close();

} catch (Exception e) {

System.out.println(e.toString() + " 文件转换失败");

return false;

}

return true;

}

}

**//Config.java**

package common;

public class Config {

public static final String IP = "192.168.71.2";

public static final int PORT = 6666;

}