广州大学学生实验报告

开课实验室：电子楼 416B　　2019 年 10 月 15 日

学院	计算机科学与网络工程学院	年级、专业、班	软件 171	姓名	谢金宏	学号	1706300001
实验课程名称	Java 语言实验					成绩	
实验项目名称	实验 4					指导老师	王宇

一、实验目的

熟悉 Java 综合应用程序的开发。

二、实验任务

编写一个 Java 应用程序，实现图形界面多人聊天室（多线程实现）。

要求聊天室窗口标题是“欢迎使用 XXX 聊天室应用”，其中“XXX”是自己的班级姓名学号，如“软件 171 张三 1234”。

三、实验内容

1. 设计通信协议

聊天室服务器与客户端之间使用 Java 套接字进行双向通信。服务器可与多个客户端同时建立连接，起到转发消息的作用。客户端在同一时间只能与一个服务器建立连接。

客户端向要加入聊天室时，需要向服务器发起昵称登录请求。服务器检查此新客户端的昵称是否与聊天室中的现有昵称冲突。若无冲突，则允许客户端登录，并将新客户端加入转发列表中，并向转发列表中的全体客户端发送新客户端的昵称登录消息；否则，拒绝客户端的登录。客户端收到昵称登录消息时，将新成员的昵称显示在聊天面板上。

当客户端在聊天室中发言时，向服务器发送聊天消息。服务器收到聊天消息后，向在转发列表中的全体客户端转发这条聊天消息。客户端收到新聊天消息时，将聊天消息显示在聊天面板上。

客户端在退出聊天室时向服务器发送昵称注销消息。服务器收到注销消息，或因网络问题失去客户端的连接时，将注销或失联的客户端从转发列表中移除，并向转发列表中的现有成员发送昵称注销消息。客户端收到昵称注销消息时，将离开成员的昵称显示在聊天面板上。

2. 服务器代码

```
/**
 * 聊天室应用的服务器端
 */
public class ChatServer {
    public static void main(String... args) {
        new ChatServer();
    }

    private ChatServerWindow window;
    private ServerSocket serverSocket;
    private java.util.List<Messenger> clientMessengers;


    ChatServer() {
        window = new ChatServerWindow(this);
        clientMessengers = Collections.synchronizedList(new ArrayList<Messenger>
());

        window.showLog(ChatApplication.serverGreeting);
    }



    /**
     * 返回服务器是否处于监听状态。
```

```
     *
     * @return
     */
    boolean isStarted() {
        return serverSocket != null && serverSocket.isBound();
    }


    /**
     * 启动服务器并监听指定端口。
     *
     * @param port
     * @return 返回服务器是否启动成功。
     */
    synchronized boolean start(int port) {
        try {
            serverSocket = new ServerSocket(port);
        } catch (Exception e) {

            window.showLog("服务器无法使用" + port + "端口", Color.RED);

            return false;
        }

        ChatServer thisServer = this;
        new Thread() {
            @Override
            public void run() {
                while (isStarted()) {
                    try {
                        Socket clientSocket = serverSocket.accept();
                        Messenger messenger = new Messenger(Messenger.WorkMode.S
ERVER, clientSocket);
                        thisServer.handshake(messenger);
                        thisServer.hearFromClient(messenger);
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
            }
        }.start();


        window.showLog("服务器已启动，正在监听" + port + "端口。", Color.GREEN);

        return true;
    }
```

```
    /**
     * 停止服务器。
     */
    synchronized void stop() {
        if (serverSocket != null) {
            try {
                for (Messenger messenger : clientMessengers) {
                    messenger.disconnect();
                }
                clientMessengers.clear();
                serverSocket.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        serverSocket = null;

        window.showLog("服务器已停止。", Color.RED);
    }


    /**
     * 与某个特定的客户端的握手过程。
     *
     * 握手过程中若出现任何异常，则拒绝该客户端的连接。
     */
    public synchronized void handshake(Messenger messenger) {
        synchronized (clientMessengers) {
            try {

                window.showLog("正在与" + messenger.getSocketDescription()  + "进
行握手。");


                NicknameLoginMessage loginMessage = (NicknameLoginMessage)messen
ger.receive();
                messenger.setNickname(loginMessage.nickname);
                if (nicknameConflicts(messenger.getNickname())) {
                    messenger.send(new NicknameLoginRejectedMessage());

                    window.showLog("因昵称冲突而拒绝了
```

```
" + messenger.getSocketDescription()  + "的登录。");

                } else {
                    messenger.send(new NicknameLoginAcceptedMessage());

                    window.showLog("接受了

" + messenger.getSocketDescription()  + "的登录。");

                    clientMessengers.add(messenger);
                    boardcastMessage(loginMessage);
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }


    /**
     * 持续从客户端接收消息。
     */
    public void hearFromClient(Messenger clientMessenger) {
        ChatServer thisServer = this;
        new Thread() {
            @Override
            public void run() {
                while (clientMessenger.isConnected()) {
                    try {
                        Message message = clientMessenger.receive();
                        thisServer.handleMessage(message);
                    } catch (MessengerFailException e) {
                        thisServer.handleMessengerFailException(e);
                    }
                }
            }
        }.start();
    }


    /**
     * 与指定的客户端断开连接。
     */
    private synchronized void disconnectClient(Messenger clientMessenger) {
        synchronized (clientMessengers) {
            if (clientMessenger != null) {
                String clientDescription = clientMessenger.getSocketDescription(
);
                String clientNickname = clientMessenger.getNickname();
                boolean boardcastFlag = clientMessengers.contains(clientMessenge
r);

                clientMessenger.disconnect();
                clientMessengers.remove(clientMessenger);
                if (boardcastFlag) {
                    boardcastMessage(new NicknameLogoutMessage(clientNickname));
                }
                window.showLog("断开了" + clientDescription + "的连接。");
            }
        }
    }


    /**
     * 断开与 Nickname 对应的客户端的连接。
     */
    public void disconnectClientWithNickname(String nickname) {
        Messenger messenger = getMessengerByNickname(nickname);
        if (messenger != null) {
            disconnectClient(messenger);
        }
    }


    /**
     * 获取 Nickname 对应的 MessageThread。
     */
    private Messenger getMessengerByNickname(String nickname) {
        for (Messenger messenger : clientMessengers) {
            if (messenger.getNickname().equals(nickname)) {
                return messenger;
            }
        }
        return null;
    }


    /**
```

```
     * 返回 Nickname 是否与现有 Nickname 冲突。
     *
     * @return 若冲突，返回 true。
     */
    public boolean nicknameConflicts(String nickname) {
        for (Messenger messenger : clientMessengers) {
            if (messenger.getNickname().equals(nickname)) {
                return true;
            }
        }
        return false;
    }


    /**
     * 处理某个 Messenger 产生的异常。
     */
    public void handleMessengerFailException(MessengerFailException e) {
        disconnectClient(e.source);
    }


    /**
     * 处理从某一个客户端收到的消息。
     *
     * 每收到一个聊天消息，就向已连接的全体客户端广播这个消息。
     * @param message
     */
    public void handleMessage(Message message) {
        switch (message.getClass().getSimpleName()) {
            case "NicknameLogoutMessage": {
                disconnectClientWithNickname(((NicknameLogoutMessage)message).ni
ckname);
                return;
            }
        }

        boardcastMessage(message);
    }


    /**
     * 向所有已连接的客户端广播消息。
     */
    public void boardcastMessage(Message message) {
        for (Messenger messenger : clientMessengers) {
            try {
                messenger.send(message);
            } catch (MessengerFailException e) {
                handleMessengerFailException(e);
            }
        }
    }
}
```

3. 服务器窗口线程代码

```
/**
 * 服务器端窗口
 */
class ChatServerWindow extends JFrame {
    static final long serialVersionUID = 0x01C4AD270344EB15L;

    private ChatTextbox logPane;


    ChatServerWindow(ChatServer server) {
        this.setTitle(ChatApplication.serverTitle);
        this.setLayout(new BorderLayout());

        logPane = new ChatTextbox();
        this.add(logPane, BorderLayout.CENTER);

        JPanel controlPanel = new JPanel(new FlowLayout());

        controlPanel.add(new JLabel("服务器监听端口："));


        JTextField portField = new JTextField(Integer.toString(ChatApplicationDe
faults.serverPort), 8);
        controlPanel.add(portField);


        JButton startOrStopServerButton = new JButton("启动服务器");

        startOrStopServerButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if (server.isStarted()) {
```

```
                    server.stop();

                    startOrStopServerButton.setText("启动服务器");

                } else {
                    int port = 0;

                    try {
                        port = Integer.parseInt(portField.getText());
                    } catch (NumberFormatException err) {

                        showLog("端口必须为数字。", Color.RED);

                    }

                    if (port < 1025 || port > 65535) {

                        showLog("指定的端口不合法，合法的端口范围是 1025~65535。
", Color.RED);

                        return;
                    }

                    boolean serverOK = server.start(port);
                    if (serverOK) {

                        startOrStopServerButton.setText("停止服务器");

                    }
                }
            }
        });
        controlPanel.add(startOrStopServerButton);

        this.add(controlPanel, BorderLayout.SOUTH);

        this.setSize(ChatApplicationDefaults.serverWindowWidth, ChatApplicationD
efaults.serverWindowHeight);
        this.setLocationRelativeTo(null);
        this.setVisible(true);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }


    /**
     * 在日志窗口显示一条日志。
     *
     * @param log 日志
     */
    void showLog(String log) {
        this.showLog(log, Color.BLACK);
    }


    /**
     * 在日志窗口显示一条带颜色的日志。
     *
     * @param log 日志
     * @param color 日志颜色
     */
    void showLog(String log, Color color) {
        logPane.appendText(new Date().toString() + ": " + log + "\n", color);
    }
}
```

4. 客户端代码

```
/**
 * 聊天室应用的客户端
 */
public class ChatClient {
    public static void main(String... args) {
        new ChatClient();
    }

    private ChatClientWindow window;
    private Messenger messenger;
    private String nickname;


    ChatClient() {
        window = new ChatClientWindow(this);
    }


    /**
     * 返回客户端是否连接到了聊天室。
     *
     * @return 若连接到聊天室，返回 true。
     */
```

```
    public boolean isConnected() {
        return messenger != null && messenger.isConnected();
    }


    /**
     * 使用指定参数连接聊天室。
     *
     * @param hostname 聊天室主机名
     * @param port 聊天室监听端口
     * @param nickname 连接聊天室时使用的昵称
     */
    public synchronized void connect(String hostname, int port, String nickname)
{
        if (!isConnected()) {
            this.nickname = nickname;

            ChatClient thisClient = this;
            new Thread() {
                @Override
                public void run() {
                    try {
                        messenger = new Messenger(Messenger.WorkMode.CLIENT, hos
tname, port, nickname);
                        thisClient.handshake();
                        thisClient.receiveAndHandleMessages();
                    } catch(MessengerFailException e) {
                        thisClient.handleMessengerFailException(e);
                    }
                }
            }.start();
        }
    }


    /**
     * `connect()`过程中与服务器进行握手的过程。
     *
     * 向服务器发送 NicknameLoginMessage，若服务器同意本次登录则返回
NicknameLoginAcceptedMessage;
     * 否则返回 NicknameLoginReejctedMessage。
     */
    private void handshake() throws MessengerFailException {
        assert(isConnected());

        messenger.send(new NicknameLoginMessage(this.nickname));
        Message response = messenger.receive();

        if (response instanceof NicknameLoginAcceptedMessage) {
            window.setUIMode(ChatClientWindow.UIMode.CONNECTED);
        } else {
            disconnect();

            window.showText("昵称已被占用，请更换一个昵称后再试。\n", Color.ORANGE);

            window.setUIMode(ChatClientWindow.UIMode.DISCONNECTED);
        }
    }


    /**
     * 断开与服务器的连接。
     */
    public synchronized void disconnect() {
        if (isConnected()) {
            try {
                messenger.disconnect();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        messenger = null;
    }


    /**
     * 向服务器发送一条聊天消息。
     *
     * @param content 聊天消息的内容
     */
    public void sendChatMessage(String content) {
        try {
            messenger.send(new ChatMessage(nickname, content));
        } catch (MessengerFailException e) {
```

```
            this.handleMessengerFailException(e);
        }
    }


    /**
     * 处理消息在网络上传输时产生的异常。
     */
    public synchronized void handleMessengerFailException(MessengerFailException
e) {
        e.printStackTrace();

        if (isConnected()) {
            disconnect();

            window.showText("与服务器已断开连接。\n", Color.RED);

            window.setUIMode(ChatClientWindow.UIMode.DISCONNECTED);
        }
    }


    /**
     * 接收并处理服务器传送的消息。
     */
    private void receiveAndHandleMessages() throws MessengerFailException {
        while (isConnected()) {
            Message message = messenger.receive();
            switch (message.getClass().getSimpleName()) {
                case "ChatMessage": {
                    handleChatMessage((ChatMessage)message);
                    break;
                }
                case "NicknameLoginMessage": {
                    handleNicknameLoginMessage((NicknameLoginMessage)message);
                    break;
                }
                case "NicknameLogoutMessage": {
                    handleNicknameLogoutMessage((NicknameLogoutMessage)message);
                    break;
                }
            }
        }
    }
```

```
    /**
     * 将聊天消息显示在窗口上。
     */
    private void handleChatMessage(ChatMessage message) {
        Color senderColor = Colorful.getColorFromString(message.sender);
        window.showText(message.sender, senderColor);

        window.showText("在" + message.sentDate.toString() + "时说：\n");

        window.showText("    " + message.content + "\n");
    }


    /**
     * 显示某位用户的登陆消息。
     */
    private void handleNicknameLoginMessage(NicknameLoginMessage message) {
        Color nicknameColor = Colorful.getColorFromString(message.nickname);
        window.showText(message.nickname, nicknameColor);

        window.showText("在" + new Date().toString() + "时进入了聊天室。\n");

    }


    /**
     * 显示某位用户的离开消息。
     */
    private void handleNicknameLogoutMessage(NicknameLogoutMessage message) {
        Color nicknameColor = Colorful.getColorFromString(message.nickname);
        window.showText(message.nickname, nicknameColor);

        window.showText("在" + new Date().toString() + "时离开了聊天室。\n");

    }
}
```

5. 客户端窗口线程代码

```
class ChatClientWindow extends JFrame implements ActionListener {
    static final long serialVersionUID = 0xA02C3B05814A7B8BL;

    static enum UIMode {
        DISCONNECTED, CONNECTING, CONNECTED;
    };

    private ChatClient client;
```

```
    private JTextField serverHostnameInput;
    private JTextField serverPortInput;
    private JTextField nicknameInput;
    private JButton connectOrDisconnectServerButton;
    private JTextField chatMessageInput;
    private JButton submitMessageButton;
    private ChatTextbox chatTextbox;


    ChatClientWindow(ChatClient client) {
        this.client = client;

        this.setTitle(ChatApplication.clientTitle);
        this.setLayout(new BorderLayout());

        this.setupUI();
        this.setUIMode(UIMode.DISCONNECTED);

        this.setSize(ChatApplicationDefaults.clientWindowWidth, ChatApplicationD
efaults.clientWindowHeight);
        this.setLocationRelativeTo(null);
        this.setVisible(true);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }


    /**
     * 在启动时建立图形界面。
     */
    private synchronized void setupUI() {
        chatTextbox = new ChatTextbox();
        this.add(chatTextbox, BorderLayout.CENTER);

        JPanel controlPane = new JPanel(new BorderLayout());
        this.add(controlPane, BorderLayout.SOUTH);

        JPanel configPane = new JPanel(new FlowLayout());
        controlPane.add(configPane, BorderLayout.NORTH);

        serverHostnameInput = new JTextField("127.0.0.1", 20);
        configPane.add(new JLabel("服务器主机名："));
        configPane.add(serverHostnameInput);

        serverPortInput = new JTextField(Integer.toString(ChatApplicationDefault
```

```
s.serverPort), 8);
        configPane.add(new JLabel("端口："));
        configPane.add(serverPortInput);

        nicknameInput = new JTextField(12);
        configPane.add(new JLabel("聊天昵称："));
        configPane.add(nicknameInput);


        connectOrDisconnectServerButton = new JButton("连接服务器");
        connectOrDisconnectServerButton.addActionListener(this);
        configPane.add(connectOrDisconnectServerButton);

        JPanel messagePane = new JPanel(new FlowLayout());
        controlPane.add(messagePane, BorderLayout.CENTER);

        chatMessageInput = new JTextField(64);
        messagePane.add(new JLabel("聊天消息："));
        messagePane.add(chatMessageInput);


        submitMessageButton = new JButton("发送消息");
        submitMessageButton.addActionListener(this);
        messagePane.add(submitMessageButton);
    }


    /**
     * 设置图形界面的模式。
     *
     * 图形界面的模式包括未连接模式、正在连接模型和已连接模式。
     * @see ChatClientWindow.UIMode
     */
    public synchronized void setUIMode(UIMode mode) {
        switch (mode) {
            case DISCONNECTED: {
                serverHostnameInput.setEditable(true);
                serverPortInput.setEditable(true);
                nicknameInput.setEditable(true);

                connectOrDisconnectServerButton.setText("连接聊天室");
                connectOrDisconnectServerButton.setEnabled(true);
```

```
                chatMessageInput.setEditable(false);
                submitMessageButton.setEnabled(false);
                break;
            }
            case CONNECTING: {
                serverHostnameInput.setEditable(false);
                serverPortInput.setEditable(false);
                nicknameInput.setEditable(false);

                connectOrDisconnectServerButton.setText("尝试连接中");

                connectOrDisconnectServerButton.setEnabled(false);
                break;
            }
            case CONNECTED: {
                serverHostnameInput.setEditable(false);
                serverPortInput.setEditable(false);
                nicknameInput.setEditable(false);

                connectOrDisconnectServerButton.setText("离开聊天室");

                connectOrDisconnectServerButton.setEnabled(true);
                chatMessageInput.setEditable(true);
                submitMessageButton.setEnabled(true);
                break;
            }
        }
    }


    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == connectOrDisconnectServerButton) {
            onClickConnectOrDisconnectServerButton();
        } else if (e.getSource() == submitMessageButton) {
            onClickSubmitMessageButton();
            return;
        }
    }


    private void onClickConnectOrDisconnectServerButton() {
        if (!client.isConnected()) {
            String serverHost = serverHostnameInput.getText();
            int serverPort = 0;
            String nickname = nicknameInput.getText();

            if (serverHost.isEmpty()) {
                this.showText("服务器主机地址不能为空。\n", Color.ORANGE);
                return;
            }

            if (serverPortInput.getText().isEmpty()) {
                this.showText("服务器端口不能为空。\n", Color.ORANGE);
                return;
            }

            try {
                serverPort = Integer.parseInt(serverPortInput.getText());
                if (serverPort < 1025 || serverPort > 65535) {
                    throw new NumberFormatException();
                }
            } catch (NumberFormatException e) {
                this.showText("服务器端口不合法，合法的端口为 1025~65535 范围内的数字。
\n", Color.ORANGE);
                return;
            }

            if (nickname.isEmpty()) {
                this.showText("用户昵称不能为空。\n", Color.ORANGE);
                return;
            }

            client.connect(serverHost, serverPort, nickname);

            this.setUIMode(UIMode.CONNECTING);

            this.showText("尝试连接到聊天室……\n", Color.GRAY);
        } else {
            client.disconnect();
            this.setUIMode(UIMode.DISCONNECTED);

            this.showText("已离开聊天室。\n", Color.GRAY);
        }
    }


    private void onClickSubmitMessageButton() {
        String content = chatMessageInput.getText().trim();
        chatMessageInput.setText("");
        if (!content.isEmpty()) {
```

```
            client.sendChatMessage(content);
        } else {
            this.showText("不能发送空白消息。\n", Color.ORANGE);
        }
    }

    /**
     * 在聊天窗口上显示文本。
     */
    void showText(String txt) {
        this.chatTextbox.appendText(txt);
    }

    /**
     * 在聊天窗口上显示带颜色的文本。
     */
    void showText(String txt, Color color) {
        this.chatTextbox.appendText(txt, color);
    }
}
```

6. 消息代码

```
/**
 * 服务器与客户端之间传递消息的公共基类
 */
class Message implements Serializable {
    final static long serialVersionUID = 0xEAF24721CC7A9F01L;
}

/**
 * 客户端之间的聊天消息
 */
class ChatMessage extends Message {
    final static long serialVersionUID = 0x0E3A4BB9F7344E253L;

    public String sender;  // 消息的发送方
    public Date sentDate;  // 消息发送的时间

    public String content; // 消息的内容

    public ChatMessage(String sender, String content) {
        this.sender = sender;
        this.sentDate = new Date();
        this.content = content;
    }
}

/**
 * 客户端发送的昵称登录消息
 */
class NicknameLoginMessage extends Message {
    final static long serialVersionUID = 0x9E11DA1DFB66BEDAL;

    public String nickname;

    NicknameLoginMessage(String nickname) {
        this.nickname = nickname;
    }
}

/**
 * 服务器端在接受客户端昵称登录请求时发送的消息
 */
class NicknameLoginAcceptedMessage extends Message {
    final static long serialVersionUID = 0xAC4A3CD4D1DB047DL;
}

/**
 * 服务器端在拒绝客户端昵称登录请求时发送的消息
 */
class NicknameLoginRejectedMessage extends Message {
    final static long serialVersionUID = 0xB41FC5354327354BL;
}

/**
```

```
 * 客户端发送的昵称注销消息
 */
class NicknameLogoutMessage extends Message {
    final static long serialVersionUID = 0xC08A44F095C06D66L;

    public String nickname;


    NicknameLogoutMessage(String nickname) {
        this.nickname = nickname;
    }
}
```

7. 消息代理中间件代码

```
/**
 * 客户端与服务器的消息通信代理
 */
public class Messenger {
    static enum WorkMode {
        SERVER, CLIENT
    };

    private String nickname;

    private Socket socket;
    private ObjectInputStream in;
    private ObjectOutputStream out;


    Messenger(WorkMode mode, Socket socket) throws MessengerFailException {
        try {
            this.socket = socket;
            establishStreams(mode);
        } catch (Exception e) {
            throw raise(e);
        }

    }


    Messenger(WorkMode mode, String hostname, int port, String nickname) throws
MessengerFailException {
        try {
            this.socket = new Socket(hostname, port);
            this.nickname = nickname;
            establishStreams(mode);
        } catch (Exception e) {
            throw raise(e);
        }
    }


    /**
     * 建立 ObjectInputStream 和 ObjectOutputStream
     */
    private void establishStreams(WorkMode mode) throws IOException {
        // 客户端与服务建立对象流的顺序相对应，

        // 若一端先建立输入流，则另一端先建立输出流。

        // 详见 ObjectInputStream/ObjectOutputStream 构造函数注释。
        switch (mode) {
            case CLIENT: {
                out = new ObjectOutputStream(socket.getOutputStream());
                in = new ObjectInputStream(socket.getInputStream());
                break;
            }
            case SERVER: {
                in = new ObjectInputStream(socket.getInputStream());
                out = new ObjectOutputStream(socket.getOutputStream());
                break;
            }
        }
    }


    public void setNickname(String nickname) {
        this.nickname = nickname;
    }


    public String getNickname() {
        return nickname;
    }


    public String getSocketDescription() {
        return socket.getInetAddress().toString() + ":" + socket.getPort() + "["
+ nickname + "]";
```

```
    }


    /**
     * 返回是否成功建立了连接。
     * @return 若连接成功，返回 true。
     */
    boolean isConnected() {
        return socket != null && socket.isConnected();
    }


    /**
     * 断开连接。
     */
    public void disconnect() {
        if (socket != null && socket.isConnected()) {
            try {
                out.writeObject(new NicknameLogoutMessage(nickname));
                socket.close();
            } catch(Exception e) {
                e.printStackTrace();
            }
        }
        socket = null;
    }


    /**
     * 发送一条消息。
     */
    public void send(Message message) throws MessengerFailException {
        try {
            out.writeObject(message);
        } catch (Exception e) {
            throw raise(e);
        }
    }


    /**
     * 接收一条消息。
     */
    public Message receive() throws MessengerFailException {
        try {
            Message message = (Message)in.readObject();
            return message;
        } catch (Exception e) {
            throw raise(e);
        }
    }


    /**
     * 包装异常。
     */
    private MessengerFailException raise(Exception e) {
        return new MessengerFailException(this, e);
    }
}


/**
 * 消息传递过程中的异常
 */
class MessengerFailException extends Exception {
    final static long serialVersionUID = 0x140C2BDFACB72301L;


    /** 产生异常的 Messenger 实例 */
    public Messenger source;


    /** 引发 MessengerFailException 的内部异常 */
    public Exception innerException;

    MessengerFailException(Messenger source, Exception innerException) {
        this.source = source;
        this.innerException = innerException;
    }
}
```

8. 彩色文本控件和其他工具代码

```
/**
 * 生成颜色的工具类
```

```
 */
public class Colorful {

    /**
     * 为一个字符串生成它的颜色。
     */
    static Color getColorFromString(String str) {
        int r = 0, g = 0, b = 0;
        try {
            byte[] bytes = MessageDigest.getInstance("MD2").digest(str.getBytes(
));
            r = Byte.toUnsignedInt(bytes[0]);
            g = Byte.toUnsignedInt(bytes[1]);
            b = Byte.toUnsignedInt(bytes[2]);
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
        return new Color(r, g, b);
    }
}
/**
 * 封装插入格式化文本功能，并可垂直滚动的文本控件
 */
public class ChatTextbox extends JScrollPane {
    final static long serialVersionUID = 0xB35AB87EBE4BA35DL;

    private JTextPane textPane = new JTextPane();


    ChatTextbox() {
        this.setViewportView(textPane);
        textPane.setEditable(false);
    }


    public void appendText(String txt) {
        appendText(txt, Color.BLACK);
    }


    public void appendText(String txt, Color color) {
        StyledDocument doc = textPane.getStyledDocument();

        Style style = textPane.addStyle("Color Style", null);
        StyleConstants.setForeground(style, color);
        try {
            doc.insertString(doc.getLength(), txt, style);
        }
        catch (BadLocationException e) {}
    }
}
```

四、实验结果记录（程序运行结果截图）

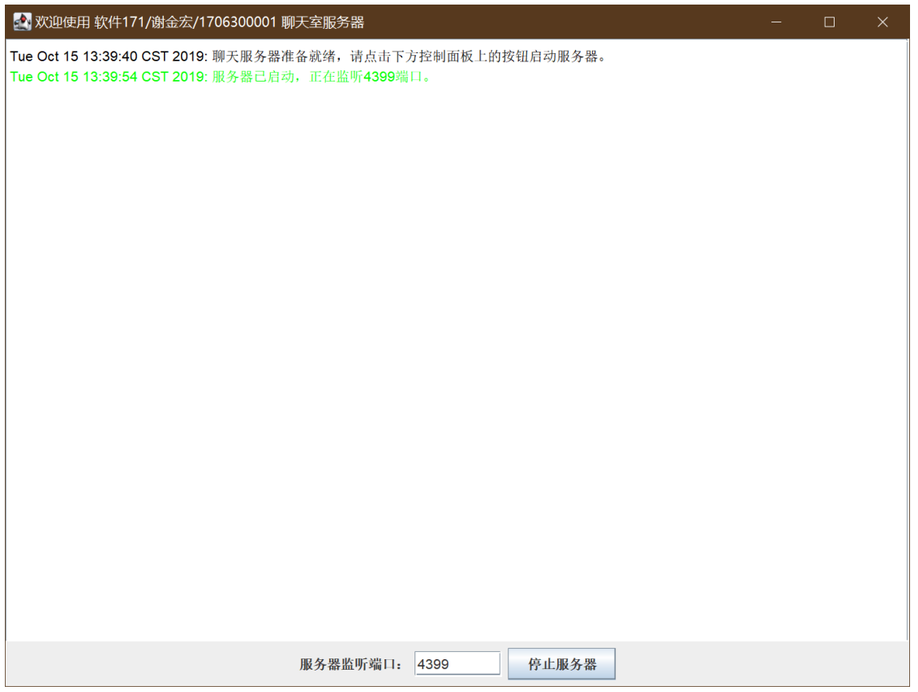

Figure 1 启动服务器

Figure 2 启动客户端

Figure 3 开始聊天

Figure 4 中途加入和退出聊天

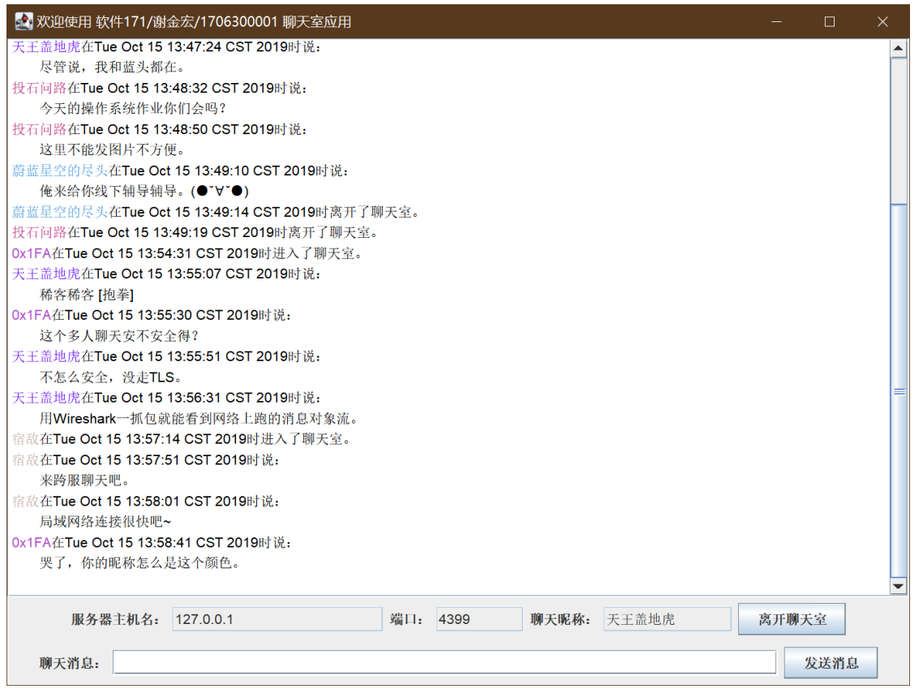

Figure 5 聊天进行了一会

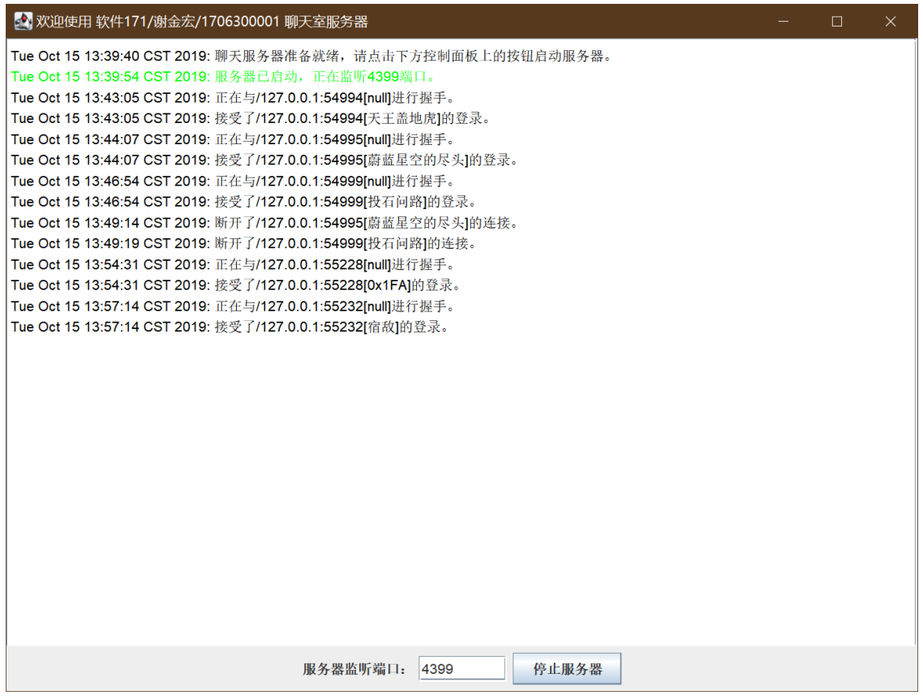

Figure 6 服务器端日志