Java gui 与网络编程：基于 tcp 协议，springboot，mybatis，swing 的网络聊天室

2025 年 5 月 20 日

一、设计思路：

1. 客户端思路：

将整个聊天室的客户端拆分为前端和后端，前端使用 swing 框架完成，类名：MainFrame，将后端的客户端 client 对象组合进 MainFrame（虽然我认为 client 应该声明为 MainFrame 中的内部类，但 client 的代码体积较大，且刚开始项目的时候先写的 client，所以 client 就被单独写在一个文件里了）

当 MainFrame 的主函数被调用的时候，主界面先被隐藏，弹出登录页面，当按下登录按钮后会为 MainFrame 中的 client 进行构造，如果登陆失败即 client 构造过程抛出异常，则会弹出登陆失败的对话框

client 构造时会给服务端发送请求，服务端会返回一个 user 对象，即为登录的用户对象，如果某个用户还没注册就会自动注册，如果返回的用户 id 为-2（即密码错误）则会抛出登陆失败异常

登录成功后，client 会开启一个线程处理服务端传来的消息，登录框被关闭，设置主窗口可见，此时主窗口的 windowListener 会被触发，进行初始化 ui 界面

ui 界面的左边是用户按钮（除当前用户，因为网络聊天室不能自言自语），初始化用户按钮的时候 client 会给服务端发请求，获得所有除自己的用户，当点击按钮后，mainFrame 会记录自己目前在和哪个用户对话，且 client 向服务端发请求，获得 MessageDTO 的列表对象并展示在右边，MessageDTO 是消息类

按下发送信息按钮后，该客户端会给服务端发消息，服务端给目标客户端转发请求，目标客户端进行 ui 回显（如果目标用户正在和发信息的用户

聊天，则会刷新目标用户的聊天页面，如果没有在聊天，则会有一个红点提示）

client 处理的核心逻辑：

client 共会发送四种请求，每一种请求都会在发送的字符串的第一个空格前声明，使用 printwriter 封装 socket 输出流进行请求的发送，当需要服务端返回数据的时候，使用阻塞队列进行获取，client 开启的处理线程在接收到服务端信息的时候会将对应信息放入阻塞队列

client 的处理线程：

client 的处理线程会一直试图从服务端读取一个 json 字符串（本来使用的对象流，但发现出现了一些问题，在服务端的部分会说到，最后决定改用将对象封装为 json 字符串的形式用字符流传输），我将返回的数据封装在 response 类中，response 中有一个枚举类型字段标识传来的是什么对象，根据该字段分别处理，并将处理结果放入阻塞队列中，但此处比较特殊的是 SINGLE_MESSAGE（因为其他的类型都是客户端主动发请求要求服务端返回信息，但是这个是另一个客户端给这个客户端发的信息，是被动的接收信息，这里有一个 bug 修了好久才修明白，在介绍完 server 后会解释）

2. 服务端设计：

将服务端声明为 springboot 的 component，这样当 springboot 主类运行起来的时候服务端会自动运行，且可以享受 springboot 的 Ioc 容器功能，将 mapper 由 springboot 注入进来，之后所有的数据库操作都由注入进来的 mapper 完成

server 每连接到一个客户端，都会先进行登陆操作，如果登录失败则不进行接下来的操作，如果登录成功则开启一个新线程专门处理该客户端发送的信息，用一个 ConcurrentHashMap 记录所有客户端的字符输出流，在一个客户端向另一个客户端发消息的时候服务端就会从这个线程安全哈希表中获取目标客户端的输出流来进行输出

server 处理客户端的线程统一接收字符串，在第一个空格前面的是请求的数据方式，后面的是携带的参数，统一写出 json 字符串

二. 出现的核心 bug 及修复方案

（1）对象序列化解析端请求头错误：一开始我采用对象序列化的方式响应数据，但出现了对象解析异常导致 socket 异常关闭，原因为在一个客户端向另一个客户端发消息的时候，服务端在发消息的客户端的管理线程中新建了一个目标客户端的对象写出流输出消息，同时目标客户端的管理线

程又写入了数据导致解析错误

（2）在某个客户端向另一个客户端发信息的时候，按下发送按钮后会阻塞在原地无法进行操作

以下是引发错误的核心代码

```java
        public void getMessage(Integer id)  {
                if (Objects.equals(id, nowId)) {
                        loadAllMessages();
                } else {
                        // 添加未读提示
                        SwingUtilities.invokeLater(() -> {
                                if (unreadIds.add(id)) {
                                        updateButtonUnreadStatus
                                            (id, true);
                                }
                        });
                }
        }


        private void loadAllMessages() {
                List<ChatMessage> messages = getAllMessages();
                updateMessageDisplay(messages);
        }

        private List<ChatMessage> getAllMessages()  {
                List<ChatMessage> messages = new ArrayList<>();
                List<MessageDTO> messageDTOS=client.getMessage(
                    nowId);
                for (int i = 0; i < messageDTOS.size(); i++) {
                        messages.add(new ChatMessage(
                        messageDTOS.get(i).getSendUsername(),
                        messageDTOS.get(i).getSendTime(),
                        messageDTOS.get(i).getMessage()
                        ));
                }
                return messages;
        }
```

以上是 MainFrame 中出现该 bug 的核心代码，getMessage 从客户端得到服务端发来的 id 后会调用 loadAllMessage，该函数会调用客户端的 getMessage 函数向服务端发送请求

```java
        Thread thread=new Thread(()->{
                try(BufferedReader br=new BufferedReader(new
                    InputStreamReader(clientSocket.
                    getInputStream())))
                {
                        while(true){
                                String json=br.readLine();
                                Response response=JSONObject.
                                    parseObject(json,Response.
                                    class);
                                System.out.println(response.
                                    getData().getClass());
                                switch(response.getType())
                                {
                                        case SINGLE_MESSAGE ->
                                            mainFrame.
                                            getMessage((Integer
                                            )response.getData()
                                            );
                                        case MESSAGE_LIST -> {
                                List<MessageDTO> messages =
                                    JSONArray.parseArray(
                                    response.getData().toString
                                    (), MessageDTO.class);
                                messageQueue.put(messages);
                                        }
                                        case USER_LIST -> {
                                List<User> userList =
                                    JSONArray.parseArray(
                                    response.getData().toString
                                    (),User.class);
                                userQueue.put(userList);
                                        }
                                }
                        }
```

```java
            }
            catch(IOException e){
                System.out.println(clientSocket.isClosed());
                e.printStackTrace();
                //throw new RuntimeException();
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
    });


    public List<MessageDTO> getMessage(Integer nowId) {
        try {
            PrintWriter pw = new PrintWriter(new OutputStreamWriter(clientSocket.getOutputStream()), true);
            pw.println("getmessage␣" + user.getId() + "␣" + nowId);

            // 阻塞直到队列中有数据
            return messageQueue.take();
        } catch (IOException | InterruptedException e) {
            throw new RuntimeException(e);
        }
    }
```

以上为客户端出现 bug 的核心代码（服务端响应数据没有问题所以就不贴服务端的代码了）

当服务端发来 SINGLE_MESSAGE 的信息后会调用 MainFrame 的 getMessage，当得到的 id 和窗口正在对话的用户的 id 相同时，就会调用 loadAllMessage 刷新聊天页面，loadAllMessage 会向服务端发请求来获得聊天记录，然而当服务端将数据反馈回来的时候，客户端处理服务端信息的线程里依旧阻塞在那个 case 语句（因为 mainFrame 的 getMessage 并未结束），由于 getMessage 阻塞在这里导致客户端无法进入下一个循环解析服务端传来的信息，而无法解析服务端传来的信息则会导致 getAllMessage 阻塞，从而继续阻塞住 getMessage，导致发生死锁，解决方案如下

```java
        public void getMessage(Integer id)  {
                if (Objects.equals(id, nowId)) {
                        new Thread(this::loadAllMessages).start
                            ();
                } else {
                        // 添加未读提示
                        SwingUtilities.invokeLater(() -> {
                                if (unreadIds.add(id)) {
                                        updateButtonUnreadStatus
                                            (id, true);
                                }
                        });
                }
        }
```

用一个新线程处理 loadAllMessage

三. 心得与收获：

1. 为了避免网络编程所带来的多线程在同一个 socket 的对象流中写入信息导致客户端无法正常解析数据等问题，应该尽量只包装一次 socket 的流，当其他线程需要用的时候去线程安全的哈希表中取即可

2. 线程安全是很重要的，在多线程开发中应该尽量多使用阻塞队列等容器来进行信息的生产-消费机制，使用 ConcurrentHashMap 等线程安全的容器来保存信息，总之，在多线程开发中应该尽量多使用线程安全的容器

3. 使用 springboot 集成的 mybatis 框架操作数据库是极其简便的，但要注意不要随意集成 springboot 框架（一开始做的思路是让 client 也集成进 springboot，拥有自己查询数据库的能力，但后来发现由于 client 需要多开，使其集成进 springboot 是极其不容易的，因为在 server 集成进 springboot 后，每次运行 client 都会导致 server 创建新实例，遂完全没有办法使 client 集成进 springboot）

4. 要遵守单一职责原则以及类的设计理念，查数据库返回数据的操作本来就是服务端的工作，不应该把这些工作交给客户端来做，且真实的情况里也没有客户端查数据库的说法和可能性，设计方向走错了很可能需要很久才能找回正确的路线

5. 在遇到 bug 的时候要多想几层，用断点调试快速定位 bug 出现的代码块和触发条件，遇到死锁等无法通过断点调试解决的 bug 时，先定位到

死锁发生的大概位置，然后思考一下函数的调用栈，或许会有不一样的收获

6. 千万不要在 catch 中写空体，否则会浪费很多时间去查找很容易就能发现的 bug

四. 源代码

```java
package org.example.onlinecontact.client.login;

import org.example.onlinecontact.exception.LoginErrorException;
import org.example.onlinecontact.mapper.UserMapper;
import org.example.onlinecontact.pojo.User;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.springframework.util.DigestUtils;

import java.util.Scanner;

@Component // 标记为 Spring 组件
public class Login {
    @Autowired
    private UserMapper userMapper; // 非静态字段，由 Spring 注入

    public User login(String username, String password) {
        password = DigestUtils.md5DigestAsHex(password.getBytes());
        User user = User.builder().passWord(password).userName(username).build();

        user=userMapper.login(user);
        if (user != null) {
            return user;
        } else {
            user = User.builder().passWord(password).userName(username).build();
            if(userMapper.getByUserName(username)!=0){user.setId(-2); return user;}
            userMapper.insert(user);
            return user;
        }
    }
}
```

```
package org.example.onlinecontact.client;

import com.alibaba.fastjson2.JSONArray;
import com.alibaba.fastjson2.JSONObject;
import org.example.onlinecontact.dto.MessageDTO;
import org.example.onlinecontact.exception.LoginErrorException;
import org.example.onlinecontact.frame.MainFrame;
import org.example.onlinecontact.pojo.Response;
import org.example.onlinecontact.pojo.User;


import java.io.*;
import java.net.Socket;
import java.util.List;
import java.util.Scanner;
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.LinkedBlockingQueue;


public class Client {
    public static final String serverAddress = "localhost";
    public static final int serverPort = 8888;
    public Socket clientSocket;
    public User user;

    public MainFrame mainFrame;

    private final BlockingQueue<List<MessageDTO>> messageQueue
        = new LinkedBlockingQueue<>();
    private final BlockingQueue<List<User>> userQueue = new
        LinkedBlockingQueue<>();

    public Client(String userName,String password,MainFrame
        mainFrame) throws IOException, ClassNotFoundException {


```

```java
        clientSocket = new Socket(serverAddress, serverPort);
        this.mainFrame = mainFrame;
        String login="login␣"+userName+"␣"+password;

        PrintWriter pw=new PrintWriter(new OutputStreamWriter(clientSocket.getOutputStream()),true);
        pw.println(login);
        User user=null;
        ObjectInputStream ois=new ObjectInputStream(clientSocket.getInputStream());

        user=(User) ois.readObject();
        if(user.getId()==-2) throw new LoginErrorException("密码错误");
        this.user = user;
        Thread thread=new Thread(()->{
            try(BufferedReader br=new BufferedReader(new InputStreamReader(clientSocket.getInputStream())))
            {
                while(true){
                    String json=br.readLine();
                    Response response=JSONObject.parseObject(json,Response.class);
                    System.out.println(response.getData().getClass());
                    switch(response.getType())
                    {
                        case SINGLE_MESSAGE -> mainFrame.getMessage((Integer)response.getData());
                        case MESSAGE_LIST -> {
                            List<MessageDTO> messages = JSONArray.parseArray(response.getData().toString(), MessageDTO.class);
                            messageQueue.put(messages);
                        }
                        case USER_LIST -> {
                            List<User> userList =  JSONArray.
                                parseArray(response.getData().
                                toString(),User.class);
                            userQueue.put(userList);
                        }
                    }
                }
            }
            catch(IOException e){
                System.out.println(clientSocket.isClosed());
                e.printStackTrace();
                //throw new RuntimeException();
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        });
        thread.setName(user.getUserName());
        thread.setDaemon(true);
        thread.start();

    }
    public Integer getId() throws IOException,
        ClassNotFoundException {
        Scanner scanner = new Scanner(System.in);
        String username = scanner.nextLine();
        String password = scanner.nextLine();
        String login="login␣"+username+"␣"+password;
        PrintWriter pw=new PrintWriter(new OutputStreamWriter(
            clientSocket.getOutputStream()),true);
        pw.println(login);
        User user=null;
        ObjectInputStream ois=new ObjectInputStream(
            clientSocket.getInputStream());
        while((user= (User) ois.readObject())==null){}
        if(user.getId()==-2) throw new LoginErrorException("密
            码错误");
        return user.getId();
    }

```

```java
    public List<User> getUser()  {
        try
        {
            PrintWriter pw=new PrintWriter(new
                OutputStreamWriter(clientSocket.getOutputStream
                ()),true);
            pw.println("getUser␣"+user.getId());
            return userQueue.take();

        } catch (IOException e) {
            throw new RuntimeException(e);
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }

    public List<MessageDTO> getMessage(Integer nowId) {
        try {
            PrintWriter pw = new PrintWriter(new
                OutputStreamWriter(clientSocket.getOutputStream
                ()), true);
            pw.println("getmessage␣" + user.getId() + "␣" +
                nowId);

            // 阻塞直到队列中有数据
            return messageQueue.take();
        } catch (IOException | InterruptedException e) {
            throw new RuntimeException(e);
        }
    }

    public void sendMessage(String s) {
        try{
            PrintWriter pw=new PrintWriter(new
                OutputStreamWriter(clientSocket.getOutputStream
                ()),true);
            pw.println(s);
        }
        catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
}
```

```
package org.example.onlinecontact.dto;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;

import java.io.Serializable;
import java.time.LocalDateTime;
@Data
@Builder
@AllArgsConstructor
public class MessageDTO implements Serializable {
    private String sendUsername;
    private String getUsername;
    private String message;
    private LocalDateTime sendTime;
    private Integer sendUserId;
    private Integer getUserId;
}
```

```
package org.example.onlinecontact.exception;

public class LoginErrorException extends RuntimeException {
    public LoginErrorException(String message) {
        super(message);
    }
}
```

```
package org.example.onlinecontact.frame;

import org.example.onlinecontact.client.Client;
import org.example.onlinecontact.client.login.Login;
import org.example.onlinecontact.exception.LoginErrorException;

```

```java
import javax.swing.*;
import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.IOException;

public class LoginDialog extends JDialog {


    Integer id=null;
    public LoginDialog(MainFrame frame) {
        super(frame, "Login",true);

        setSize(350,200);
        setLocationRelativeTo(null);

        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        //container = getContentPane();

        JPanel panel = new JPanel();
        // 添加面板
        addWindowListener(new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                if(id==null)
                {
                    Window window1 = SwingUtilities.
                        getWindowAncestor(LoginDialog.this);
                    window1.dispose();
                }
            }
        });
        placeComponents(panel);
        add(panel);
        setResizable(false);

        setVisible(true);
    }
    private void placeComponents(JPanel panel) {

        /* 布局部分我们这边不多做介绍
         * 这边设置布局为 null
         */
        panel.setLayout(null);

        // 创建 JLabel
        JLabel userLabel = new JLabel("User:");
        /* 这个方法定义了组件的位置。
         * setBounds(x, y, width, height)
         * x 和 y 指定左上角的新位置，由 width 和 height 指定新
             的大小。
         */
        userLabel.setBounds(10,20,80,25);
        panel.add(userLabel);

        /*
         * 创建文本域用于用户输入
         */
        JTextField userText = new JTextField(20);
        userText.setBounds(100,20,165,25);
        panel.add(userText);

        // 输入密码的文本域
        JLabel passwordLabel = new JLabel("Password:");
        passwordLabel.setBounds(10,50,80,25);
        panel.add(passwordLabel);

        /*
         *这个类似用于输入的文本域
         * 但是输入的信息会以点号代替，用于包含密码的安全性
         */
        JPasswordField passwordText = new JPasswordField(20);
        passwordText.setBounds(100,50,165,25);
        panel.add(passwordText);

        // 创建登录按钮
        JButton loginButton = new JButton("login");
        loginButton.setBounds(10, 80, 80, 25);
        loginButton.addActionListener(e -> {
            String user = userText.getText();
            String password = passwordText.getText();
            if(user!=null && password!=null) {
                try{
                    MainFrame mainFrame = (MainFrame) getParent
                        ();
                    Client client=new Client(user,password,
                        mainFrame);
                    Window window = SwingUtilities.
                        getWindowAncestor((Component) e.
                        getSource());
                    Window window1 = SwingUtilities.
                        getWindowAncestor(LoginDialog.this);
                    mainFrame.setUser(client.user);
                    mainFrame.setCLient(client);
                    window1.setVisible(true);
                    window.dispose();


                }
                catch(LoginErrorException ex){
                    JOptionPane.showMessageDialog(null, ex.
                        getMessage());
                } catch (IOException ex) {
                    throw new RuntimeException(ex);
                } catch (ClassNotFoundException ex) {
                    throw new RuntimeException(ex);
                }
            }
        });
        panel.add(loginButton);

    }
    /*public Integer getId()
    {
        return id;
    }*/
}
```

```java
package org.example.onlinecontact.frame;

import org.example.onlinecontact.client.Client;
import org.example.onlinecontact.dto.MessageDTO;
import org.example.onlinecontact.pojo.User;

import javax.swing.*;


import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.IOException;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.*;
import java.util.List;

public class MainFrame extends JFrame {
    private JPanel userPanel;
    private User user;
    private Client client;

    // 聊天界面组件
    private JPanel messagePanel;
    private JTextArea txtInput;
    private JButton btnSend;

    private HashMap<JButton,Integer> buttonId;
    private HashMap<Integer,JButton> idButton;

    private Set<Integer> unreadIds = new HashSet<>();
    private HashMap<Integer, String> originalButtonTexts = new
        HashMap<>();

    private Integer nowId;
    public MainFrame() {
        super("Online␣Contact");
        //setDefaultUser();
        addWindowListener(new WindowAdapter() {
            @Override
            public void windowOpened(WindowEvent e) {

                initializeUI();
            }
        });

        //loadAllMessages();
        setVisible(false);
    }

    private void initializeUI() {
        configureWindow();
        createLeftPanel();
        createRightPanel();
        createWelcomeLabel();
    }

    private void configureWindow() {
        setLayout(new BorderLayout());
        setSize(1000, 650);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    private void createLeftPanel() {
        JPanel leftPanel = new JPanel(new BorderLayout());
        leftPanel.setPreferredSize(new Dimension(250, getHeight
            ()));

        userPanel = new JPanel(new GridBagLayout());
        JScrollPane scrollPane = new JScrollPane(userPanel);
        loadUserList();

        leftPanel.add(scrollPane, BorderLayout.CENTER);
        add(leftPanel, BorderLayout.LINE_START);
    }

    private void createRightPanel() {
        JPanel rightPanel = new JPanel(new BorderLayout());
        rightPanel.setBorder(BorderFactory.createTitledBorder("
            聊天区域"));

        JScrollPane messageScroll = createMessageScroll();
        rightPanel.add(messageScroll, BorderLayout.CENTER);
        rightPanel.add(createInputArea(), BorderLayout.SOUTH);
        add(rightPanel, BorderLayout.CENTER);

        addWindowListener(new WindowAdapter() {
            @Override
            public void windowOpened(WindowEvent e) {
                createWelcomeLabel();
                scrollToBottom();
            }
        });
    }

    private JScrollPane createMessageScroll() {
        messagePanel = new JPanel();
        messagePanel.setLayout(new BoxLayout(messagePanel,
            BoxLayout.Y_AXIS));

        JScrollPane scrollPane = new JScrollPane(messagePanel);
        scrollPane.getVerticalScrollBar().addAdjustmentListener
            (e -> {
            if (!e.getValueIsAdjusting()) {
                e.getAdjustable().setValue(e.getAdjustable().
                    getMaximum());
            }
        });
        return scrollPane;
    }

    private JPanel createInputArea() {
        JPanel panel = new JPanel(new BorderLayout());
        panel.setBorder(BorderFactory.createEmptyBorder(5, 5,
```

```java
            5, 5));

        // 输入区域占整个聊天区的1/4高度
        panel.setPreferredSize(new Dimension(0, getHeight()/4));

        txtInput = new JTextArea();
        txtInput.setLineWrap(true);
        txtInput.setFont(new Font("微软雅黑", Font.PLAIN, 14));

        // 带滚动条的输入框
        JScrollPane inputScroll = new JScrollPane(txtInput);
        inputScroll.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED);

        // 发送按钮面板（保持按钮高度自适应）
        JPanel buttonPanel = new JPanel(new BorderLayout());
        btnSend = new JButton("发送");
        btnSend.addActionListener(e -> sendMessage());
        btnSend.setPreferredSize(new Dimension(80, 0)); // 宽度固定，高度自适应
        buttonPanel.add(btnSend, BorderLayout.CENTER);

        // 组合布局
        panel.add(inputScroll, BorderLayout.CENTER);
        panel.add(buttonPanel, BorderLayout.EAST);

        return panel;
    }

    private void createWelcomeLabel() {
        JLabel welcomeLabel = new JLabel(
                "欢迎来到聊天室: " + user.getUserName(),
                JLabel.CENTER
        );
        welcomeLabel.setFont(new Font("微软雅黑", Font.BOLD, 20));
        welcomeLabel.setBorder(BorderFactory.createEmptyBorder(10, 0, 10, 0));
        add(welcomeLabel, BorderLayout.NORTH);
        revalidate();
    }

    private void loadUserList() {
/*        List<String> users = new ArrayList<>();*/
        List<User> user=client.getUser();
        /*for (int i = 0; i < user.size(); i++) {
            users.add(user.get(i).getUserName());
        }*/
        updateUserButtons(user);
    }

    private void updateUserButtons(List<User> users) {
        userPanel.removeAll();
        GridBagConstraints gbc = new GridBagConstraints();
        gbc.gridx = 0;
        gbc.fill = GridBagConstraints.HORIZONTAL;
        gbc.weightx = 1.0;
        gbc.anchor = GridBagConstraints.NORTH;
        gbc.insets = new Insets(3, 3, 3, 3);

        // 初始化映射关系
        buttonId = new HashMap<>();
        idButton = new HashMap<>();
        originalButtonTexts.clear(); // 清空原始文本缓存

        for (int i = 0; i < users.size(); i++) {
            User currentUser = users.get(i);
            gbc.gridy = i;

            // 创建按钮时检查是否有未读消息
            boolean hasUnread = unreadIds.contains(currentUser.
                getId());
            JButton btn = createUserButton(
                    hasUnread ?
                            addUnreadDot(currentUser.
                                getUserName()) :
                            currentUser.getUserName()
            );

            userPanel.add(btn, gbc);

            // 保存映射关系
            buttonId.put(btn, currentUser.getId());
            idButton.put(currentUser.getId(), btn);
            originalButtonTexts.put(currentUser.getId(),
                currentUser.getUserName());

            // 按钮点击事件
            btn.addActionListener(e -> {
                nowId = buttonId.get(btn);

                // 清除未读状态
                if (unreadIds.remove(nowId)) {
                    updateButtonUnreadStatus(nowId, false);
                }

                loadAllMessages();
            });

            // 最后一个元素设置垂直权重
            gbc.weighty = (i == users.size() - 1) ? 1.0 : 0.0;
        }

        // 添加顶部对齐的占位符
        userPanel.add(Box.createVerticalGlue(), gbc);

        userPanel.revalidate();
        userPanel.repaint();
    }

    // 辅助方法：为按钮文本添加红点
    private String addUnreadDot(String originalText) {
        return "<html>" + originalText + "␣<font␣color='red
            '>•</font></html>";
    }
    private void updateButtonUnreadStatus(Integer id, boolean
        hasUnread) {
        JButton button = idButton.get(id);
        if (button != null) {
            String originalText = originalButtonTexts.get(id);
            if (hasUnread) {
                button.setText("<html>" + originalText + "␣<
                    font␣color='red'>•</font></html>");
            } else {
                button.setText(originalText);
            }
        }
    }

    private JButton createUserButton(String text) {
        JButton btn = new JButton(text);
        btn.setFont(new Font("微软雅黑", Font.PLAIN, 14));
        btn.setFocusPainted(false);
        btn.setPreferredSize(new Dimension(200, 40));
        btn.setMaximumSize(new Dimension(Integer.MAX_VALUE, 40)
            );
        return btn;
    }

    private void loadAllMessages() {
        List<ChatMessage> messages = getAllMessages();
        updateMessageDisplay(messages);
    }

    private List<ChatMessage> getAllMessages()  {
        List<ChatMessage> messages = new ArrayList<>();
        List<MessageDTO> messageDTOS=client.getMessage(nowId);
        for (int i = 0; i < messageDTOS.size(); i++) {
            messages.add(new ChatMessage(
                    messageDTOS.get(i).getSendUsername(),
                    messageDTOS.get(i).getSendTime(),
                    messageDTOS.get(i).getMessage()
            ));
        }
        return messages;
```

```java
    }

    private void updateMessageDisplay(List<ChatMessage> messages) {
        messagePanel.removeAll();
        messages.forEach(msg -> {
            messagePanel.add(createMessagePanel(msg));
            messagePanel.add(Box.createVerticalStrut(8));
        });
        messagePanel.revalidate();
        messagePanel.repaint();
        scrollToBottom();
    }

    private JPanel createMessagePanel(ChatMessage msg) {
        JPanel panel = new JPanel();
        panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
        panel.setBorder(BorderFactory.createCompoundBorder(
                BorderFactory.createMatteBorder(0, 0, 1, 0, Color.LIGHT_GRAY),
                BorderFactory.createEmptyBorder(5, 10, 5, 10)
        ));

        JPanel header = new JPanel(new FlowLayout(FlowLayout.LEFT, 0, 0));
        JLabel sender = new JLabel(msg.getSender());
        sender.setFont(new Font("微软雅黑", Font.BOLD, 12));

        JLabel time = new JLabel("  " + formatTime(msg.getTime()));
        time.setFont(new Font("微软雅黑", Font.PLAIN, 10));
        time.setForeground(Color.GRAY);

        header.add(sender);
        header.add(time);

        JTextPane content = new JTextPane();
        content.setText(msg.getContent());
        content.setEditable(false);
        content.setBackground(UIManager.getColor("Panel.
            background"));

        panel.add(header);
        panel.add(content);
        return panel;
    }

    private void scrollToBottom() {
        JScrollBar vertical = ((JScrollPane)messagePanel.
            getParent().getParent())
                .getVerticalScrollBar();
        vertical.setValue(vertical.getMaximum());
    }

    private void sendMessage() {
        String content = txtInput.getText().trim();
        if (!content.isEmpty()) {
            // 实际发送消息逻辑
            txtInput.setText("");
            client.sendMessage("sendMessage␣"+content+"␣"+nowId
                );
            loadAllMessages();
        }
    }

    private String formatTime(LocalDateTime time) {
        return time.format(DateTimeFormatter.ofPattern("yyyy-MM
            -dd␣HH:mm:ss"));
    }

    private void setDefaultUser() {
        user = User.builder().userName("zhangsan").id(4).build
            ();
        try {
            client = new Client("zhangsan","123456",this);
        } catch (IOException | ClassNotFoundException e) {
            JOptionPane.showMessageDialog(this, "连接服务器失败
                ");
        }
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> new LoginDialog(new
            MainFrame()));
    }

    static class ChatMessage {
        private final String sender;
        private final LocalDateTime time;
        private final String content;

        public ChatMessage(String sender, LocalDateTime time,
            String content) {
            this.sender = sender;
            this.time = time;
            this.content = content;
        }

        public String getSender() { return sender; }
        public LocalDateTime getTime() { return time; }
        public String getContent() { return content; }
    }
    void setUser(User user) {
        this.user = user;
    }
    void setCLient(Client client) {
        this.client = client;
    }
    public void getMessage(Integer id)  {
        if (Objects.equals(id, nowId)) {
            new Thread(this::loadAllMessages).start();
        } else {
            // 添加未读提示
            SwingUtilities.invokeLater(() -> {
                if (unreadIds.add(id)) {
                    updateButtonUnreadStatus(id, true);
                }
            });
        }
    }
}
```

```
package org.example.onlinecontact.mapper;

import com.github.pagehelper.Page;
import org.apache.ibatis.annotations.Insert;
import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Select;
import org.example.onlinecontact.dto.MessageDTO;

@Mapper
public interface MessageMapper {
    @Insert("insert␣into␣message(send_username,␣get_username,␣message,␣send_time,get_user_id,send_user_id)␣VALUES␣" +
            "(#{sendUsername},#{getUsername},#{message},#{sendTime},#{getUserId},#{sendUserId})")
    void insert(MessageDTO messageDTO);
    @Select("select␣*␣from␣message␣where␣(send_user_id=#{id}␣and␣get_user_id=#{sendUserId})␣or" +
            "(send_user_id=#{sendUserId}␣and␣get_user_id=#{id})␣␣")
    Page<MessageDTO> page(Integer id, Integer sendUserId);
}
```

```
package org.example.onlinecontact.mapper;

import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Select;
import org.example.onlinecontact.pojo.User;

import java.util.ArrayList;

@Mapper
public interface UserMapper {

```

```java

    @Select("select␣*␣from␣user␣where␣user_name=#{userName}␣and␣password=#{passWord}")
    User login(User user);

    void insert(User user);
    @Select("select␣count(*)␣from␣user␣where␣user_name=#{username}")
    Integer getByUserName(String username);
    @Select("select␣user_name␣from␣user␣where␣id=#{id}")
    String getUsernameById(Integer id);
    @Select("select␣*␣from␣user␣where␣id!=#{id}")
    ArrayList<User> getExcludeId(Integer id);
}
```

```java
package org.example.onlinecontact.pojo;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.io.Serializable;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class Response implements Serializable {
    private static final long serialVersionUID = 1L;
    private ResponseType type;
    private Object data;
    public enum ResponseType {
        USER_LIST, MESSAGE_LIST, SINGLE_MESSAGE, LOGIN_RESULT
    }
}
```

```java
package org.example.onlinecontact.pojo;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;

import java.io.Serializable;

@Data
@Builder
@AllArgsConstructor
public class User implements Serializable {
    private Integer id;
    private String userName;
    private String passWord;
}
```

```java
package org.example.onlinecontact.server;

import com.alibaba.fastjson2.JSONObject;
import org.example.onlinecontact.dto.MessageDTO;
import org.example.onlinecontact.mapper.MessageMapper;
import org.example.onlinecontact.mapper.UserMapper;
import org.example.onlinecontact.pojo.Response;
import org.example.onlinecontact.pojo.User;

import java.io.*;
import java.net.Socket;
import java.net.SocketException;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

public class ClientHandlerThread extends Thread {
    private final Socket socket;
    private final Map<Integer, Socket> clients;
    private final Integer clientId;
    private final MessageMapper messageMapper;
    private final UserMapper userMapper;
    private static ConcurrentHashMap<Integer, PrintWriter>
        userMap;
    private volatile boolean isRunning = true; // 控制循环的状态标志
    static{
        userMap = new ConcurrentHashMap<>();
    }
    public ClientHandlerThread(
            Socket socket,
            Map<Integer, Socket> clients,
            Integer clientId,
            MessageMapper messageMapper,
            UserMapper userMapper) {
        this.socket = socket;
        this.clients = clients;
        this.clientId = clientId;
        this.messageMapper = messageMapper;
        this.userMapper = userMapper;
        this.setDaemon(true); // 设置为守护线程
        setName(clientId.toString());
    }

    @Override
    public void run() {

        try(BufferedReader reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            PrintWriter out = new PrintWriter(new OutputStreamWriter(socket.getOutputStream()), true);)
        {
            userMap.put(clientId, out);
            while (isRunning && !socket.isClosed()) { // 检查 Socket 是否已关闭

                String message = reader.readLine();
                if (message == null) { // 客户端主动关闭连接
                    continue;
                }
                String[] messages = message.split("␣");
                if(messages[0].equals("getUser"))
                {
                    List<User> users = selectUser(Integer.parseInt(messages[1]));
                    out.println(JSONObject.toJSONString(new Response(Response.ResponseType.USER_LIST,users)));
                    continue;
                }
                if(messages[0].equals("getmessage"))
                {
                    List<MessageDTO> messageDTOS=getMessage(Integer.parseInt(messages[1]),Integer.parseInt(messages[2]));
                    out.println(JSONObject.toJSONString(new Response(Response.ResponseType.MESSAGE_LIST,messageDTOS)));

                    continue;
                }
                if(messages[0].equals("sendMessage"))
                {
                    int targetUserId;
                    try {
                        targetUserId = Integer.parseInt(messages[2]);
                    } catch (NumberFormatException e) {
                        System.err.println("目标用户ID格式错误:␣" + messages[2]);
                        continue;
                    }

                    String content = messages[1];
                    MessageDTO messageDTO = buildMessageDTO(clientId, targetUserId, content);
                    messageMapper.insert(messageDTO); // 存储消息到数据库

                    // 转发消息给目标用户
                    Socket targetSocket = clients.get(
                        targetUserId);
                    if (targetSocket != null && !targetSocket.isClosed()) {
                        PrintWriter printWriter = userMap.get(targetUserId);
                        printWriter.println(JSONObject.toJSONString(new Response(Response.ResponseType.SINGLE_MESSAGE, messageDTO.getSendUserId())));
                    } else {
                        System.out.println("目标用户 " + targetUserId + " 不在线");
                    }
                    System.out.println(userMap.get(targetUserId).toString());
                }

            }
        } catch (SocketException e) {
            System.out.println("客户端 " + clientId + " 连接异常断开: " + e.getMessage());
        } catch (IOException e) {
            System.err.println("客户端 " + clientId + " 处理异常: " + e.getMessage());
        } finally {
            //closeResources();
        }
    }

    private List<MessageDTO> getMessage(int sendId, int getId) {
        return messageMapper.page(sendId, getId);
    }

    // 安全关闭资源
    private void closeResources() {
        isRunning = false; // 停止循环
        try {
            if (socket != null && !socket.isClosed()) {
                socket.close();
            }
        } catch (IOException e) {
            System.err.println("关闭␣Socket␣失败:␣" + e.getMessage());
        }
        clients.remove(clientId); // 从在线列表中移除
        System.out.println("客户端␣" + clientId + "␣资源已释放");
    }

    // 构建消息DTO
    private MessageDTO buildMessageDTO(Integer senderId, Integer receiverId, String content) {
        return MessageDTO.builder()
                .sendUserId(senderId)
                .getUserId(receiverId)
                .message(content)
                .sendTime(LocalDateTime.now())
                .sendUsername(userMapper.getUsernameById(senderId))
                .getUsername(userMapper.getUsernameById(receiverId))
                .build();
    }
    private ArrayList<User> selectUser(Integer id) {
        return userMapper.getExcludeId(id);
    }
}
```

```java
package org.example.onlinecontact.server;

import jakarta.annotation.PostConstruct;
import jakarta.annotation.PreDestroy;
import org.example.onlinecontact.client.login.Login;
import org.example.onlinecontact.mapper.MessageMapper;
import org.example.onlinecontact.mapper.UserMapper;
import org.example.onlinecontact.pojo.User;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.HashMap;


@Component // 标记为 Spring 组件
public class Server {
    public static final int PORT = 8888;
    private ServerSocket serverSocket;
    private volatile boolean isRunning = true;
    private HashMap<Integer, Socket> clients = new HashMap<>();


    @Autowired
    MessageMapper messageMapper;
    @Autowired
    UserMapper userMapper;
    @Autowired
    Login login;
    @PostConstruct // Spring 容器初始化后自动调用
    public void start() {
        System.out.println("服务端已创建");
        new Thread(() -> {
            try {
                serverSocket = new ServerSocket(PORT);
                System.out.println("TCP␣服务器已启动，监听端口：
                    ␣" + PORT);
                while (isRunning) {
                    Socket socket = serverSocket.accept();
                    System.out.println(socket);
                    String loginString;
                    BufferedReader in = new BufferedReader(new
                        InputStreamReader(socket.getInputStream
                        ()));
                    while((loginString= in.readLine())==null){}
                    String[] s = loginString.split("␣");
                    User user=login(s[1],s[2]);
                    System.out.println("登陆成功：id="+user.getId());
                    ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStream());
                    out.writeObject(user);
                    out.flush();
                    if(user.getId()==-2){
                        System.out.println("登陆失败");
                        continue;
                    }

                    clients.put(user.getId(), socket);
                    new ClientHandlerThread(socket, clients, user.getId(),messageMapper,userMapper).start();
                }
            } catch (IOException e) {
                if (!isRunning) {
                    System.out.println("TCP␣服务器已正常关闭");
                } else {
                    e.printStackTrace();
                }
            }
        }).start();
    }



    @PreDestroy // Spring 容器销毁前自动调用
    public void stop() {
        isRunning = false;
        try {
            if (serverSocket != null && !serverSocket.isClosed()) {
                serverSocket.close();
            }
            // 关闭所有客户端连接
```

```java
                for (Socket socket : clients.values()) {
                    if (!socket.isClosed()) {
                        socket.close();
                    }
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

    public User login(String username, String password) {
        return login.login(username, password);
    }
}
```

```java
package org.example.onlinecontact;

import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.
    SpringBootApplication;

@SpringBootApplication
public class OnlineContactApplication implements
    CommandLineRunner {

    public static void main(String[] args) {
        SpringApplication.run(OnlineContactApplication.class,
            args);
    }

    @Override
    public void run(String... args) throws Exception {
        // 阻塞主线程，防止退出
        Thread.currentThread().join();
    }
}
```