浙江大学计算机科学与技术学院

Java 应用技术课程报告

2020—2021 学年 秋冬 学期

题目	“聊在浙大”多人即时通信软件开发
学号	3180103772
学生姓名	张溢弛
所在专业	软件工程
所在班级	软件工程 1801

目 录

一、引言

本次作业我开发的是一款类似于 QQ 聊天室的即时聊天通讯软件，包含客户端和服务端两个部分，其中客户端基于 Java 的 Swing 图形库进行界面的设计，服务端采用 MySQL 作为存储关键信息的数据库，使用 JDBC 技术进行数据库的连接，客户端和服务端基于 Java 的 socket 库进行通信，同时服务端多线程并发的设计，为每个客户端单独创建一个子线程来运行，支持多组访问。因此**本次作业实现了老师的所有要求以及 bonus 中的要求**。

具体的程序介绍将在下面的文档中具体展开。

1.1 设计目的

本次作业基于 Java Swing、socket、Java 的线程库以及 JDBC，具体功能和设计思路如下：

- 用户打开客户端软件之后输入账号密码，客户端将账号密码发送到服务端验证登录是否成功，如果账号密码错误就不能登录
- 客户端登录之后显示出聊天界面，用户可以查看历史消息记录和当前聊天室中的用户列表，也可以在文本框中输入和发送消息，消息框中将显示消息记录
- 服务端收到客户端登录的消息之后为每个客户端创建一个子线程用于消息的收发，并将每次新增的消息记录写入到数据库中
- 每个服务端的子线程不断监听 I/O 流中收到的消息，根据一定的格式对收到的数据进行解析并进行对应的操作
- 数据库使用 MySQL，并使用 JDBC 进行数据库驱动
- 客户端和服务端之间的通信基于 Socket
- 客户端退出的时候会发送登出信息并关闭 socket 连接

1.2 设计说明

本程序采用 Java 程序设计语言，在 JetBrains IDEA 平台下编辑、编译与调试。具体程序都是我一个人完成的，具体的工作安排如下图所示：

表 1 各成员分工表

成员名称	完成的主要工作	
	程序设计	**课程报告**
张溢弛	负责整个程序前期的需求分析和整体功能的架构 程序中棋盘的布局设计 程序后期的测试与运行	报告的第 1 章、第 2 章和第 4 章
张溢弛	负责程序中客户端的界面设计和代码编写，设计交互逻辑	报告的第 3 章
张溢弛	服务端的架构设计和具体代码编写，设计数据表、并发和通信协议	目录、总结和参考文献的整理 报告后期的格式设置

二、总体设计

2.1 功能模块设计

本程序需实现的主要功能有：

(1) 设计并实现客户端界面，包含登陆界面和聊天主界面

(2) 设计并实现服务端的并发架构，为每个客户端连接创建一个子线程

(3) 实现客户端和服务端的 socket 通信

(4) 实现服务端的数据库连接，采用 MySQL 和 JDBC

程序的总体功能如图 1 所示：

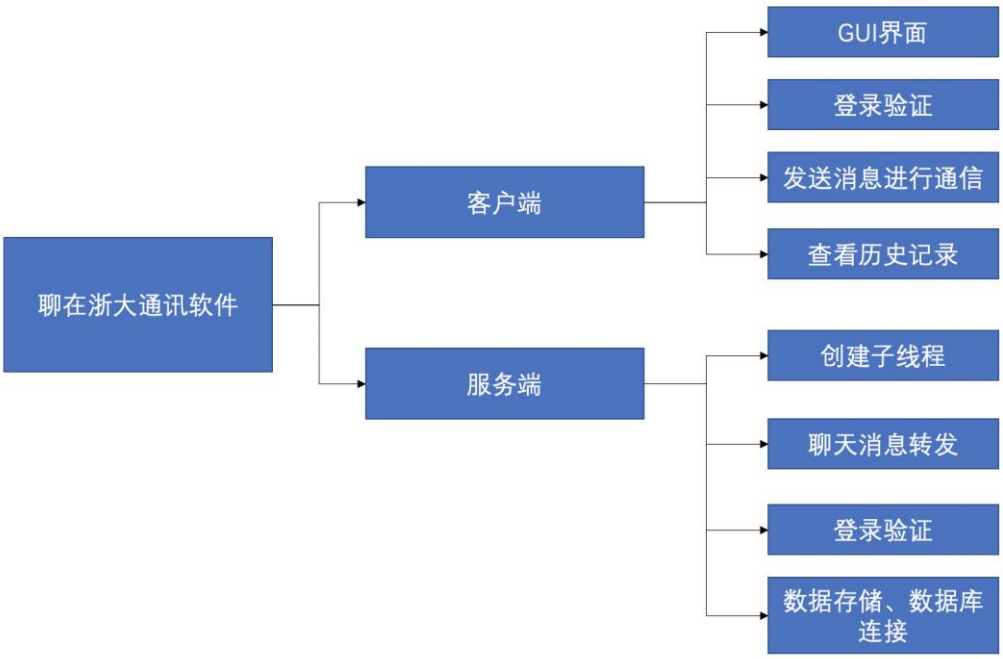
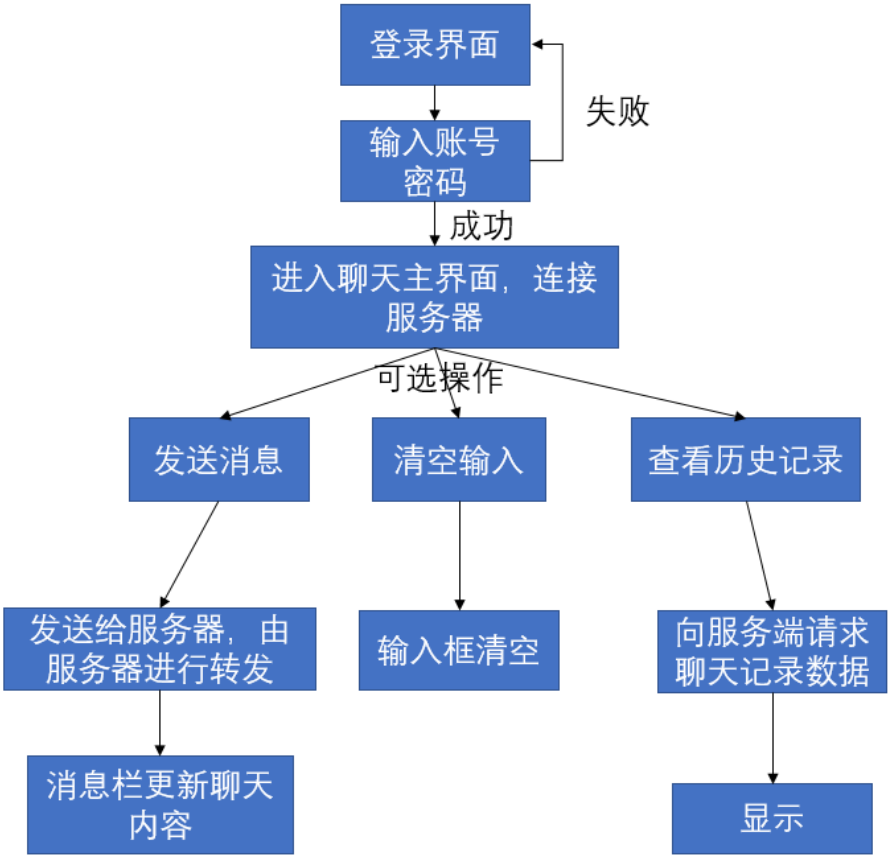


图 1 总体功能图

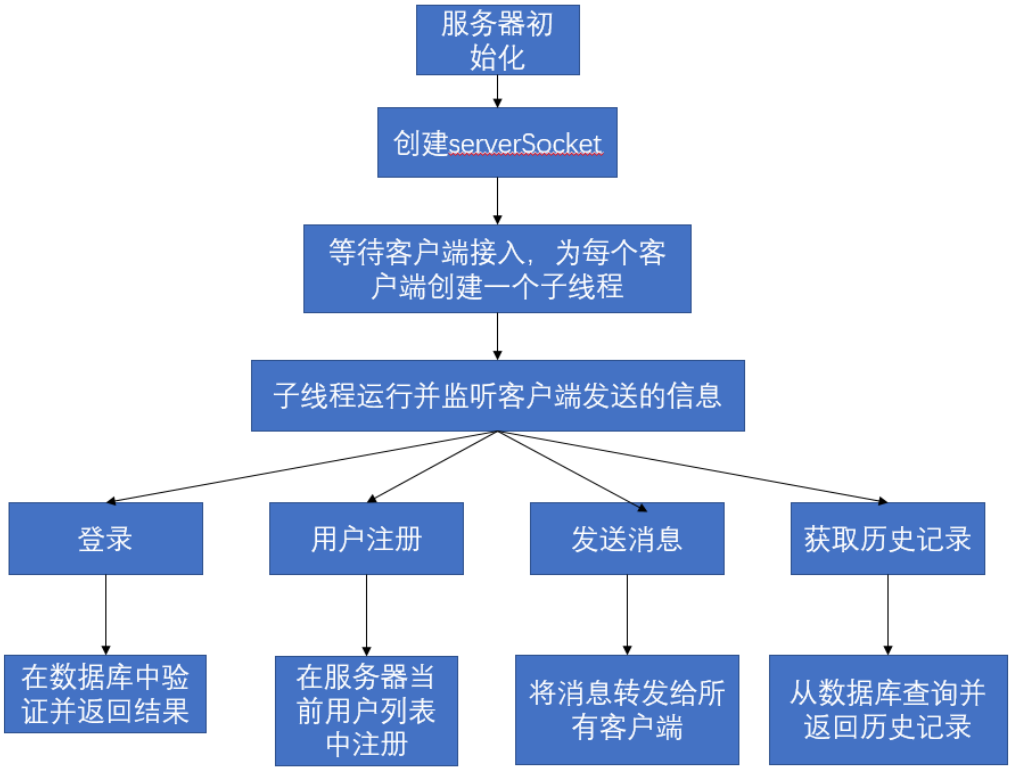
2.2 流程图设计

程序总体流程如下所示：

- 客户端的总体操作流程：

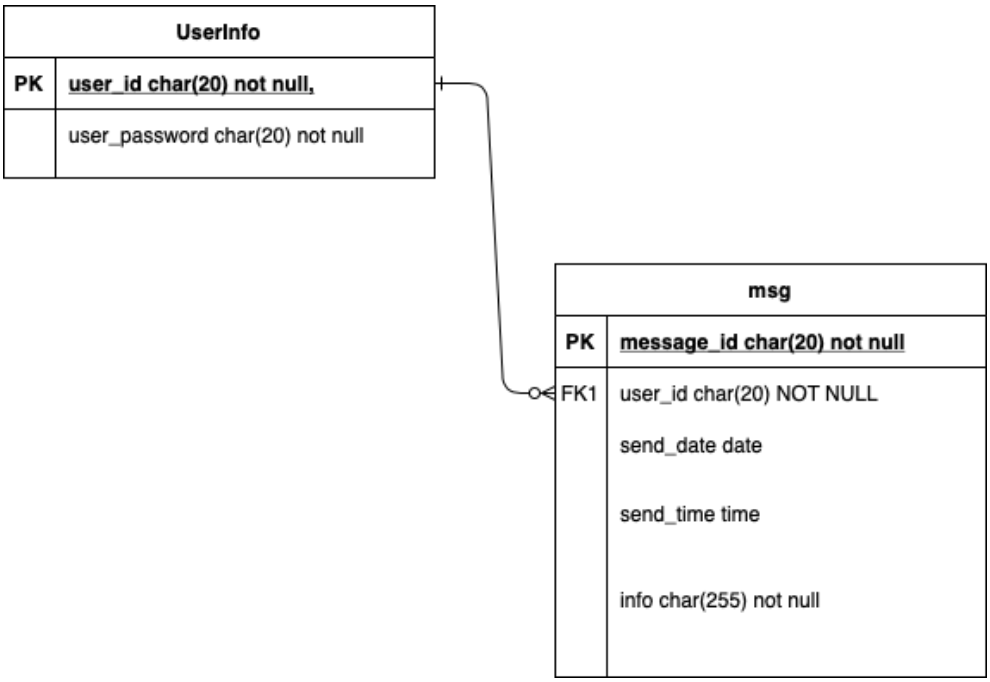


- 服务端的总体流程如图所示：
  - 此外还有关闭连接的服务，当服务端收到发送自客户端的退出登录请求之后就会关闭 I/O 流和 socket 连接，防止报错



2.3 数据库设计

本次作业的数据库设计ER图如下所示，共分为两张数据表，一张存储用户的账号密码，一张存储发送的消息。数据库使用 Mysql，在代码开发的时候安装了 JDBC 驱动所需的 jar 包并进行了对应的操作。



数据库的基本配置信息如下所示

- 数据库类型：MySQL
- 数据库服务器 localhost:3306 采用了本地服务器
- 账号：root，密码：略

2.4 Socket 通信数据格式设计

本程序在客户端和服务端之间通过建立 socket 通信来进行数据的交互，在服务端建立 serverSocket 而在客户端用 socket 与之建立 socket 通信。而不同的数据交互之间需要有一定的格式，本程序通过设计一定的前缀，来在进行 socket 通信标识不同的操作，不同的通信格式和所代表的功能如下表格所示：

前缀	代表的功能
!loginin!	登陆验证
!register!	注册用户
!msg!	聊天消息发送
!list!	获取用户列表
!history!	获取历史记录
!logout!	退出登录

表格 1 socket **通信数据格式**

三、详细设计

3.1 客户端设计

客户端的设计主要分为 GUI 界面的设计和业务逻辑的设计，其中 GUI 需要设计登陆界面和聊天主界面，业务逻辑主要是按钮的响应时间，通过对 JButton 组添加点击事件响应响应来完成。

客户端的设计主要采用了 MVC 的设计模式，实现了 GUI 界面和业务逻辑以及数据的分离，具体的设计如下所示：

3.1.1GUI 界面设计

3.1.1.1 登陆界面

登陆界面用一个类 LoginIn 来表示，登陆界面主要组件是两个输入框和一个点击按钮，以及若干用于提示和引导的的 label，主要的界面设计所需的组件和布局代码如下所示：

```
public class LoginIn {
    public JFrame loginFrame;
    public JPanel panel;
    public JLabel idLabel, pwdLabel;
    public JTextField idField;
    public JPasswordField passwordField;
    public JButton button;
    public boolean tryLogin = false;
    public String id, password;
    private void initLoginIn() {
        loginFrame = new JFrame("登陆界面");
        loginFrame.setSize(400, 250);

    loginFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE
    );
        panel = new JPanel();
        loginFrame.add(panel);
        panel.setLayout(null);
        // 两个标签
```

```
        idLabel = new JLabel("账号");
        pwdLabel = new JLabel("密码");
        idLabel.setBounds(80, 20, 80, 35);
        panel.add(idLabel);
        pwdLabel.setBounds(80, 80, 80, 35);
        panel.add(pwdLabel);
        // 两个输入框
        idField = new JTextField(20);
        idField.setBounds(165, 20, 120, 35);
        panel.add(idField);
        passwordField = new JPasswordField(20);
        passwordField.setBounds(165, 80, 120, 35);
        panel.add(passwordField);
        // 登录按钮
        button = new JButton("登录");
        button.setBounds(160, 125, 80, 30);
        panel.add(button);

        loginFrame.setVisible(true);
    }

}
```

- 登陆界面大致如下图所示：

3.1.1.2 聊天主界面

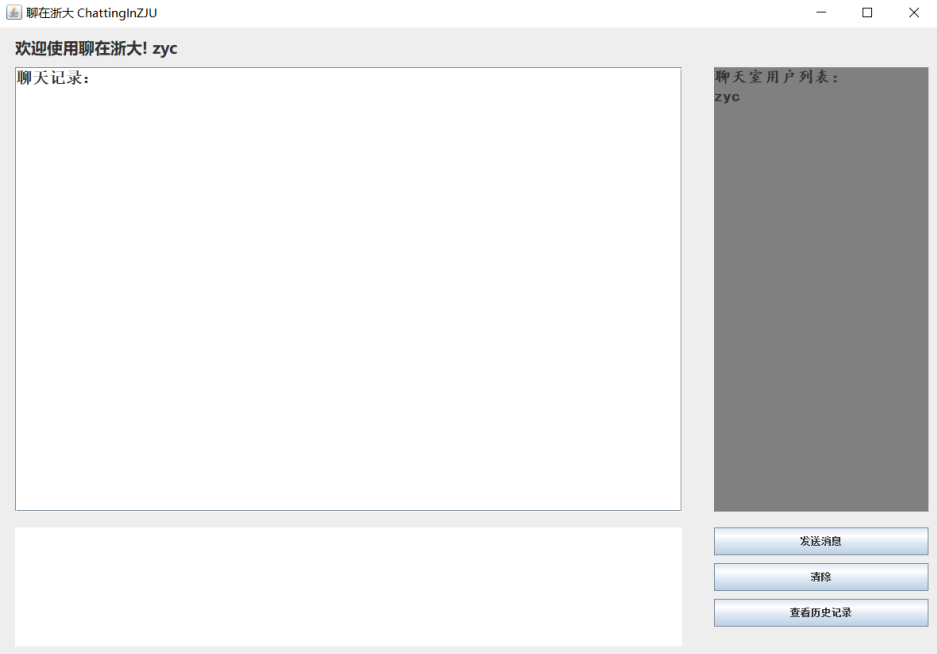
聊天界面也有一个类 MainPage 来绘制，主要使用的组件有多行输入的文本框、以及聊天记录和当前在线用户的显示框以及若干可以用于实现业务逻辑的按钮，主要的代码如下所示：

这些界面的设计虽然繁琐，但是代码逻辑上是较为简单的。主要的聊天主界面的相关代码设计如下所示：

```
public class MainPage {
    public JFrame frame;
    public JPanel panel;
    public JTextArea show, list;
    public JEditorPane input;
    public JButton send, clear, history;
    public JLabel label;
}
```

具体的页面布局的设置可以查看源代码中的 MainPage.java 中的设计

- 聊天主界面的大致设计



3.1.2 界面交互逻辑设计：Client 类

我的这次作业中主要设计了这样几种**界面内的交互逻辑**

- 点击按钮登录
- 点击按钮清除输入框
- 点击按钮发送消息
- 点击按钮查看历史纪录

其实这几种操作都是通过同一种方式实现的，那就是对鼠标点击设置事件监听，每当鼠标点击对应的按钮时就触发对应的操作，通过修改组件的显示内容和状态来达到与用户进行交互的目的

而页面外的交互逻辑主要是通过客户端和服务端建立 socket 通信进行消息的收发来进

行，具体的设计和实现将在下面逐一介绍。

3.1.2.1 响应事件监听

首先要给对应的组件（本次作业中主要是按钮）添加事件监听，这里我使用类的组合，通过一个 Client 类来控制前面设计的登陆界面和主聊天界面的显示和消失，并在 client 类中添加对上述两个类的事件监听，这里主要是对按钮设置鼠标点击的事件监听，具体实现代码如下所示：

```
private void setLoginListener() {
    this.login.button.addMouseListener(new MouseAdapter() {
        @Override
        public void mouseClicked(MouseEvent e) {
            super.mouseClicked(e);
            id = login.idField.getText();
            password =
String.valueOf(login.passwordField.getPassword());
            checkLogin();
        }
    });
}
```

上面展示的主要是登陆界面的事件监听，而主聊天界面的时间监听设计也类似，只不过因为按钮有三个因此就设置了三个不同的事件监听，具体的代码比较长这里限于篇幅就不复制粘贴了，老师可以在源代码中查看。

3.1.2.2 Socket 建立与通信

我在客户端完成其初始化之后令其建立 socket 连接，通过 socket 和服务器得 serverSocket 建立连接，并创建 I/O 流，之后就可以进行客户端和服务端的通信，其中客户端请求建立 socket 连接的代码如下：

```
public void connectServer() {
    try {
```

```
            socket = new Socket(server, port);
            receive = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            send = new PrintWriter(socket.getOutputStream(), true);
            //System.out.println(send);
            //System.out.println(receive);

        } catch (UnknownHostException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

而客户端也有一些用于通过 socket 的 I/O 流进行数据接收和发送的方法，我将其抽象成了两个单独的 api，其中发送 api 直接通过 PrintWriter 类的对象进行发送，这一 api 可以被各类事件监听函数调用：

```
public void sendToServer(String info) {
    System.out.println("要发送的信息：" + info);
    try {
        send.println(info);
        System.out.println("发送成功");
    } catch (Exception exception) {
        exception.printStackTrace();
    }
}
```

而接收数据我使用了一个单独的**内部类定义一个监听线程**，用**继承 Thread 的方法实现了多线程**，子线程覆写了 run 方法，对服务器发送过来的数据进行持续监听并读取其中的内容，具体的代码如下：

```
public void startListen() {
    receiveThread = new InfoListener();
    receiveThread.start();
}

/**
 * 定义一个socket 子线程用于接收信息
 */
class InfoListener extends Thread {
    @Override
    public void run() {
```

```
            System.out.println("客户端监听线程启动！");
            while (isRunning) {
                if (socket.isClosed()) {
                    isRunning = false;
                    break;
                }
                receiveFromServer();
                try {
                    sleep(100);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                } catch (NullPointerException e) {
                    e.printStackTrace();
                }
            }
        }
    }
```

而这里的 receiveFromServer 会接收 BufferReader 中的数据，并根据上面定义的 socket 通信格式来对收到的内容进行解析（共有五种格式），具体代码的总体架构如下，因为原本的代码比较长，这里删减了一些不重要的格式处理代码，仅保留大致的框架，详细代码可以在源代码文件中的 Client.java 中查看：

```
public void receiveFromServer() {
    String line;
    try {
        if (!socket.isClosed()) {
            line = receive.readLine();
        } else {
            System.out.println("Socket 已经关闭了！");
            return ;
        }
     if (!line.isEmpty()) {
        System.out.println("客户端收到了服务器的消息：" + line);
            // 处理登录请求
            if (line.startsWith("!login!")) {
                // login 的1，2 两种状态分别表示登陆成功，账号或者密码错误
                String res = line.substring(7);
                if (res.equals("1")) {
                    isLogin = 1;
                    System.out.println("登陆成功！");
                } else if (res.equals("2")) {
                    isLogin = -1;
```

```
                }
            } else if (line.startsWith("!msg!")) {

            } else if (line.startsWith("!list!")) {

            } else if (line.startsWith("!history!")) {

            }
        }
    } catch (IOException | NullPointerException e) {
        System.out.println("捕捉到 IO 异常");
    }
}
```

而客户端在监听到特定事件的时候又会向服务端发送对应的消息，比如点击登录按钮、点击发送按钮、点击获取历史记录的按钮等操作发生的时候，客户端会生成对应格式的字符串并通过调用 sendToServer 这个 API 来进行消息的发送，比如聊天消息的发送就用如下方式进行：

```
public void sendAMessage(String msg) {
    String info = this.id + " " +
LocalDateTime.now().toString();
    // 将消息发送到服务器
    this.sendToServer("!msg!" + info + "@" + msg);
}
```

3.1.2.3 Socket 的关闭设计

客户端和服务端的 socket 连接需要在客户端关闭的时候自动切断，因此需要先对整个 MainPage 进行窗体事件监听，当窗口退出的时候自动执行 socket 连接的关闭，并发送消息给服务端让其关闭 socket 连接，具体的代码如下：

```
this.mainPage.frame.addWindowListener(new WindowListener() {
    @Override
    public void windowClosing(WindowEvent e) {
        isRunning = false;
        System.out.println("退出登录！");
        sendToServer("!logout!");
        try {
```

```
                receive.close();
                send.close();
                socket.close();
                System.out.println("closed!");
            } catch (IOException ioException) {
                ioException.printStackTrace();
            }
            mainPage.frame.dispose();
        }
});
```

3.2 服务端设计

服务端的主要功能包括维护当前聊天室中的用户信息，**使用 serverSocket 为每个请求建立 socket 连接的客户端都创建一个 socket 连接，并进行消息的转发和处理以及数据库的增删查改操作。**具体的各个功能模块的设计如下：

3.2.1 数据库操作的设计

本次作业我在服务端实现了基于 JDBC 的数据库相关操作，其中数据表的设计在上面第二章已经介绍了，而数据库的相关操作主要有：

- 建立 JDBC 数据库驱动，这一步在服务器的构造函数中就已经实现，通过

```
public Server() throws SQLException {
    // 注册数据库 JDBC 驱动
    DriverManager.registerDriver(new Driver());
}
```

- 涉及到用户的数据库操作主要有：查询用户的账号密码判断账号密码是否正确，以确定用户是否可以登录、用户的注册这些操作都需要现在数据库驱动上建立 connect 并构造 statement 来执行 SQL 语句，而 SQL 语句则由服务端根据查询条件生成，具体的代码实现如下：
-
```
public boolean checkLogin(String id, String pwd) throws
SQLException {
    boolean res = true;
    //DriverManager.registerDriver(new Driver());
    Connection connection = DriverManager.getConnection(url,
user, password);
```

```
        Statement statement = connection.createStatement();
        String select = "select * from userinfo where
user_id=\'" + id + "\' and user_password=\'" + pwd + "\';";
        //System.out.println(select);
        ResultSet resultSet = statement.executeQuery(select);
        if (!resultSet.next()) {
            System.out.println("空的! ");
            res = false;
        }
        resultSet.close();
        statement.close();
        connection.close();
        return res;
    }

    public void registerUser(String id, String pwd) throws
SQLException {
        Connection connection = DriverManager.getConnection(url,
user, password);
        Statement statement = connection.createStatement();
        String insert = "insert into userinfo values(\'" + id +
"\',\'" + pwd + "\');";
        System.out.println(insert);
        int result = statement.executeUpdate(insert);
        statement.close();
        connection.close();
    }
```

- 而涉及到聊天记录的主要数据库操作有向数据库中添加新的聊天记录和向数据库中查询所有的历史记录，具体的实现方法和上面类似，这里我截取两个方法中的主要部分，删节了部分无关源代码：

```
    public void storeMessageToDB(String line, String user)
throws SQLException {
        String msg = line.substring(5);
        int pos1 = msg.indexOf(' ');
        int pos2 = msg.indexOf('@');
        String info = msg.substring(pos1 + 1, pos2);
        String text = msg.substring(pos2 + 1);
        String date = info.substring(0, 10);
        String time = info.substring(11, 19);
        // 连接数据库进行存储
        Connection connection = DriverManager.getConnection(url,
```

```
Server.user, password);
    Statement statement = connection.createStatement();
    String insert = "insert into msg values(\'" + user +
"\',\'" + date + "\',\'"
            + time + "\',\'" + text + "\');";
    System.out.println(insert);
    statement.executeUpdate(insert);
    statement.close();
    connection.close();
}
```

- 在数据库中可以查询到聊天的历史记录，并按照一定的格式返回，之后由子线程发送给对应的客户端

```
public Vector<String> selectHistoryMessageFromDB() throws
SQLException {
    Connection connection = DriverManager.getConnection(url,
user, password);
    Statement statement = connection.createStatement();
    String select = "select * from msg;";
    ResultSet resultSet = statement.executeQuery(select);
    Vector<String> history = new Vector<String>();
    while (resultSet.next()) {
        String line1 = resultSet.getString("user_id") + " "
                + resultSet.getString("send_date") + " "
                + resultSet.getString("send_time");
        history.add(line1);
        String line2 = resultSet.getString("msg");
        history.add(line2);
    }
    return history;
}
```

- 数据库的相关操作统一在服务端的主线程中进行，而 socket 通信中的子线程可以调用主线程中的相关方法获取数据
- 我将数据库的操作和线程通信的操作进行了解耦，为数据库操作构建了一组 RESTfule 的 API 供子线程调用，这样不仅提高了开发效率和 debug 的效率，也增加了代码的可复用性。

3.2.2 服务端 Socket 通信设计

首先服务器需要先进行 serverSocket 的初始化，具体的代码如下：

```
private void serverStart() {
    try {
        InetAddress ip = InetAddress.getByName(addr);
        serverSocket = new ServerSocket(port, 0, ip);
        clients = new Vector<ClientThread>();
        System.out.println("服务器正常开启……等待客户端连接中");
    } catch (UnknownHostException e) {
        e.printStackTrace();
        System.out.println("host 异常");
    } catch (IOException e) {
        e.printStackTrace();
        System.out.println("无法部署 socket 服务!");
    }
}
```

服务端初始化之后就建立一个监听方法接收来自客户端的 connect 请求，每次为一个客户端建立单独的子线程并进行 socket 通信，这一部分的具体代码如下：

```
public void serverListen() {
    try {
        while (true) {
            Thread.sleep(200);
            Socket socket = serverSocket.accept();
            ClientThread clientThread = new ClientThread(socket,
this);
            Thread thread = new Thread(clientThread);
            thread.start();
        }
    } catch (InterruptedException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

这里的 ClientThread 是我为每个客户端连接设计的子线程类，具体的将在下一节说明。要注意的是，部分消息收发的功能需要服务端的主线程中实现，比如数据库的查询和向每一个客户端发送更新的聊天记录，这些功能是子线程调用主线程来进行的，比如向所有的客户端发送信息这一方法的定义如下：

```
/**
 * 向所有的客户端发送广播消息
 *
 * @param msg 要发送的消息
 */
public void sendToAllClient(String msg) {
    Iterator<ClientThread> it = clients.iterator();
    while (it.hasNext()) {
        ClientThread temp = it.next();
        temp.sendInfo(msg);
    }
}
```

这里的 clients 就是一个 ClientThread 组成的 Vector，我们在服务端的主线程中维护这一信息，并在需要的时候使用。

3.2.3 并发子线程设计

上面 3.2.1 中的 ClientThread 正是为每个客户端创立的服务端子线程，**支持多个客户端的并发操作，通过实现 runnable 接口来实现多线程并发，该类的定义如下：**

```
public class ClientThread implements Runnable {
    private Socket socket;
    private Server server;
    private boolean isRunning;
    private String name;

    public BufferedReader input = null;
    public PrintWriter output = null;

    public ClientThread(Socket socket, Server server);
```

```
    @Override
    public void run();
    public void sendInfo(String things);
    public void sendHistory(Vector<String> history);
    public void receiveInfo();
}
```

该子线程类需要两个参数进行初始化，分别是建立好的 socket 以及主线程 server 的引用，其中第二个参数主要用于绑定主线程，并调用主线程中的方法。

子线程类包含了收发消息的功能，其中该类覆写了 run 方法并进行客户端发送消息的监听：

```
@Override
public void run() {
    try {
        input = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
        output = new PrintWriter(socket.getOutputStream(),
true);
        System.out.println("服务端建立新的子线程：" +
socket.getInetAddress());
        this.receiveInfo();
        System.out.println("子线程运行结束.......服务器依然在运
行......");
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

而 reeiveInfo 方法用于在建立好的 I/O 流中监听客户端发送过来的消息，按照确定的消息格式进行解析，并对不同的消息进行不同的操作，这个方法的基本代码架构如下，具体字符串处理的细节因为比较长且没必要因此予以删减：

```
public void receiveInfo() {
    String line;
    try {
        while (isRunning) {
            Thread.sleep(10);
            line = input.readLine();
            if (!line.isEmpty()) {
```

```
                String msg = "服务端收到消息：" +
this.socket.getInetAddress() + " " + line;
                System.out.println(msg);
                if (line.startsWith("!register!")) {
                        // do something
                } else if (line.startsWith("!login!")) {
                    //登录处理部分
                   if (this.server.checkLogin(name, pass)) {
                        this.sendInfo("!login!1");
                    } else {
                        this.sendInfo("!login!2");
                    }
                } else if (line.startsWith("!msg!")) {
                    //收发聊天消息的处理部分,广播给所有的客户端
                } else if (line.startsWith("!history!")) {

this.sendHistory(this.server.selectHistoryMessageFromDB());
                } else if (line.startsWith("!logout!")) {
                        // do logout
                }
            }

        }
    }
catch (InterruptedException | IOException | SQLException e) {
        e.printStackTrace();
    }
}
```

而发送消息分为向当前客户端发送和向所有客户端发送两种，像当前客户端发送至需要在 output 中调用 write 就可以，而向所有的客户端发送需要调用主线程中的 sendToAllClient 方法，遍历当前服务端正在运行的所有客户端分别调用其 send 方法进行消息的发送。

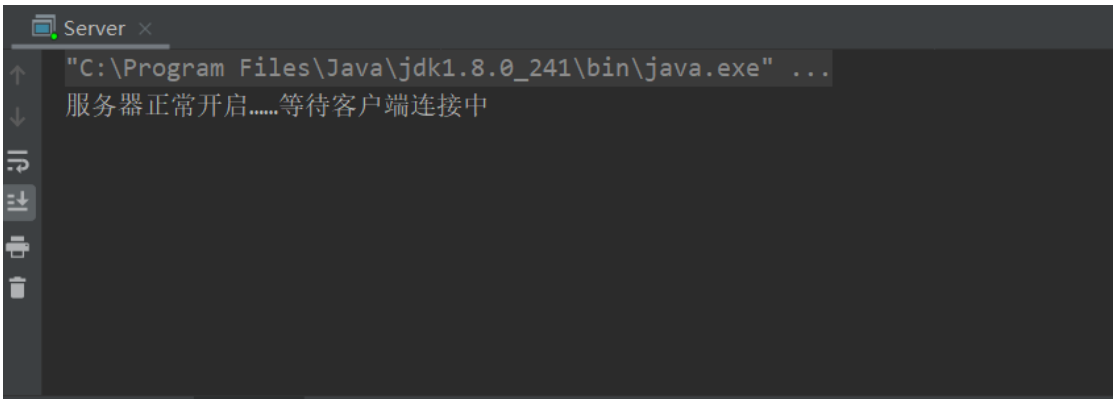
4. 测试与运行

4.1 程序测试

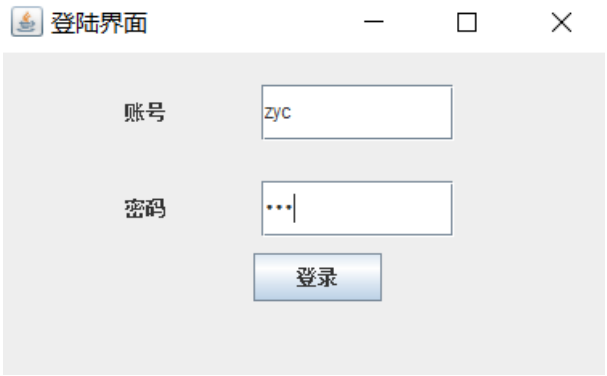
在编写完本次作业的代码之后，我对程序的基本功能进行了简单的测试发现基本符合作业要求和我的预期目标，实现了用户的登录和在线通信，但是程序仍然有非常多的代码可以优化，也有非常多的新功能可以添加，或许以后有时间我会再完善这一程序。此外，我将客户端程序打包成了 jar 包来运行，可以支持多个用户的同时使用。

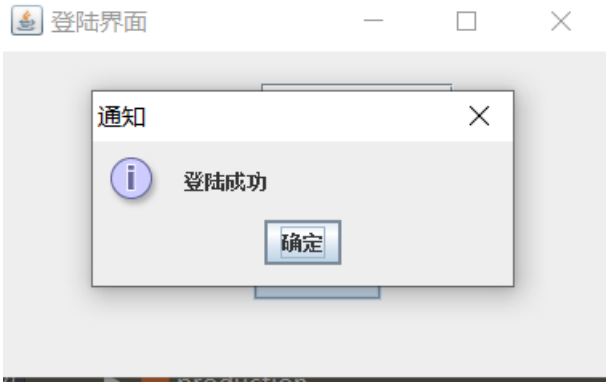
4.2 程序运行

- 服务器开启时的状态

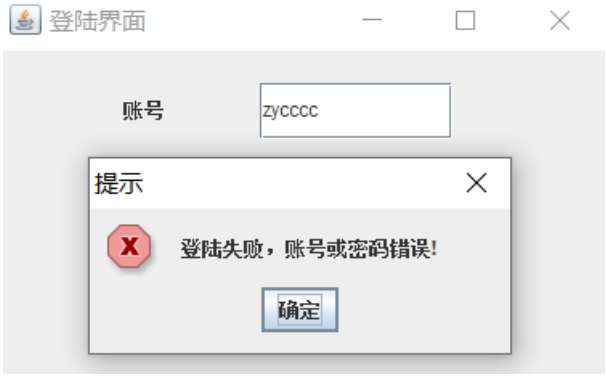


- 用户输入账号密码登录

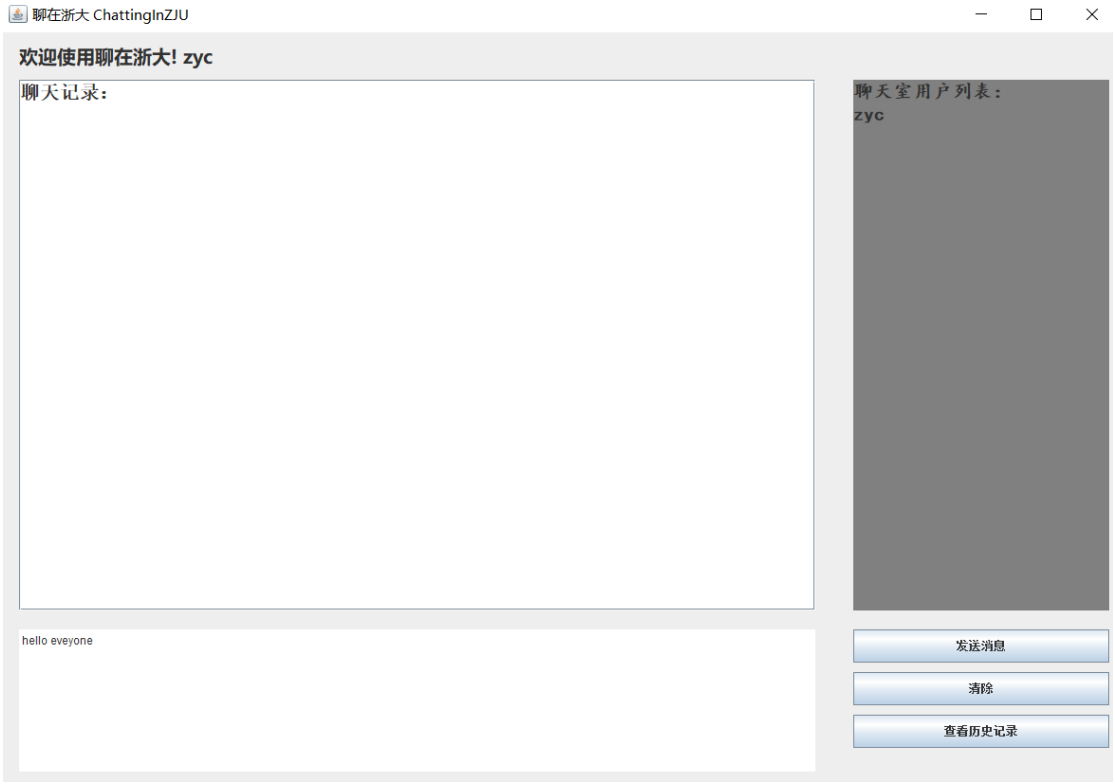


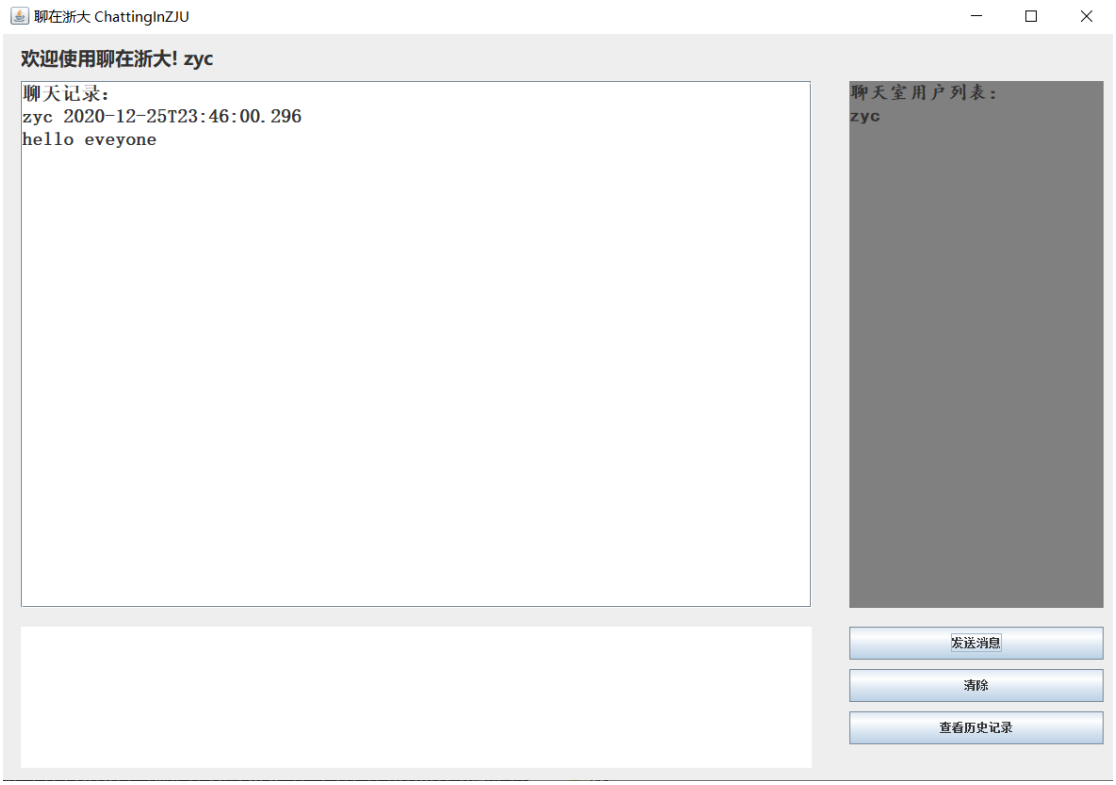


- 登陆失败（这个账号并不存在）

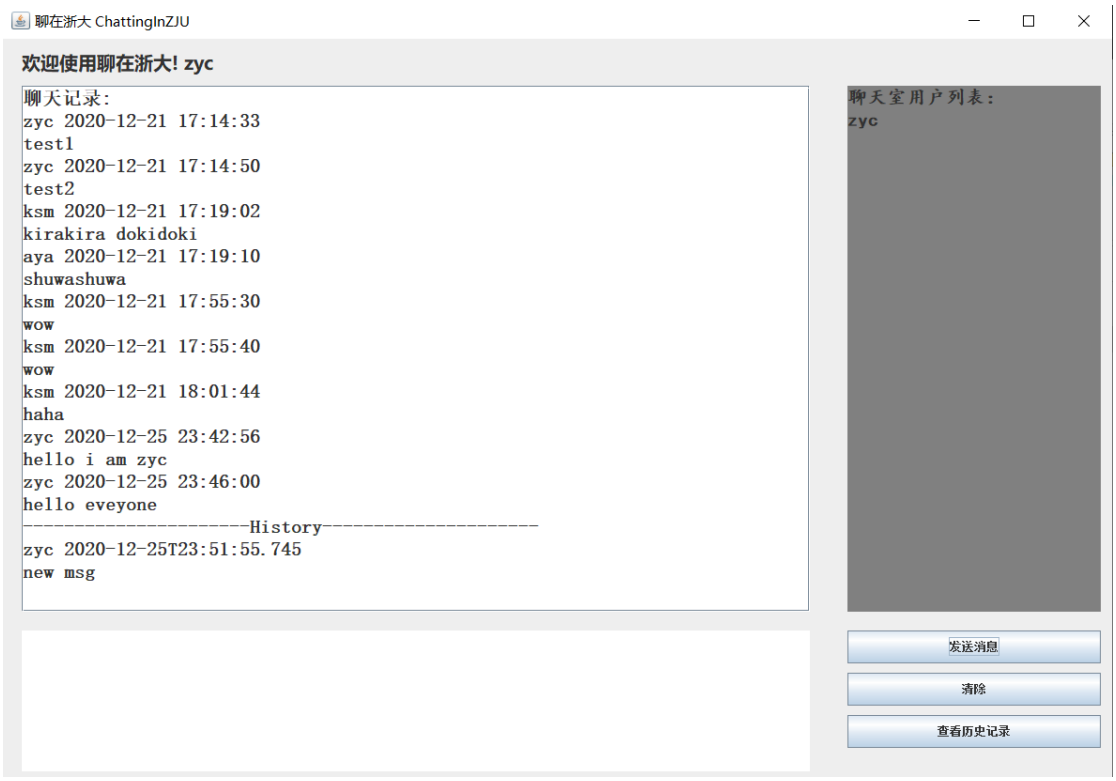


- 进入聊天界面输入并点击按钮发送消息

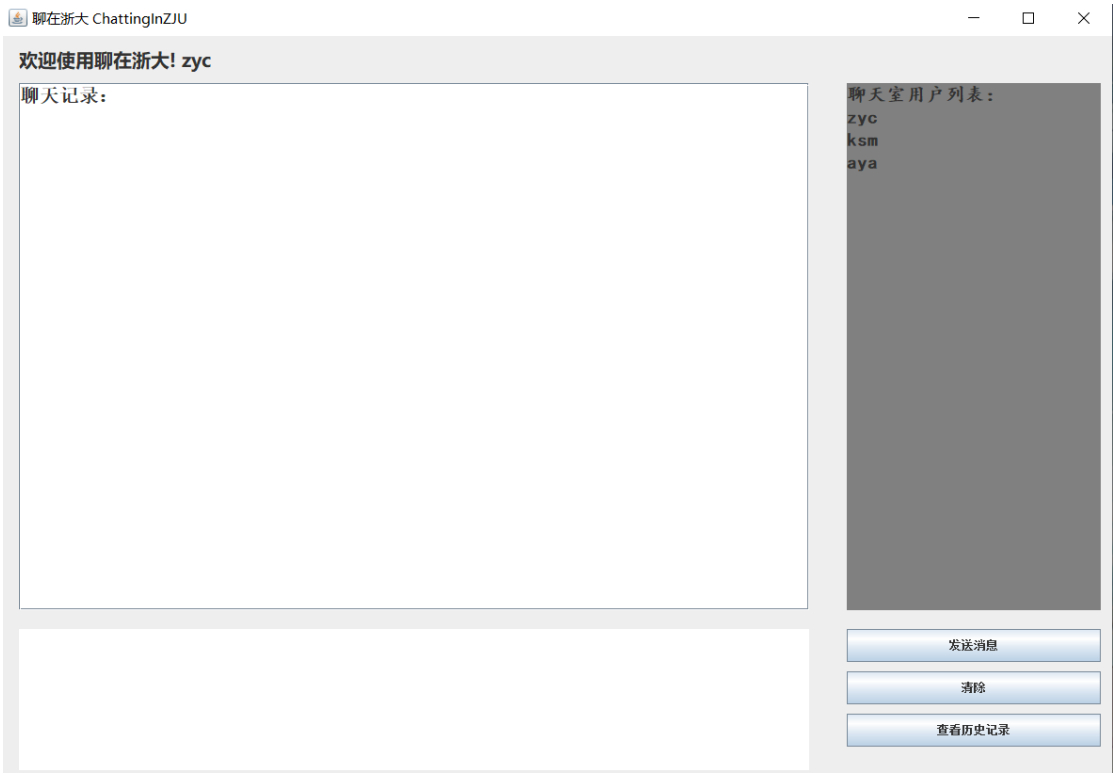




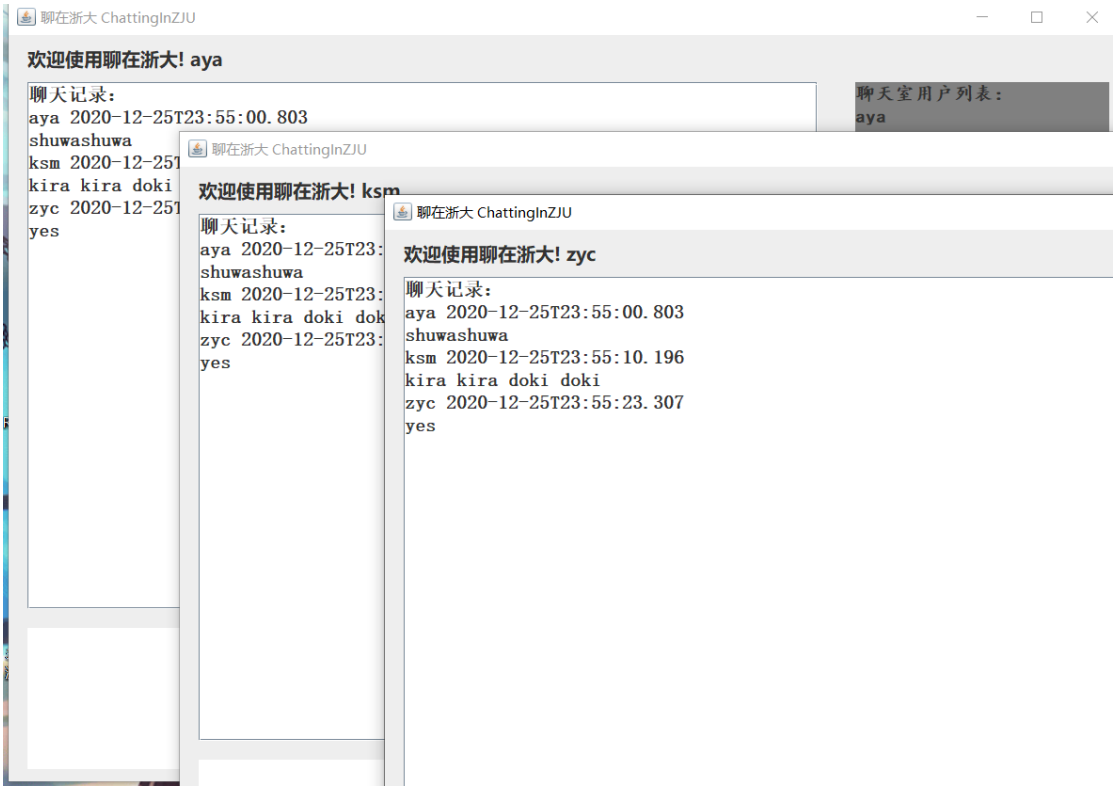
- 点击按钮并查看历史记录（注：这一操作需要等待几秒）



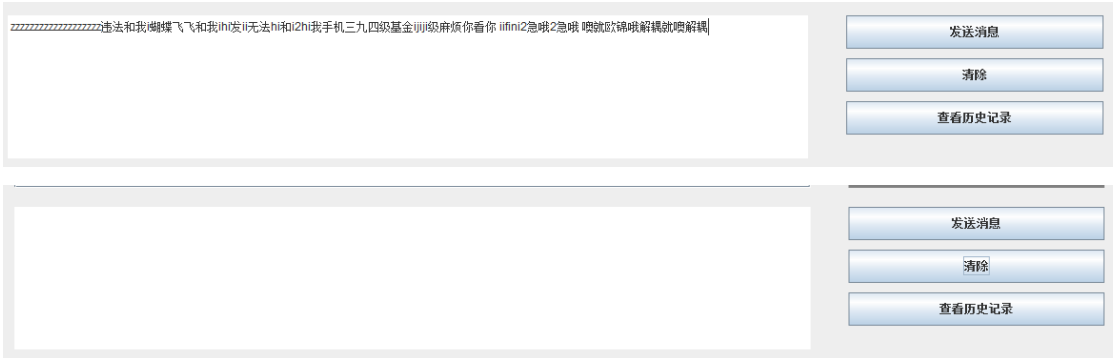
- 多个用户同时使用聊天室进行聊天，注意到用户列表中有多个用户



多人之间发送消息进行聊天对话：



- 清空输入框，点击清除



- 基本的功能展示到此结束，观察服务器端的输出：

```
服务器正常开启.....等待客户端连接中
服务端建立新的子线程：/127.0.0.1
服务端收到消息：/127.0.0.1 !login!zyc 123
输入的用户名和密码是：
zyc
123
服务端收到消息：/127.0.0.1 !register!zyc
当前连接用户：zyc
服务端建立新的子线程：/127.0.0.1
服务端收到消息：/127.0.0.1 !login!ksm 123
输入的用户名和密码是：
ksm
123
服务端收到消息：/127.0.0.1 !register!ksm
当前连接用户：ksm
服务端建立新的子线程：/127.0.0.1
服务端收到消息：/127.0.0.1 !login!aya 123
输入的用户名和密码是：
aya
123
服务端收到消息：/127.0.0.1 !register!aya
当前连接用户：aya
服务端收到消息：/127.0.0.1 !msg!aya 2020-12-25T23:55:00.803@shuwashuwa
insert into msg values('aya','2020-12-25','23:55:00','shuwashuwa');
服务端收到消息：/127.0.0.1 !msg!ksm 2020-12-25T23:55:10.196@kira kira doki doki
insert into msg values('ksm','2020-12-25','23:55:10','kira kira doki doki');
服务端收到消息：/127.0.0.1 !msg!zyc 2020-12-25T23:55:23.307@yes
insert into msg values('zyc','2020-12-25','23:55:23','yes ');
```

- 整个操作流程中客户端和服务端都没有报错，说明 socket 和 I/O 流都已经正常关闭。

5. 总结

本次作业任务量较大并且综合性较高，结合了 Java 的 Swing GUI 组件以及网络编程、数据库编程和并发编程，充分考察了 Java 应用技术课程后半个学期学习的内容，我花了 2-3 天的时间完成了本次作业的代码和文档。

我在完成作业的过程中不断学习和实践上课讲到过的理论知识，像 socket 编程、数据库编程和并发编程这些内容只有自己真的动手写过才能掌握其用法，我在设计不同种类的 socket 的时候也设计了一些简单的通信格式来区分不同的通信内容，而在界面设计的过程中也实践了 MVC 设计模式。

我在完成本次作业的过程中也遇到过很多问题，比如 GUI 设计中因为没有关闭默认的页面布局而导致界面设计一直出现组件重叠的问题，比如 socket 建立通信之后 I/O 流可能存在的临界区问题，比如数据库 JDBC 外部 jar 包导入的时候出现了一堆错误等等，但是这些问题最后都被我一一解决了。

总而言之，虽然本次作业有点难度，但做完之后我确实收获很大。但是这个程序依然有很多可以改进的地方，比如代码架构的优化和细节的修改，这个程序也有很多新的功能可以添加。

参考文献

[1] 耿祥义. Java 大学实用教程[M]. 北京：清华大学出版社，2009.

[2] 耿祥义. Java 课程设计[M]. 北京：清华大学出版社，2008.

[3] 王鹏. Java Swing 图形界面开发与案例详解[M]. 北京：清华大学出版社，2008.

[4] 丁振凡. Java 语言实验教程[M]. 北京：北京邮电大学出版社，2005.

[5] 郑莉. Java 语言程序设计[M]. 北京：清华大学出版社，2006.