网络应用开发实训项目报告

题目：聊天程序

姓名：

学号 ：

班级：

目录

[第一章 引言 4](#_Toc519456731)

[第二章 概要设计 4](#_Toc519456732)

[1 使用平台及技术 4](#_Toc519456733)

[⑴ Java的优点 4](#_Toc519456734)

[⑵ MySQL介绍 5](#_Toc519456735)

[2环境的搭建过程 5](#_Toc519456736)

[⑴ Eclipse的安装 5](#_Toc519456737)

[⑵ MySQL的安装 5](#_Toc519456738)

[⑶ JDBC驱动包的下载与引入 5](#_Toc519456739)

[3 功能和设计方法 5](#_Toc519456740)

[⑴ 注册功能 6](#_Toc519456741)

[⑵ 登录功能 6](#_Toc519456742)

[⑶ 忘记密码 6](#_Toc519456743)

[⑷ 单独聊天（私聊） 6](#_Toc519456744)

[⑸ 多人聊天（群聊） 6](#_Toc519456745)

[⑹ 添加好友 7](#_Toc519456746)

[⑺ 修改密码 7](#_Toc519456747)

[第三章 详细设计 7](#_Toc519456748)

[1 流程图介绍 7](#_Toc519456749)

[⑴ 系统总体结构设计 7](#_Toc519456750)

[⑵ 系统主体处理流程 8](#_Toc519456751)

[⑶ 系统流程逻辑 8](#_Toc519456752)

[2 主要类图说明 9](#_Toc519456753)

[⑴ 客户端中主要类图 9](#_Toc519456754)

[⑵ 服务端中主要类图 9](#_Toc519456755)

[3 数据库表展示 10](#_Toc519456756)

[⑴ 数据库构成截图 10](#_Toc519456757)

[⑵ 用户表（tb\_user）的结构及存放的数据 11](#_Toc519456758)

[⑶ 某一用户好友表结果及数据 11](#_Toc519456759)

[4 主要算法、类的描述 11](#_Toc519456760)

[⑴ 客户端中主要类、算法的描述 11](#_Toc519456761)

[⑵ 服务器端主要类、算法的描述 12](#_Toc519456762)

[5 主要的代码段 12](#_Toc519456763)

[⑴ 客户端的主要代码段（略去包名） 12](#_Toc519456764)

[⑵ 服务器端主要代码段（略去包名） 22](#_Toc519456765)

[第四章 测试和演示 37](#_Toc519456766)

[1 测试内容及预测结果 37](#_Toc519456767)

[2 测试结果与截图 37](#_Toc519456768)

[⑴ 测试点一 37](#_Toc519456769)

[⑵ 测试点二 39](#_Toc519456770)

[⑶ 测试点三 39](#_Toc519456771)

[⑷ 测试点四、七一起测试 40](#_Toc519456772)

[第五章 总结 41](#_Toc519456773)

# 第一章 引言

本系统是基于java开发的聊天室。有用户注册、用户登陆、修改密码、忘记密码、添加好友、用户聊天、群聊功能。如果服务器还没有启动，则客户端是不可以登陆、注册、忘记密码，如果在运行过程中，服务器断开则系统会有提示，聊天对象如果下线发送消息后会有弹窗提示，添加好友后有是否添加好友成功提示。

# 第二章 概要设计

## 1 使用平台及技术

此项目运行在Windows 10上，使用Eclipse作为IDE，用MySQL作为数据库。以Java为主要设计语言。

⑴ Java的优点

①简单性

Java实现了自动的功能收集，简化了内存管理的工作，这使程序设计更简便，同时减少了出错的可能。

②面向对象

Java提供了简单的类机制和动态的架构模型。对象中封装了它的状态变量和方法。很好地实现了模块化和信息隐藏；而类则提供了一类对象的原型，通过继承和重载机制，子类可以使用或重新定义父类或超类所提供的方法，从而既实现了代码的复用，又提供了一种动态的解决方案。

Java同样支持继承特性，java的类可以从其他类中继承行为，但java只支持类的单重继承，即每个类只能从一个类中继承。

Java支持界面，界面允许程序员定义方法但不立即实现，一个类可以实现多个界面，利用界面可以得到多重继承的许多优点而又没有多重继承的问题。

③多线程

多线程使应用程序可以同时进行不同的操作，处理不同的事件。在多线程机制中，不同的线程处理不同的任务，他们之间互不干涉，不会由于一处等待影响其他部分，这样容易实现网络上的实时交互操作。多线程保证了较高的执行效率。

④安全性

用于网络、分布环境下的java必须要防止病毒的入侵，java不支持指针，一切对内存的访问都必须通过对象的实例变量来实现，这样就防止了程序员使用欺骗手段访问对象的私有成员，同时也避免了指针操作中容易产生的错误。

⑵ MySQL介绍

MySQL是一个关系型数据库管理系统，由瑞典MySQL AB 公司开发，目前属于 Oracle 旗下产品。MySQL 是最流行的关系型数据库管理系统之一，在 WEB 应用方面，MySQL是最好的 RDBMS (Relational Database Management System，关系数据库管理系统) 应用软件。

MySQL是一种关系数据库管理系统，关系数据库将数据保存在不同的表中，而不是将所有数据放在一个大仓库内，这样就增加了速度并提高了灵活性。

MySQL所使用的 SQL 语言是用于访问数据库的最常用标准化语言。MySQL 软件采用了双授权政策，分为社区版和商业版，由于其体积小、速度快、总体拥有成本低，尤其是开放源码这一特点，一般中小型网站的开发都选择 MySQL 作为网站数据库。

## 2环境的搭建过程

⑴ Eclipse的安装

在<https://www.eclipse.org/downloads/> 中选择64bit的选择下载安装到本地D盘。

⑵ MySQL的安装

在<https://www.mysql.com/downloads/>中选择Microsoft Windows版本下载到本地，然后安装（按默认安装即可），设置密码MySQL Root Password并记住。

⑶ JDBC驱动包的下载与引入

由于Eclipse不自带JDBC驱动，然而JDBC驱动是Java连接数据库必要的包，因此按以下步骤下载引入包。

①JDBC的下载

在浏览器输入<https://dev.mysql.com/downloads/connector/j/>，选择zip文件下载到本地，解压到合适的位置。

②JDBC包的导入

在项目处右键 ->Build Path ->Add External Archives ->选择下载好的jdbc驱动包.jar文件 ->打开。

## 3 功能和设计方法

总体而言，在客户端：当用户登录后，生成唯一的socket， 存放在Client实体类中，在整个客户端就一个Client类和一个socket。有一个窗口控制器——ChatUIList，用来记录用户和好友聊天框是否打开，当收到消息后，首先在ChatUIList中查询是否有好友的窗口，如果没有则新建聊天框弹出并显示消息，如果存在与好友的窗口则将消息追加到原聊天框并重新着重显示该窗口。在客户端还拥有一个“命令控制中心”——ChatTread类，在ChatTread类中判断并处理来自服务器中的命令（消息），如果是“message”那么客户端收到是来自好友的消息，如果是“requeste\_add\_friend”则是好友申请命令，类似有“WorldChat”, “accept\_add\_friend”，“refuse\_to\_add”，“changepwd”等命令。在服务端：有多个socket，用SockList管理连接成功的用户名及其socket。同样在服务端也有一个“命令控制中心”——ServerTread类，它负责处理来自客户端的命令（消息），判断命令的类型，并正确处理他们，给出处理结果和判断是否处理成功，将处理后的命令转发给正确的用户。

⑴ 注册功能

设计一个注册UI（RegisterUI）类，在打开程序后，模仿QQ在左下方有一个注册按钮，点击注册按钮后弹出注册页面，用户填完必填信息后由客户端将命令发送给服务端（如果服务器在线），服务器收到“register”命令后，连接数据库判断，如果注册成功则返回注册成功消息并弹窗提示，如果失败则弹窗提示注册失败。

⑵ 登录功能

打开客户端后，类似QQ有登录按钮，当用户填完用户账号和用户密码并提交后，客户端将登录请求发送给给服务端判断（如果服务端在线），如果密码正确则用户登录成功，显示朋友列表（FriendsUI），否则提示密码错误或账号不存在。

⑶ 忘记密码

在客户端右下侧有忘记密码按钮，用户点击按钮后弹出忘记密码页面（ForgetUI），用户填写用户账号后客户端将消息发往服务器，服务器在数据库中检测该账号是否存在，如果存在则显示提示问题，如果不存在则提示账号不存在。用户填写完后续相关信息后，点击“重置密码”按钮后，如果找回密码答案正确则向服务器发送修改密码请求，如果失败则弹窗提示密码错误。最后服务器将处理结果（修改密码是否成功）返还给客户端。

⑷ 单独聊天（私聊）

用户登录成功后，双击好友后，首先判断用户与该好友是否有聊天框存在，如果不存在则创建新的聊天框（ChatUI）并在ChatUIList中登记，如果存在则将改聊天框突出显示。用户可以再聊天页面（ChatUI）发送消息，如果好友不在线，服务器会返回好友不在线提示，客户端弹窗提示，如果好友在线收到消息则无提示（类似Linux，没有消息就是好消息）。好友收到消息时，在ChatUIList中查询是否有与该好友的聊天窗口，如果没有则新建窗口显示并在ChatUIList中注册，如果存在则直接将消息追加到聊天窗口上并突出显示。

⑸ 多人聊天（群聊）

这里实现的多人聊天式世界喊话，即在线用户都能收到世界喊话的消息，没有好友限制，实现与单独聊天类似。不同的是，服务器收到“世界喊话”命令后，在SocketList中查询当前在线用户，并将世界喊话消息发送给这些用户。

⑹ 添加好友

在好友列表页面左下角有添加好友按钮，点击该按钮后弹出添加好友框（AddFriendUI），在添加好友框中重复输入两次欲添加的好友name便可向服务器发送好友请求。当用户收到好友请求后，同意或拒绝都像添加方反馈，添加成功后重新登录便可刷新好友列表。

⑺ 修改密码

在好友列表右下方有修改密码按钮，点击该按钮后弹出修改密码框（ChangePwdUI），在这里只需要重复输入两次新密码即可修改密码，是否修改成功服务器都会做出应答，客户端有弹窗提示。在服务端对数据库进行操作，由于可能数据库会出错，如果数据库未成功修改密码，那么要提醒客户

# 第三章 详细设计

## 1 流程图介绍

⑴ 系统总体结构设计

基于java实现聊天室

客户端

服务器端

聊天室

接收信息

清屏

聊天记录

发送信息

⑵ 系统主体处理流程

发送请求

服务端

客户端

返回应答

⑶ 系统流程逻辑

开始

运行服务端代码

运行客户端代码

登录成功

发送信息

接收信息

清屏

显示聊天内容

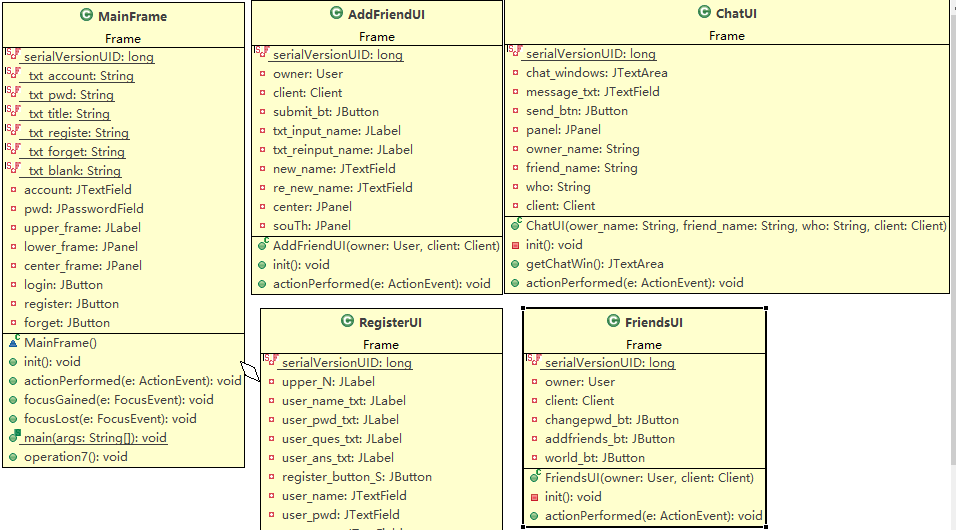
是否退出

结束

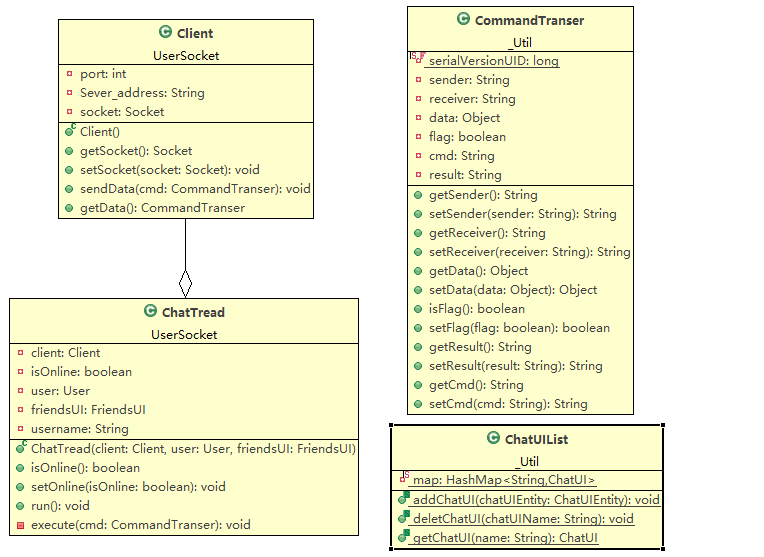
## 2 主要类图说明

⑴ 客户端中主要类图

①UI类图

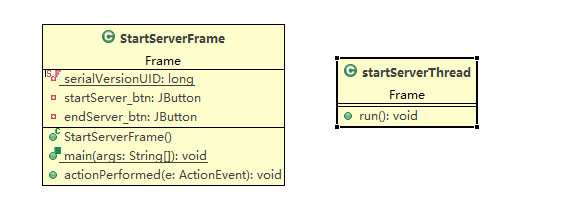


②消息处理、控制类图

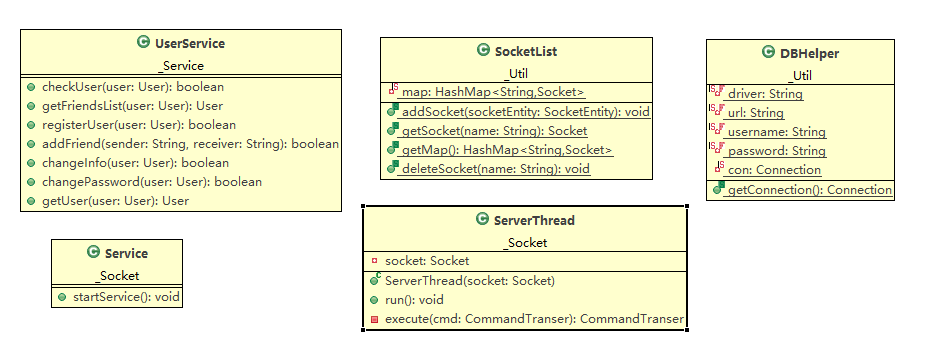


⑵ 服务端中主要类图

①UI类图



②消息处理、控制类、数据库相关图

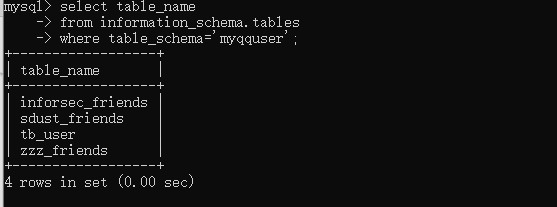


## 3 数据库表展示

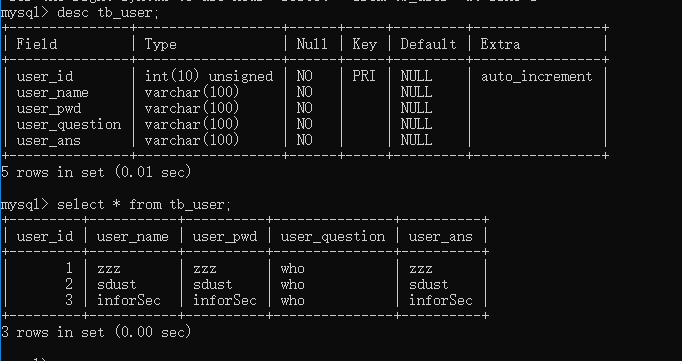
数据库名为myqquser， 此数据库中若干个表，一个用户表（tb\_user），用来存储用户的信息，如用户名，用户密码，用户问题，用户答案；每一个用户有一个好友表，好友表里存着用户的好友名。

⑴ 数据库构成截图

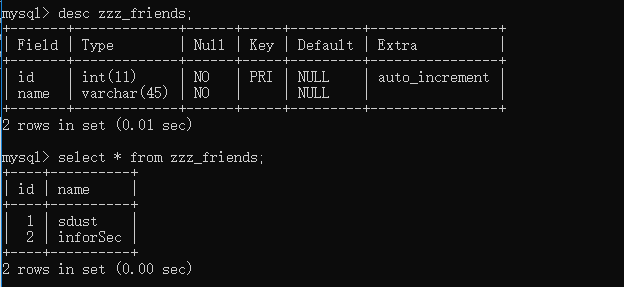
此时数据库中有三个用户分别为inforSec、 zzz、sdust，tb\_user中存放着用户的信息，inforSec\_friends中存放着inforSec的好友信息，zzz\_friends存放着zzz的好友信息，sdust\_friends中存放着sdust的好友列表。



⑵ 用户表（tb\_user）的结构及存放的数据



⑶ 某一用户好友表结果及数据



## 4 主要算法、类的描述

⑴ 客户端中主要类、算法的描述

① UI类

MainFrame、FriendsUI、ChatUI、ChangeUI、AddFriendUI、RegeditUI、ForgetUI分别为主窗口页面、好友列表页面、聊天窗口、修改密码窗口、添加好友窗口、注册窗口、忘记密码窗口。这一部分主要是显示处理和逻辑处理。

② 消息处理、通信类

ChatUIList类主要记录客户端打开的聊天页面，处理与好友的消息弹窗。Client类中有socket负责和服务器通信。ChatTreat类是客户端的消息处理中心，处理来自服务器的各种消息并做出相应。

⑵ 服务器端主要类、算法的描述

① UI类

服务器端只有一个UI页面——StartServerFrame，因为服务器没有太多消息要显示，所以一个启动窗口即可。

② 消息处理、通信类

Service类中有socket，负责与客户端建立通信，每个建立的通信都存储在SocketList中，供服务器查询哪些用户上线。ServerThread类是服务器端的控制中心，负责处理来自用户端的消息，并转发给正确的用户，有时还会对数据库进行操作。

③ 数据库处理相关

DBHelper类负责和数据库建立连接，UserService类负责处理具体的和数据库交互的内容，如查询用户账号和密码是否匹配、修改密码、注册用户、忘记密码、添加好友等操作。

## 5 主要的代码段

⑴ 客户端的主要代码段（略去包名）

① 主页面（MainFrame）

public class MainFrame extends JFrame implements ActionListener, FocusListener {

private static final long serialVersionUID = 1L;

private static final String \_txt\_account = "QQ密码/手机/邮箱";

private static final String \_txt\_pwd = "密码";

private static final String \_txt\_title = "QQ登录";

private static final String \_txt\_registe = "注册";

private static final String \_txt\_forget = "忘记密码";

private static final String \_txt\_blank = "";

private JTextField account;

private JPasswordField pwd;

private JLabel upper\_frame;

private JPanel lower\_frame, center\_frame;

private JButton login, register, forget;

MainFrame() {

//部分的形成

init();

//整体形成

add(upper\_frame, BorderLayout.NORTH);

add(center\_frame, BorderLayout.CENTER);

add(lower\_frame, BorderLayout.SOUTH);

ImageIcon logo = new ImageIcon("image/logo.jpg"); //左上方小图标

setIconImage(logo.getImage());

setBounds(500, 230, 430, 360); //设定大小及位置

setResizable(false); //登录框大小固定，不允许通过拖、拉改变大小

setDefaultCloseOperation(JFrame.HIDE\_ON\_CLOSE); //设置窗口右上角的叉号，点击叉号窗口关闭 注意不能EXIT\_ON\_CLOSE做参数的，用它时候使用的是System.exit方法退出应用程序。故会关闭所有窗口。

setTitle(\_txt\_title);

setVisible(true); //说明数据模型已经构造好了,允许JVM可以根据数据模型执行paint方法开始画图并显示到屏幕上，一般放在最后一句

}

public void init() {

//账号输入块

account = new JTextField(\_txt\_account);

account.setName("account");

account.setForeground(Color.gray);

account.addFocusListener(this); //产生事件响应用户行为

//密码输入块

pwd = new JPasswordField(\_txt\_pwd);

pwd.setName("pwd");

pwd.setForeground(Color.gray);

pwd.setEchoChar('\0'); //启动后密码框内一定为 “密码”

pwd.addFocusListener(this);

//注册模块

register = new JButton(\_txt\_registe);

register.setBorderPainted(false);

register.setBorder(BorderFactory.createRaisedBevelBorder()); //斜面边框（凸）

register.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));

register.addActionListener(this);

//忘记密码模块

forget = new JButton(\_txt\_forget);

forget.setBorderPainted(false);

forget.setBorder(BorderFactory.createRaisedBevelBorder());

forget.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));

forget.addActionListener(this);

//login按钮模块

login = new JButton();

ImageIcon login\_image = new ImageIcon("image/login\_image.png");

login\_image.setImage(login\_image.getImage().getScaledInstance(430, 35, Image.SCALE\_DEFAULT));

login.setIcon(login\_image);

login.setBackground(Color.white);

login.setBorderPainted(false); //如果进度条应该绘制边框，则为 true；否则为 false

login.setBorder(null); //设置此组件的边框 无

login.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR)); //将光标设为 “小手”形状

login.addActionListener(this);

//qq登录框架上半部分（无按钮之类的内容，只有一张图片）

ImageIcon upper\_image = new ImageIcon("image/upper\_image.png");

upper\_image.setImage(upper\_image.getImage().getScaledInstance(430, 160, Image.SCALE\_DEFAULT));

upper\_frame = new JLabel(upper\_image);

//qq登录中间部分 (账号、密码、 注册、forget)

center\_frame = new JPanel();

center\_frame.setName("center\_frame");

center\_frame.setLayout(null);

center\_frame.setLayout(new GridLayout(3, 3, 3, 15)); //这里用到3行3列的空间, 用空格填充

center\_frame.add(new JLabel(\_txt\_blank, JLabel.CENTER));

center\_frame.add(account);

center\_frame.add(new JLabel(\_txt\_blank, JLabel.CENTER));

center\_frame.add(new JLabel(\_txt\_blank, JLabel.CENTER));

center\_frame.add(pwd);

center\_frame.add(new JLabel(\_txt\_blank, JLabel.CENTER));

center\_frame.add(register);

center\_frame.add(new JLabel(\_txt\_blank, JLabel.CENTER));

center\_frame.add(forget);

center\_frame.setBackground(Color.white);

//qq登录框架的下半部分login

lower\_frame = new JPanel();

lower\_frame.setName("lower\_frame");

lower\_frame.setLayout(null);

//lower\_frame.setLayout(new GridLayout(1, 3, 3, 15));

lower\_frame.setLayout(new GridLayout(0, 1));

lower\_frame.add(login);

}

//按钮的点击事件用actionPerformed

@Override

public void actionPerformed(ActionEvent e){

/\*

\* 1：如果点击了登录按钮 首先判断帐号或者密码是否为空 然后封装为CommandTranser对象 向服务器发送数据 服务器通过与数据库的比对

\* 来验证帐号密码，

\* 2：如果点击了注册账号就弹出注册页面, 信息填写完整后连接服务器

\* 3：如果点击了忘记密码弹出找回密码页面

\*/

//处理登录(login)页面

if(e.getSource() == login){

String user\_name = account.getText().trim();

String user\_pwd = new String(pwd.getPassword()).trim();

if("".equals(user\_name) || user\_name == null || \_txt\_account.equals(user\_name)) {

JOptionPane.showMessageDialog(null, "请输入帐号！！");

return;

}

if("".equals(user\_pwd) || user\_pwd == null || \_txt\_pwd.equals(user\_pwd)) {

JOptionPane.showMessageDialog(null, "请输入密码！！");

return;

}

User user = new User(user\_name, user\_pwd);

CommandTranser cmd = new CommandTranser();

cmd.setCmd("login");

cmd.setData(user);

cmd.setReceiver(user\_name);

cmd.setSender(user\_name);

//实例化客户端 连接服务器 发送数据 密码是否正确?

Client client = new Client(); //创建唯一的客户端（用于接受服务器发来的消息， socket接口），

client.sendData(cmd); //发送数据

cmd = client.getData(); //接受反馈的消息

if(cmd != null) {

if(cmd.isFlag()) {

this.dispose(); //关闭MainFrame页面

JOptionPane.showMessageDialog(null, "登陆成功");

user = (User)cmd.getData();

FriendsUI friendsUI = new FriendsUI(user, client); //将user的全部信息传到FriendsUI中，并将唯一与服务器交流的接口传到FriendUI中 这里传client仅为了发送消息

ChatTread thread = new ChatTread(client, user, friendsUI); //这里传client为了收消息， 整个客户端用一个 ChatTread，一个client

thread.start();

}else {

JOptionPane.showMessageDialog(this, cmd.getResult());

}

}

}

//处理注册(register)页面

if(e.getSource() == register){

RegisterUI registerUI = new RegisterUI(this);

//

}

//处理找回密码(forget)页面

if(e.getSource() == forget){

ForgetUI forgetUI = new ForgetUI(this);

}

}

//鼠标的点击或移动之类的用focuslistener

@Override

public void focusGained(FocusEvent e) {

//处理账号输入框

if(e.getSource() == account){

if(\_txt\_account.equals(account.getText())){

account.setText("");

account.setForeground(Color.BLACK);

}

}

//处理密码输入框

if(e.getSource() == pwd){

if(\_txt\_pwd.equals(pwd.getText())){

pwd.setText("");

pwd.setEchoChar('\*');

pwd.setForeground(Color.BLACK);

}

}

}

@Override

public void focusLost(FocusEvent e) {

//处理账号输入框

if(e.getSource() == account){

if("".equals(account.getText())){

account.setForeground(Color.gray);

account.setText(\_txt\_account);

}

}

//处理密码输入框

if(e.getSource() == pwd){

if("".equals(pwd.getText())){

pwd.setForeground(Color.gray);

pwd.setText(\_txt\_pwd);

pwd.setEchoChar('\0');

}

}

}

public static void main(String[] args) {

// TODO Auto-generated method stub

MainFrame mainframe = new MainFrame();

}

}

② 消息处理核心——ChatTread类

public class ChatTread extends Thread{

private Client client;

private boolean isOnline = true;

private User user; //如果同意好友请求， 则刷新好友列表

private FriendsUI friendsUI; //刷新好友列表用

private String username; //如果创建新的聊天窗口（chatUI)那么必须将username传进去 用来发送消息

public ChatTread(Client client, User user, FriendsUI friendsUI) {

this.client = client;

this.user = user;

this.friendsUI = friendsUI;

this.username = user.getUsername();

//this.chat\_windows = chat\_windows;

}

public boolean isOnline() {

return isOnline;

}

public void setOnline(boolean isOnline) {

this.isOnline = true;

}

//run()方法是不需要用户来调用的，当通过start方法启动一个线程之后，当线程获得了CPU执行时间，

//便进入run方法体去执行具体的任务。注意，继承Thread类必须重写run方法，在run方法中定义具体要执行的任务

@Override

public void run() {

if(!isOnline) {

JOptionPane.showMessageDialog(null, "unbelievable ！！！");

return;

}

while(isOnline) {

CommandTranser cmd = client.getData();

//与服务器端相同处理接收到的消息(命令)

//这里处理来自服务器的消息(命令)

if(cmd != null) {

execute(cmd);

//System.out.println(cmd.getCmd());

}

}

}

//处理消息(命令)

private void execute(CommandTranser cmd) {

//登录、忘记密码、注册消息未在此处处理

System.out.println(cmd.getCmd());

//聊天消息请求

if("message".equals(cmd.getCmd())) {

if(cmd.isFlag() == false) {

JOptionPane.showMessageDialog(null, cmd.getResult());

return;

}

//查询是否有与该好友的窗口该窗口

String friendname = cmd.getSender();

ChatUI chatUI = ChatUIList.getChatUI(friendname);

if(chatUI == null) {

chatUI = new ChatUI(username, friendname, username, client);

ChatUIEntity chatUIEntity = new ChatUIEntity();

chatUIEntity.setName(friendname);

chatUIEntity.setChatUI(chatUI);

ChatUIList.addChatUI(chatUIEntity);

} else {

chatUI.show(); //如果以前创建过仅被别的窗口掩盖了 就重新显示

}

Date date = new Date();

SimpleDateFormat sdf = new SimpleDateFormat(

"yy-MM-dd hh:mm:ss a");

String message = friendname + "说："

+ (String) cmd.getData() + "\t" + sdf.format(date)

+ "\n";

chatUI.getChatWin().append(message); //追加消息

return;

}

if("WorldChat".equals(cmd.getCmd())) {

//查询是否有与该好友的窗口该窗口

String friendname = cmd.getSender();

ChatUI chatUI = ChatUIList.getChatUI("WorldChat");

if(chatUI == null) {

chatUI = new ChatUI("WorldChat", "WorldChat", user.getUsername(), client);

ChatUIEntity chatUIEntity = new ChatUIEntity();

chatUIEntity.setName("WorldChat");

chatUIEntity.setChatUI(chatUI);

ChatUIList.addChatUI(chatUIEntity);

} else {

chatUI.show(); //如果以前创建过仅被别的窗口掩盖了 就重新显示

}

Date date = new Date();

SimpleDateFormat sdf = new SimpleDateFormat(

"yy-MM-dd hh:mm:ss a");

String message = friendname + "说："

+ (String) cmd.getData() + "\t" + sdf.format(date)

+ "\n";

chatUI.getChatWin().append(message); //追加消息

return;

}

if("requeste\_add\_friend".equals(cmd.getCmd())) {

if(cmd.isFlag() == false) {

JOptionPane.showMessageDialog(null, cmd.getResult());

return;

}

String sendername = cmd.getSender();

int flag = JOptionPane.showConfirmDialog(null, "是否同意" + sendername + "的好友请求", "好友请求", JOptionPane.YES\_NO\_OPTION);

System.out.println(flag);

if(flag == 0) {

cmd.setCmd("accept\_add\_friend");

} else {

cmd.setCmd("refuse\_add\_friend");

}

cmd.setSender(username);

cmd.setReceiver(sendername);

client.sendData(cmd);

return;

}

if("accept\_add\_friend".equals(cmd.getCmd())) {

JOptionPane.showMessageDialog(null, cmd.getResult());

return;

}

if("refuse\_to\_add".equals(cmd.getCmd())) {

JOptionPane.showMessageDialog(null, cmd.getResult());

return;

}

if("changepwd".equals(cmd.getCmd())) {

JOptionPane.showMessageDialog(null, cmd.getResult());

return;

}

return;

}

}

③ Client 类——负责通信

public class Client {

private int port = 2222;

private String Sever\_address = "127.0.0.1"; //服务器主机ip

private Socket socket;

//实例化， 建立连接

public Client(){

try {

socket = new Socket(Sever\_address, port);

} catch(UnknownHostException e) {

JOptionPane.showMessageDialog(null, "服务器端未开启");

}catch(IOException e) {

JOptionPane.showMessageDialog(null, "服务器端未开启");

}

}

public Socket getSocket() {

return socket;

}

public void setSocket(Socket socket) {

this.socket = socket;

}

//向服务端发送数据

public void sendData(CommandTranser cmd) {

ObjectOutputStream oos = null; //主要的作用是用于写入对象信息与读取对象信息。 对象信息一旦写到文件上那么对象的信息就可以做到持久化了

try {

if(socket == null) {

return;

}

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

oos.writeObject(cmd);

} catch(UnknownHostException e) {

JOptionPane.showMessageDialog(null, "服务器端未开启");

}catch(IOException e) {

JOptionPane.showMessageDialog(null, "服务器端未开启");

}

}

//接受服务端发送的消息

public CommandTranser getData() {

ObjectInputStream ois = null;

CommandTranser cmd = null;

if(socket == null) {

//System.out.println("weishenme");

return null;

}

try {

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

cmd = (CommandTranser) ois.readObject();

} catch (IOException e) {

return null;

} catch (ClassNotFoundException e) {

return null;

}

return cmd;

}

}

⑵ 服务器端主要代码段（略去包名）

① CommandTranser类——用于命令传递

public class CommandTranser implements Serializable {

private static final long serialVersionUID = 1L;

private String sender = null;// 发送者

private String receiver = null;// 接受者

private Object data = null; // 传递的数据

private boolean flag = false; // 指令的处理结果

private String cmd = null; // 服务端要做的指令

private String result = null; //处理结果

public String getSender() {

return sender;

}

public String setSender(String sender) {

return this.sender = sender;

}

public String getReceiver() {

return receiver;

}

public String setReceiver(String receiver) {

return this.receiver = receiver;

}

public Object getData() {

return data;

}

public Object setData(Object data) {

return this.data = data;

}

public boolean isFlag() {

return flag;

}

public boolean setFlag(boolean flag) {

return this.flag = flag;

}

public String getResult() {

return result;

}

public String setResult(String result) {

return this.result = result;

}

public String getCmd() {

return cmd;

}

public String setCmd(String cmd) {

return this.cmd = cmd;

}

}

② ServerThread类——服务器端的消息控制中心

public class ServerThread extends Thread{

private Socket socket;

public ServerThread(Socket socket) {

this.socket = socket;

}

@Override

public void run() {

ObjectInputStream ois = null;

ObjectOutputStream oos1 = null;

ObjectOutputStream oos2 = null;

//ObjectOutputStream oos3 = null;

while(socket != null) {

try {

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

CommandTranser cmd = (CommandTranser) ois.readObject();

//执行命令来自客户端的请求

cmd = execute(cmd);

//消息对话请求，服务器将sender发来的消息发送给receiver

if("message".equals(cmd.getCmd())) {

//如果 msg.ifFlag即 服务器处理成功 可以向朋友发送信息 如果服务器处理信息失败 信息发送给发送者本人

if(cmd.isFlag()) {

//System.out.println("对方在线");

oos1 = new ObjectOutputStream(SocketList.getSocket(cmd.getReceiver()).getOutputStream());

} else {

//System.out.println("对方未在线");

oos2 = new ObjectOutputStream(socket.getOutputStream());

}

}

if ("WorldChat".equals(cmd.getCmd())) {

HashMap<String, Socket> map = SocketList.getMap();

Iterator<Map.Entry<String, Socket>> it = map.entrySet().iterator();

while(it.hasNext()) {

Map.Entry<String, Socket> entry = it.next();

if(!entry.getKey().equals(cmd.getSender())) {

oos1 = new ObjectOutputStream(entry.getValue().getOutputStream());

oos1.writeObject(cmd);

}

}

continue;

}

//登录请求 将数据发送给sender

if ("login".equals(cmd.getCmd())) {

oos1 = new ObjectOutputStream(socket.getOutputStream());

}

//注册请求 将数据发送给sender

if ("register".equals(cmd.getCmd())) {

System.out.println("向客户端发送消息");

oos1 = new ObjectOutputStream(socket.getOutputStream());

}

//添加好友请求将数据发送给 receiver

if ("requeste\_add\_friend".equals(cmd.getCmd())) {

//在线，将请求发给receiver

if(cmd.isFlag()) {

oos1 = new ObjectOutputStream(SocketList.getSocket(cmd.getReceiver()).getOutputStream());

} else {

//不管在不在线都要向发送方提示消息发送成功

oos2 = new ObjectOutputStream(socket.getOutputStream());

}

}

//同意添加好友请求将数据发送给 receiver和sender

if ("accept\_add\_friend".equals(cmd.getCmd())) {

//无论是否成功插入数据库都要将结果反馈，但有可能最初请求的客户下线了

oos1 = new ObjectOutputStream(socket.getOutputStream());

if(SocketList.getSocket(cmd.getReceiver()) != null) {

oos2 = new ObjectOutputStream(SocketList.getSocket(cmd.getReceiver()).getOutputStream());

}

}

//拒绝添加好友请求将数据发送给 receiver

if ("refuse\_to\_add".equals(cmd.getCmd())) {

//被拒绝方在线

if(cmd.isFlag()) {

oos1 = new ObjectOutputStream(SocketList.getSocket(cmd.getReceiver()).getOutputStream());

}else { //被拒方不在线则向拒绝方发送消息

oos2 = new ObjectOutputStream(socket.getOutputStream());

}

}

//修改资料请求 发送给sender 功能暂未实现

if ("changeinfo".equals(cmd.getCmd())) {

oos1 = new ObjectOutputStream(socket.getOutputStream());

}

//修改密码请求 将数据发送给sender

if ("changepwd".equals(cmd.getCmd())) {

oos1 = new ObjectOutputStream(socket.getOutputStream());

}

//忘记密码 发送给sender

if ("forgetpwd".equals(cmd.getCmd())) {

oos1 = new ObjectOutputStream(socket.getOutputStream());

}

//用户下线

if("logout".equals(cmd.getCmd())) {

//

}

if(oos1 != null) {

oos1.writeObject(cmd);

}

if(oos2 != null) {

oos2.writeObject(cmd);

}

} catch(IOException e) {

socket = null;

} catch (ClassNotFoundException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

//处理客户端发来的命令

private CommandTranser execute(CommandTranser cmd) {

//登录请求

if("login".equals(cmd.getCmd())) {

UserService userservice = new UserService();

User user = (User)cmd.getData();

cmd.setFlag(userservice.checkUser(user));

//如果登陆成功，将该客户端加入已经连接成功的map集合里面 并且开启此用户的接受线程

if(cmd.isFlag()) {

// 将该线程加入连接成功的map集合

SocketEntity socketEntity = new SocketEntity();

socketEntity.setName(cmd.getSender());

socketEntity.setSocket(socket);

SocketList.addSocket(socketEntity);

//从数据库获取其好友列表并将其好友列表发送至客户端

cmd.setData(userservice.getFriendsList(user));

cmd.setResult("登陆成功");

} else {

cmd.setResult("密码错误");

}

}

//注册请求

if("register".equals(cmd.getCmd())) {

UserService userservice = new UserService();

User user = (User)cmd.getData();

cmd.setFlag(userservice.registerUser(user));

//如果注册成功

if(cmd.isFlag()) {

SocketEntity socketEntity = new SocketEntity();

socketEntity.setName(cmd.getSender());

socketEntity.setSocket(socket);

SocketList.addSocket(socketEntity);

cmd.setData(userservice.getFriendsList(user));

//刚注册的肯定没有好友

cmd.setResult("注册成功");

} else {

cmd.setResult("注册失败可能该用户已存在");

}

}

//修改资料请求 功能暂未实现

if("changeInfo".equals(cmd.getCmd())) {

UserService userservice = new UserService();

User user = (User)cmd.getData();

cmd.setFlag(userservice.changeInfo(user));

if(cmd.isFlag()) {

cmd.setResult("修改信息成功");

} else {

cmd.setResult("修改信息失败");

}

}

//添加好友

if("requeste\_add\_friend".equals(cmd.getCmd())) {

//检查用户是否在线

if(SocketList.getSocket(cmd.getReceiver()) != null) {

cmd.setFlag(true);

cmd.setResult("对方收到了您的好友请求");

} else {

cmd.setFlag(false);

cmd.setResult("当前用户不在线或者改用户不存在");

}

}

//同意添加好友请求

if("accept\_add\_friend".equals(cmd.getCmd())) {

UserService userservice = new UserService();

cmd.setFlag(userservice.addFriend(cmd.getReceiver(), cmd.getSender()));

if(cmd.isFlag()) {

cmd.setResult("好友添加成功请重新登陆刷新");

} else {

cmd.setResult("服务器故障导致添加好友失败或者您们已经为好友");

}

}

//拒绝添加好友

if("refuse\_to\_add".equals(cmd.getCmd())) {

//检查是否在线

if(SocketList.getSocket(cmd.getReceiver()) != null) {

cmd.setFlag(true);

cmd.setResult("您被 " + cmd.getSender() + " 拒绝了");

} else {

cmd.setFlag(false);

cmd.setResult("对方不在线不知道你拒绝了他的好友请求");

}

}

//发送消息指令

if("message".equals(cmd.getCmd())) {

//检查是否在线

if(SocketList.getSocket(cmd.getReceiver()) != null) {

//System.out.println("神奇");

cmd.setFlag(true);

//cmd.setResult("对方成功收到您的消息");

} else {

//System.out.println("神奇啊");

cmd.setFlag(false);

cmd.setResult("当前用户不在线");

}

}

if("WordChat".equals(cmd.getCmd())) {

cmd.setFlag(true);

}

//忘记密码指令 这里最后要讲用户的问题和答案返回

if("forgetpwd".equals(cmd.getCmd())) {

UserService userservice = new UserService();

User user = (User)cmd.getData();

user = userservice.getUser(user);

//如果用户存在

if(user != null ) {

cmd.setResult("查询成功");

cmd.setData(user);

cmd.setFlag(true);

} else {

cmd.setResult("用户可能不存在");

cmd.setFlag(false);

}

}

if ("changepwd".equals(cmd.getCmd())) {

UserService userservice = new UserService();

User user = (User)cmd.getData();

cmd.setFlag(userservice.changePassword(user));

System.out.println("there 111 ");

System.out.println(user.getUsername());

if(cmd.isFlag()) {

cmd.setResult("修改密码成功");

}else {

cmd.setResult("修改密码失败");

}

}

if("logout".equals(cmd.getCmd())) {

SocketList.getSocket(cmd.getSender());

}

return cmd;

}

}

③ DBHelper类——连接数据库

public class DBHelper {

private static final String driver = "com.mysql.cj.jdbc.Driver";

private static final String url = "jdbc:mysql://localhost:3306/myqquser?&useSSL=false&serverTimezone=UTC";

private static final String username = "root";

private static final String password = "zzzz";

private static Connection con = null;

//静态代码负责加载驱动

static {

try {

Class.forName(driver); //Class.forName(xxx.xx.xx)的作用是要求JVM查找并加载指定的类，也就是说JVM会执行该类的静态代码段

} catch(ClassNotFoundException e) {

e.printStackTrace();

}

}

public static Connection getConnection() {

if(con == null) {

try {

con = DriverManager.getConnection(url, username, password);

} catch(SQLException e) {

e.printStackTrace();

}

}

return con;

}

}

④ UserService类——对数据库进行增删查改

public class UserService {

//login验证账号密码

public boolean checkUser(User user) {

PreparedStatement stmt = null; //PreparedStatement是用来执行SQL查询语句的API之一

Connection conn = null; //与特定数据库的连接（会话）。在连接上下文中执行 SQL 语句并返回结果

ResultSet rs = null; //是数据中查询结果返回的一种对象，可以说结果集是一个存储查询结果的对象，但是结果集并不仅仅具有存储的功能，他同时还具有操纵数据的功能，可能完成对数据的更新等

conn = DBHelper.getConnection();

String sql = "select \* from tb\_user where user\_name =? and user\_pwd =?";

try {

stmt = conn.prepareStatement(sql);

stmt.setString(1, user.getUsername());

stmt.setString(2, user.getUserpwd());

rs = stmt.executeQuery();

if(rs.next()) {

return true;

}

} catch(SQLException e) {

e.printStackTrace();

} finally{

try {

if(rs != null) {

rs.close();

}

if(stmt != null) {

stmt.close();

}

} catch(SQLException e) {

e.printStackTrace();

}

}

return false;

}

//登陆后向客户端发送其好友列表

public User getFriendsList(User user) {

PreparedStatement stmt = null;

Connection conn = null;

ResultSet rs = null;

conn = DBHelper.getConnection();

String sql = "select \* from " + user.getUsername() + "\_friends";

ArrayList<String> friendslist = new ArrayList<String>(); //这里假设好友不超过20个

try {

stmt = conn.prepareStatement(sql);

//stmt.setString(1, user.getUsername() + "\_friends"); 这样的话会报错

rs = stmt.executeQuery();

int count = 0;

while(rs.next()) {

friendslist.add(rs.getString(2)); //获取好友name

count++;

}

user.setFriendsNum(count);

user.setFriendsList(friendslist);

return user;

} catch(SQLException e) {

e.printStackTrace();

} finally{

try {

if(rs != null) {

rs.close();

}

if(stmt != null) {

stmt.close();

}

} catch(SQLException e) {

e.printStackTrace();

}

}

return user;

}

//用户注册

public boolean registerUser(User user) {

PreparedStatement stmt1 = null; //PreparedStatement是用来执行SQL查询语句的API之一

PreparedStatement stmt2 = null;

PreparedStatement stmt3 = null;

Connection conn = null;

ResultSet rs = null;

int insertFlag = 0;

int creatFlag = 0;

conn = DBHelper.getConnection();

String sql = "select \* from tb\_user where user\_name =?";

String insertusersql = "insert into tb\_user (user\_name, user\_pwd, user\_question, user\_ans) values(?, ?, ?, ?)";

String creatfriendstabsql = "CREATE TABLE " + user.getUsername() + "\_friends " + "(id INT NOT NULL AUTO\_INCREMENT, name VARCHAR(45) NOT NULL, PRIMARY KEY (id))";

try {

stmt1 = conn.prepareStatement(sql);

stmt1.setString(1, user.getUsername());

rs = stmt1.executeQuery();

if(rs.next()) {

System.out.println("该用户已存在" + user.getUsername() + "\*\*\*");

//用户已被注册

return false;

}

else {

System.out.println("该用户不存在" + user.getUsername() + "\*\*\*");

//向用户表插入数据

stmt2 = conn.prepareStatement(insertusersql);

stmt2.setString(1, user.getUsername());

stmt2.setString(2, user.getUserpwd());

stmt2.setString(3, user.getUserQuestion());

stmt2.setString(4, user.getUserAnswer());

insertFlag = stmt2.executeUpdate();

System.out.println("向表中插入数据" + user.getUsername() + "\*\*\*" + insertFlag);

//创建好友表

stmt3 = conn.prepareStatement(creatfriendstabsql);

creatFlag = stmt3.executeUpdate();

System.out.println("创建表" + user.getUsername() + "\*\*\*" + creatFlag);

if(insertFlag == 1) {

return true;

}

System.out.println("不高兴" + user.getUsername() + "\*\*\*");

//return true;

}

} catch(SQLException e) {

e.printStackTrace();

} finally{

try {

if(rs != null) {

rs.close();

}

if(stmt1 != null) {

stmt1.close();

}

if(stmt2 != null) {

stmt2.close();

}

if(stmt3 != null) {

stmt3.close();

}

} catch(SQLException e) {

e.printStackTrace();

}

}

return false;

}

//添加好友

public boolean addFriend(String sender, String receiver) {

PreparedStatement stmt1 = null;

PreparedStatement stmt2 = null;

Connection conn = null;

int updateResult1 = 0;

int updateResult2 = 0;

conn = DBHelper.getConnection();

String sql1 = "insert into " + sender + "\_friends (name) values(?)";

//String sql1 = "insert into ? (name) values(?)";

String sql2 = "insert into " + receiver + "\_friends (name) values(?)";

//String sql2 = "insert into ? (name) values(?)";

try {

stmt1 = conn.prepareStatement(sql1);

stmt2 = conn.prepareStatement(sql2);

stmt1.setString(1, receiver);

stmt2.setString(1, sender);

updateResult1 = stmt1.executeUpdate();

updateResult2 = stmt2.executeUpdate();

if(updateResult1 == 1 && updateResult2 == 1) {

return true;

}

else {

// 如果插入不成功的话，应该将插入成功的删除....这里不做处理了

}

} catch(SQLException e) {

e.printStackTrace();

} finally{

try {

if(stmt1 != null) {

stmt1.close();

}

if(stmt2 != null) {

stmt2.close();

}

} catch(SQLException e) {

e.printStackTrace();

}

}

return false;

}

//修改信息

public boolean changeInfo(User user) {

return false;

}

//修改密码 忘记密码

public boolean changePassword(User user) {

PreparedStatement stmt1 = null; //PreparedStatement是用来执行SQL查询语句的API之一

PreparedStatement stmt2 = null; //PreparedStatement是用来执行SQL查询语句的API之一

Connection conn = null; //与特定数据库的连接（会话）。在连接上下文中执行 SQL 语句并返回结果

ResultSet rs = null; //是数据中查询结果返回的一种对象，可以说结果集是一个存储查询结果的对象，但是结果集并不仅仅具有存储的功能，他同时还具有操纵数据的功能，可能完成对数据的更新等

int updateFlag = 0;

conn = DBHelper.getConnection();

//String sql = "select \* from tb\_user where user\_question =? and user\_ans =?";

String updatesql = "update tb\_user set user\_pwd =? where user\_name = ?";

try {

stmt2 = conn.prepareStatement(updatesql);

stmt2.setString(1, user.getUserpwd());

stmt2.setString(2, user.getUsername());

updateFlag = stmt2.executeUpdate();

if(updateFlag == 1)

return true;

//}

} catch(SQLException e) {

e.printStackTrace();

} finally{

try {

if(rs != null) {

rs.close();

}

if(stmt1 != null) {

stmt1.close();

}

if(stmt2 != null) {

stmt2.close();

}

} catch(SQLException e) {

e.printStackTrace();

}

}

return false;

}

//获得用户的相关信息

public User getUser(User user) {

PreparedStatement stmt1 = null;

PreparedStatement stmt2 = null;

Connection conn = null;

ResultSet rs = null;

conn = DBHelper.getConnection();

String sql = "select \* from tb\_user where user\_name =?";

try {

stmt1 = conn.prepareStatement(sql);

stmt1.setString(1, user.getUsername());

rs = stmt1.executeQuery();

if(rs.next()) {

user.setUsername(rs.getString("user\_name"));

user.setUserAnswer(rs.getString("user\_ans"));

user.setUserQuestion(rs.getString("user\_question"));

return user;

}

} catch(SQLException e) {

e.printStackTrace();

} finally{

try {

if(rs != null) {

rs.close();

}

if(stmt1 != null) {

stmt1.close();

}

if(stmt2 != null) {

stmt2.close();

}

} catch(SQLException e) {

e.printStackTrace();

}

}

return null;

}

}

# 第四章 测试和演示

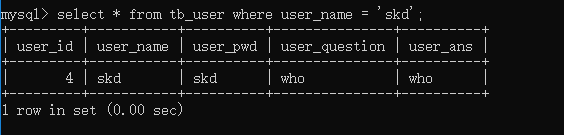
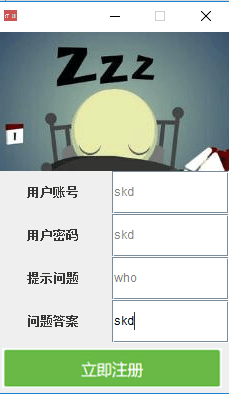
## 1 测试内容及预测结果

|  |  |  |  |
| --- | --- | --- | --- |
| 测试号 | 测试名称 | 测试操作 | 预期结果 |
| 1 | 注册用户 | 注册用户名为skd，密码为sdk的用户 | 数据库tb\_user中出现该用户信息 |
| 2 | 登录 | 登录账号 zzz | 登录成功显示好友列表 |
| 3 | 忘记密码 | 找回zzz的密码，新密码为sk | 密码找回成功 |
| 4 | 用户聊天 | zzz与 sdust聊天 | 双方收到消息 |
| 5 | 添加好友 | zzz 添加 skd为好友 | zzz 和skd好友列表出现对方 |
| 6 | 修改密码 | 修改zzz的密码 | zzz的密码修改成功 |
| 7 | 群聊 | zzz发起群聊，sdust和skd均收到消息 | sdust和skd均收到消息 |

## 2 测试结果与截图

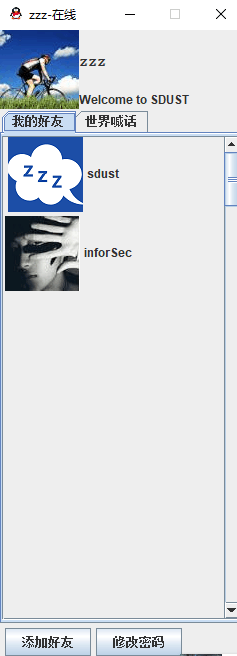
⑴ 测试点一

测试结果与预期结果一致，注册成功！截图如下：



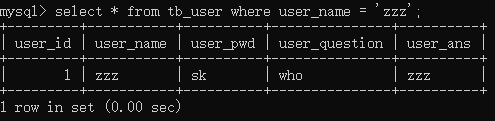
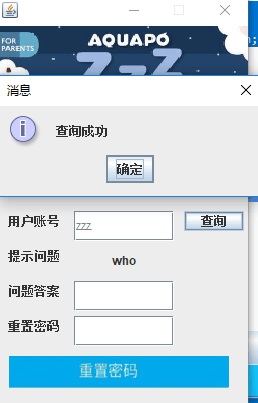
⑵ 测试点二

测试结果与预期结果一致，登录成功！截图如下：



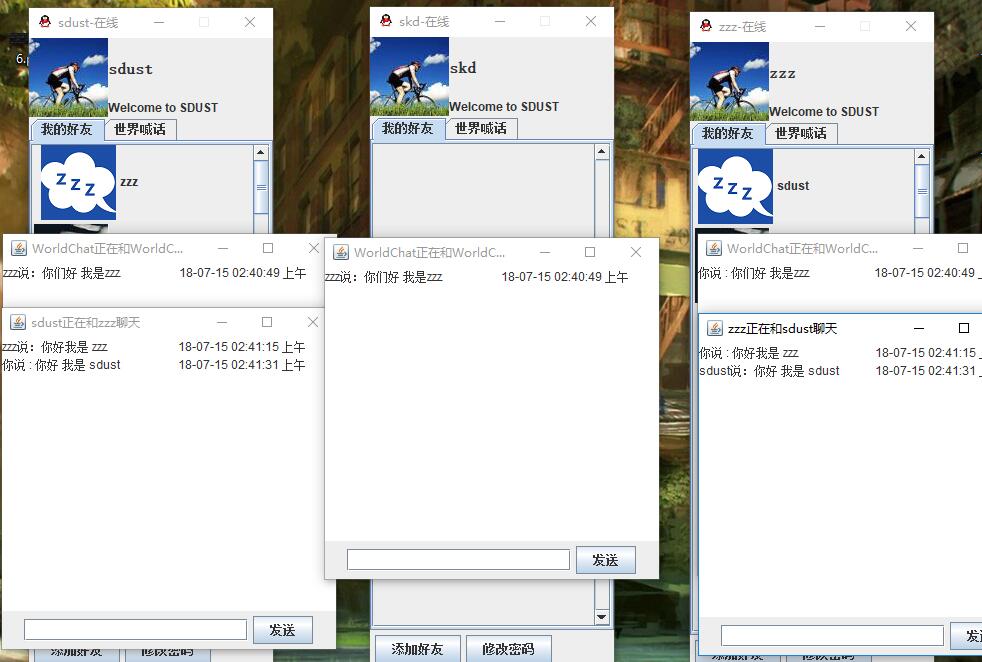
⑶ 测试点三

测试结果与预期结果一致，修改密码成功！截图如下：



⑷ 测试点四、七一起测试

测试结果与预期结果一致，私聊群聊成功！截图如下：



经测试测试点5、6也均实验成功，此处不附截图了。

# 第五章 总结

这次设计经过不懈的努力，程序基本满足聊天的需要。完成后的程序实现了用户注册、用户登录、忘记密码、写消息、发消息、回复、群聊、添加好友、修改密码等功能。给用户较好的体验。完成了登录页面、注册页面、好友列表、忘记密码、修改密码等页面框架，使得程序较美观。由于一个人完成这个项目，时间比较紧张，对网络传输的一些知识不太熟悉，所以没有完成文件传输功能。

完成的项目功能不太完善，虽然完成了群聊（世界喊话），但不能和好友列表中的具体几个人进行群聊。在进行项目时候，UI部分代码的编写不够仔细，所以部分UI不够美观。世界喊话的实现太过草率，整个页面的利用率低。后续可以在世界喊话页面增加群聊功能，在数据库中为每个人创建一个群表，记录每个人加入的群，完成真正的群聊。对UI部分进行优化，使整个项目更美观。

程序设计期间，通过询问老师、同学和网上搜索相关解答等多种方式学习到很多课堂上没有的知识，增强了解决实际问题的能力。在此之前，对于java编程技术没有很深入的认识，对其工作方式和原理缺乏透彻的认识。在这段时间里，我认真的学习了java语言开发，对编程思想有了进一步的体会，养成了一些良好的编程习惯。程序虽然完成了，但是距离完美还存在一定的差距，用java开发大型项目还有很多我需要去努力学习的知识。希望自己能不断学习和实践，争取以后做得更好。