**目 录**

分页（如果有目录，则目录与正文分页）

实验题目：P\_P网络聊天程序

1. 实验目的：
2. 掌握Java面向对象编程
3. 掌握Java网络编程
4. 掌握Java多线程编程
5. 掌握Java数据库编程
6. 掌握Java图形用户界面编程
7. 掌握Java泛型及集合框架
8. 掌握Java异常处理

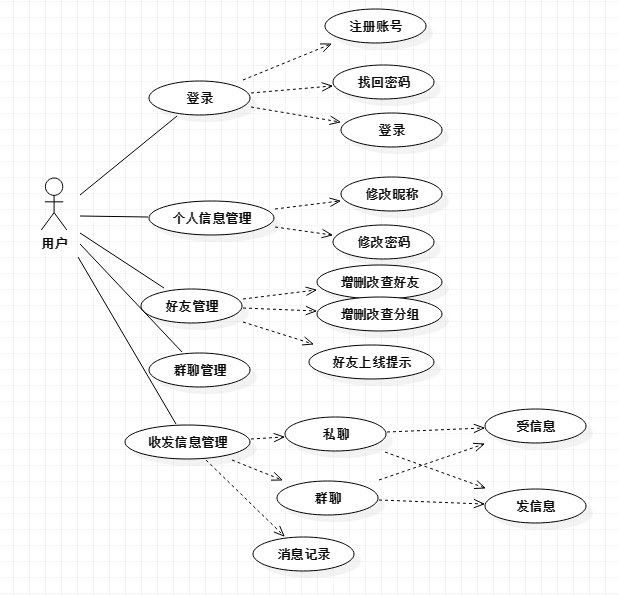
2．实验内容：

基于Java语言完成一个类QQ的简单网络聊天程序，要求如下：

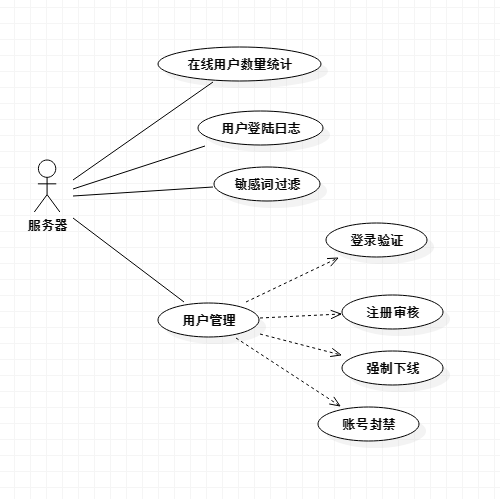
1. 支持C/S模式的网络聊天程序
2. 服务端功能
3. 在线用户数量统计
4. 用户登陆日志
5. 敏感词过滤
6. 用户管理（注册审核、登录验证、账号封禁）
7. 客户端功能
8. 登录
9. 个人信息管理
10. 好友管理
11. 群聊管理
12. 收发信息管理
13. 能查看历史纪录
14. 图形用户界面Swing

3．实验过程：

***需求分析***

***客户端***

***服务端***



编码思路：

首先用Swing GUI编程完成界面设计以及实现

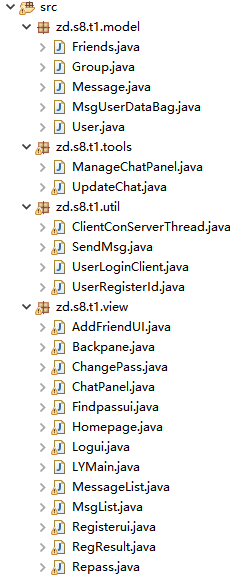
服务器端通过利用HashMap管理连接了服务端的客户端Socket，通过用户id对应保存Socket，整个系统的服务端连接数据库，一切逻辑操作和数据访问都是在服务端完成，客户端只负责将系统展示给客户，而客户端和服务端通信通过客户端发送数据包请求，服务端识别客户端的数据请求类型，进行相应类型的操作及访问，然后将处理以后的数据通过数据包的形式返回给客户端，客户端识别收到的数据包类型及数据，然后进行相应的展示操作。客户端也用HashMap管理好友聊天界面，通过唯一的用户id识别聊天界面，方便用户使用，在每一次登陆成功以后，客户端都重开一个线程，用以保持与服务器的通信，前后端之间通过对象流管道进行数据包的发送和接受，每当接入一个新客户端，服务器也开辟一个线程用以保持通信，虽然线程的开辟的昂贵的，建议使用线程池，但是在本次实验中最多也就同时在线3个客户端，所以没有必要使用线程池。而该软件的核心功能------聊天，其实也就是客户端发送给服务端的数据包，服务端转发给指定的客户端从而完成该功能。

4．实验结果：

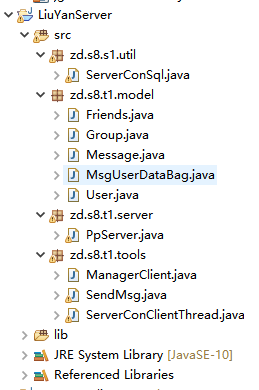
目前实现了好友聊天，注册帐户，登录，消息提醒，修改密码，添加好友的功能

以下是部分截图

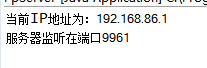
客户端项目的目录结构



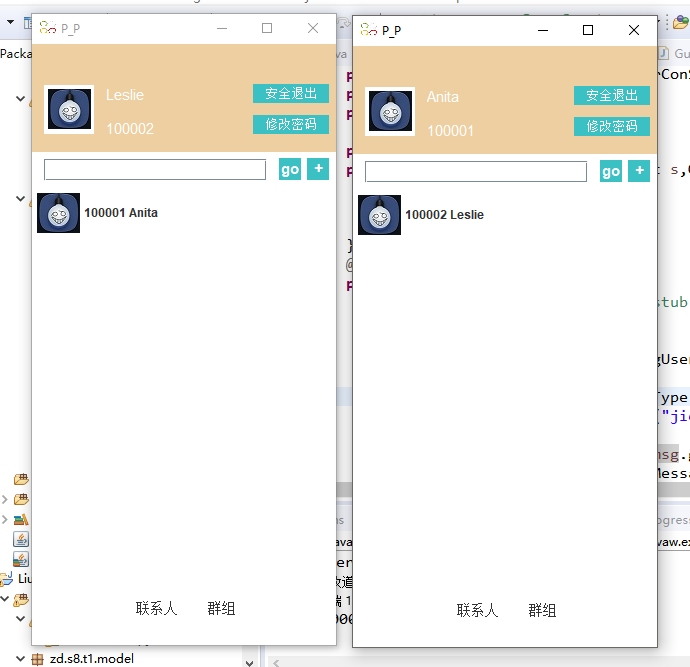
服务端项目结构



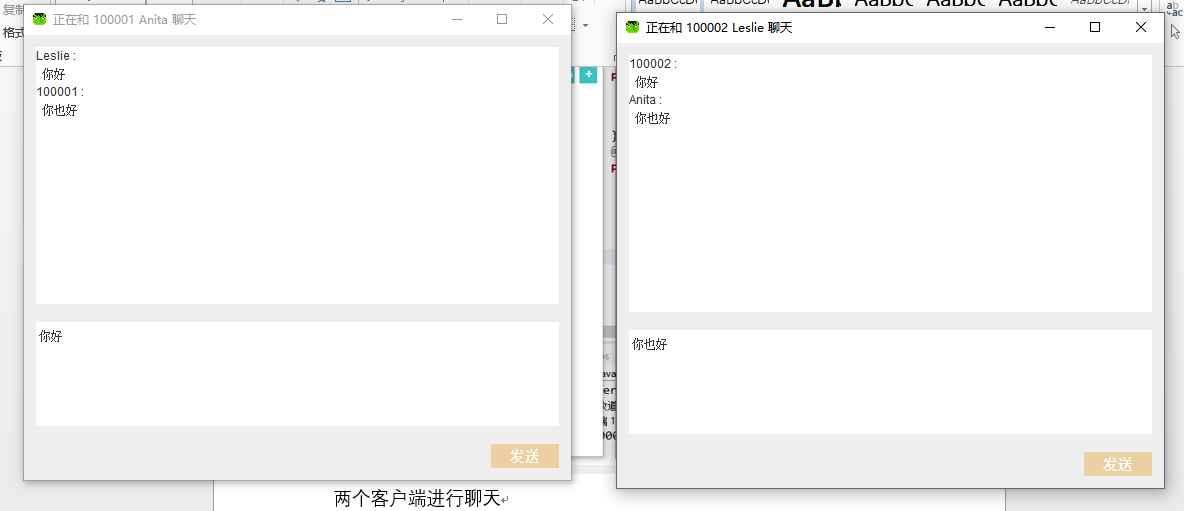
服务端打开，两个客户端连接





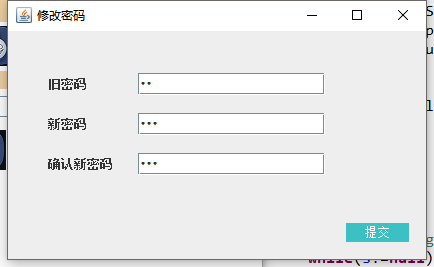


两个客户端进行聊天



100001是用户账号，后面Anita就是用户的昵称

修改密码



5．实验体会：

聊天程序不能是单线程的，对于收信息与发信息必须是互相独立的，所以应该给收信息和发信息各自分配一条线程，而对于多个客户端与服务器之间的连接也要是相互独立的

本次实验让我学到很多，关于即时聊天软件的客户端，服务端之间的通信和多线程还有集合框架比如ArrayList<>和HashMap<>,这些都让我受益匪浅，虽然没有把需求中的功能全做出来，但是我对Java多线程编程，网络编程，数据库编程的知识都有了长足的进步，这个项目其实挺有深度的，很适合用来学习Java

在实验中也遇到很多问题，下面是主要遇到的问题：

1. 需求阶段是模仿QQ的需求的，加入了很多没有实用性的功能，需求阶段花费了大量的时间弄清楚IM概念和IM类型软件的功能设计，不过好在有现成的微信（个人认为微信的设计是优于QQ的），所以后来需求大部分是从微信获得的；
2. 界面设计阶段可能太过于追求模仿QQ了，一开始画了很多界面，有很多无用功能的界面，并且难度大，代码量大，如果仅仅是一个人在一个月内是不能真正做出来的，所以后来一直缩减，甚至后来直接把模仿的界面全部删除重新画，自己画了图标，自己画了背景和头像，界面这个阶段花费了很多时间；
3. 由于前面做的无用功太多导致编码时间过于少，加上课程多，掌握的知识还很少，所以编码有点困难。在编码阶段主要遇到几个问题：
4. 由于客户端和服务端都需要序列化的数据包，两端的数据包序列号应该一致，一开始并没有设置序列号，后来导致服务端接收识别不到客户端发送的数据包；
5. 一开始并没有想到用HashMap管理聊天面板，导致一个账号和两个聊天时，如果和其中一个聊，另外一个发送过来的数据接受不了，这是因为没有管理聊天面板的聊天只有一个流管道，只要被占用了就一直监听数据，如果没有信息发来就阻塞，导致和别人的聊天不能完成，后来将每个聊天面板加入静态HashMap管理，这样就可以将流独立出来。这和服务端用HashMap管理客户端Socket是一个道理；
6. 编码注册功能时，一开始考虑不周，本应该在注册时将好友和群组列表创建起来，没有考虑到导致后来做好友列表获取和添加好友时必须回过头重新编码和重新建数据表；
7. 好友列表界面利用Jlist实现，用Jlable装载图片和昵称，再用ListModel完成带图片的Jlist组件，在此过程中也遇到了很多问题，主要就是如何让Jlist实时刷新好友列表，加了好友以后不需要重新登录就可以刷新列表完成聊天，后来我是这样解决的，在加好友成功后，立刻发送一个数据包告诉服务器，我客户端加好友成功，请求返回好友列表信息，返回以后重绘Jlist即可
8. 调试期间也是很困难的，很多Bug，有很多小问题让我措手不及，比如返回的数据包类型正确然而数据中有空值，这是由于我在编码时考虑不周导致的，很多地方必须加判断与异常捕获，其实90%的bug都是由于粗心与考虑不周导致的，只要认真调试总能解决，虽然bug不会全部解决完，但尽量减少bug吧

编码时候也没有特意收集问题，所以也没写出多少问题来，现在想想，以后也应该多收集问题，总结问题。

本次实验其实最大的收获就是锻炼了敲代码的恒心和毅力还有自主学习的能力，虽然老师上课知识教授我们知识，在实验中遇到的问题总不能一直等老师讲吧，所以自己查资料的能力很重要，本次实验让我加强了使用StackOverflow的能力，也因为这个，我英语能力也提升了一点。

6．程序代码（作为附录，代码中要有详细注释）

核心代码

PpServer.java

登录时候客户端发起第一次请求，客户端监听以后，判断登陆条件，新开一个线程与客户端保持通讯，将连接的socket加入HashMap用来管理客户端

**public** **class** PpServer {

**public** **static** **void** main(String[] args) **throws** UnknownHostException {

InetAddress host=InetAddress.*getLocalHost*();

String ip=host.getHostAddress();

System.***out***.println("当前IP地址为："+ip);

**try** {

System.***out***.println("服务器监听在端口9961");

ServerSocket ss=**new** ServerSocket(9961);

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

//客户端第一次尝试连接服务器

Socket s=ss.accept();

ObjectInputStream ois=**null**;

DataOutputStream dos=**null**;

ObjectOutputStream oos=**null**;

System.***out***.println("we have a new client coming");

ois=**new** ObjectInputStream(s.getInputStream());

Message msg=(Message)ois.readObject();

User user=(User)ois.readObject();

System.***out***.println("接收到的UserPass: "+user.getPass());

**if**(msg.getMsgType().equals("1")) {

String a=*isLogin*(msg,user);

System.***out***.println("从客户端接收道德消息类型 "+msg.getMsgType()+" 接收到的用户密码: "+user.getPass());

dos=**new** DataOutputStream(s.getOutputStream());

oos=**new** ObjectOutputStream(s.getOutputStream());

dos.writeUTF(a);

oos.writeObject(*getUser*(user));

System.***out***.println("返回给客户端 "+a);

ServerConSql scs=**new** ServerConSql();

MsgUserDataBag msgb=**new** MsgUserDataBag();

msgb.setMsg(msg);

msgb.setUser(user);

MsgUserDataBag mudb=**new** MsgUserDataBag();

mudb=scs.getFriendList(msgb);

oos.writeObject(mudb);

//管理Socket

ServerConClientThread scct =**new** ServerConClientThread(s,oos,ois);

scct.start();

ManagerClient.*addSocket*(user, s);

}

**else** **if**(msg.getMsgType().equals("4")) {

Message a=*isRegster*(msg,user);

oos=**new** ObjectOutputStream(s.getOutputStream());

oos.writeObject(a);

System.***out***.println("返回给客户端 消息类 "+a.getMsgType()+"返回个客户端账号"+a.getReceiver());

s.close();

}

}

}**catch**(IOException | ClassNotFoundException e) {

e.printStackTrace();

}

}

/\*\*

\*

\* **@param** msg

\* **@param** user

\* **@return**

\*/

**public** **static** String isLogin(Message msg,User user) {

String b="2";

ServerConSql scs=**new** ServerConSql();

User u=scs.getUser(user.getId());

**if**(msg.getMsgType().equals("1")&&u.getId().equals(user.getId())&&u.getPass().equals(user.getPass())) {

b="1";

}

**return** b;

}

**public** **static** Message isRegster(Message msg,User user) {

Message m=**new** Message();

ServerConSql scs=**new** ServerConSql();

m.setReceiver(scs.regster(user));

//m.setReceiver(scs.regster(user));

System.***out***.println("ww"+m.getReceiver());

m.setMsgType("4");

**return** m;

}

**public** **static** User getUser(User user) {

ServerConSql scs=**new** ServerConSql();

User u=scs.getUser(user.getId());

**return** u;

}

}

管理hashmap的类

ManagerClient.java

**public** **class** ManagerClient {

**private** **static** HashMap<String, Socket> *map* = **new** HashMap<String, Socket>();

**public** **static** **void** addSocket(User user,Socket s) {

*map*.put(user.getId(),s);

}

//通过id返回socket

**public** **static** Socket getSocket(String id) {

**return** *map*.get(id);

}

**public** **static** HashMap<String, Socket> getMap(){

**return** *map*;

}

**public** **static** **void** deleteSocket(String id) {

**if**(*map*.get(id) != **null**) {

*map*.remove(id);

}

**return**;

}

}

服务器与客户端连接的线程

**public** **class** ServerConClientThread **extends** Thread{

**private** ServerConSql scs=**new** ServerConSql();

**private** ObjectOutputStream oos;

**private** ObjectInputStream ois;

**private** Socket s;

**public** ServerConClientThread(Socket s,ObjectOutputStream oos,ObjectInputStream ois) {

**this**.s=s;

**this**.oos=oos;

**this**.ois=ois;

}

@Override

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

// **TODO** Auto-generated method stub

**while**(s!=**null**) {

**try** {

MsgUserDataBag msg=(MsgUserDataBag)ois.readObject();

//修改密码

**if**(msg.getMsg().getMsgType().equals("9")) {

System.***out***.println("jieshoudao类型9");

//

**if**(scs.updatePass(msg.getUser(), msg.getMsg())) {

Message m=**new** Message();

m.setMsgType("6");

MsgUserDataBag mudb=**new** MsgUserDataBag();

mudb.setMsg(m);

oos.writeObject(mudb);

}

}

**else** **if**(msg.getMsg().getMsgType().equals("8")) {//获取好友列表请求

System.***out***.println("接收到的好友列表请求");

MsgUserDataBag mudb=**new** MsgUserDataBag();

mudb=scs.getFriendList(msg);

oos.writeObject(mudb);

//方法MsgUserDataBag getFriendList(MsgUserDataBag mudb)

}**else** **if**(msg.getMsg().getMsgType().equals("3")) {

System.***out***.println(msg.getMsg().getMsgtext());

String id=msg.getMsg().getReceiver();

SendMsg smg=**new** SendMsg(ManagerClient.*getSocket*(id),msg);

smg.sendMsg(**new** ObjectOutputStream(ManagerClient.*getSocket*(id).getOutputStream()), msg);

}**else** **if**(msg.getMsg().getMsgType().equals("5")) {

System.***out***.println(msg.getMsg().getMsgtext());

String id=msg.getMsg().getReceiver();

SendMsg smg=**new** SendMsg(ManagerClient.*getSocket*(id),msg);

smg.sendMsg(**new** ObjectOutputStream(ManagerClient.*getSocket*(id).getOutputStream()), msg);

}**else** **if**(msg.getMsg().getMsgType().equals("11")) {

//添加好友成功

scs.addFriend(msg);

String id=msg.getMsg().getSender();

SendMsg smg=**new** SendMsg(ManagerClient.*getSocket*(id),msg);

smg.sendMsg(**new** ObjectOutputStream(ManagerClient.*getSocket*(id).getOutputStream()), msg);

}**else** **if**(msg.getMsg().getMsgType().equals("12")) {

//添加好友失败

String id=msg.getMsg().getSender();

SendMsg smg=**new** SendMsg(ManagerClient.*getSocket*(id),msg);

smg.sendMsg(**new** ObjectOutputStream(ManagerClient.*getSocket*(id).getOutputStream()), msg);

}

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (ClassNotFoundException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

}

客户端登录成功以后，新开辟一条线程用于保持与服务端的通信

客户端与服务器端连接的线程

**public** **class** ClientConServerThread **extends** Thread{

**private** List<User> mngfriend=**new** ArrayList<User>();

**private** **boolean** isRunning = **true**;

**private** MsgUserDataBag mudb=**new** MsgUserDataBag();

**private** Socket s;

**private** User user;

**private** Message msg;

**private** **boolean** isUpSu;

**private** ObjectOutputStream oos;

**private** ObjectInputStream ois;

**public** ClientConServerThread(Socket s,ObjectOutputStream oos,ObjectInputStream ois) {

// **TODO** Auto-generated constructor stub

**this**.setS(s);

**this**.oos=oos;

**this**.ois=ois;

}

**public** **void** receive(Socket s) {

**try** {

ois=**new** ObjectInputStream(s.getInputStream());

mudb=(MsgUserDataBag)ois.readObject();

**if**(mudb.getMsg().getMsgType().equals("3")) {

//普通聊天消息

ChatPanel cp=ManageChatPanel.*getChatUI*(mudb.getMsg().getSender());

**if**(cp==**null**) {

cp=**new** ChatPanel(**this**.getUser(),mudb.getMsg().getSender(),

mudb.getUser().getName(),s,oos,ois);

ManageChatPanel.*addChatUI*(cp);

}

**else** {

cp.setVisible(**true**);

}

UpdateChat uc=**new** UpdateChat(cp.getChatArea(),mudb.getMsg());

}**else** **if**(mudb.getMsg().getMsgType().equals("5")) {

//加好友

AddFriendUI afui=**new** AddFriendUI(getUser());

**if**(afui.getAgree().equals("2")) {//true同意添加好友，false拒绝

mudb.getMsg().setMsgType("11");

SendMsg smgadd=**new** SendMsg(s,mudb.getUser(),mudb.getMsg());

smgadd.sendMsg(oos);

}**else** **if**(afui.getAgree().equals("3")){

mudb.getMsg().setMsgType("12");

SendMsg smgadd=**new** SendMsg(s,mudb.getUser(),mudb.getMsg());

smgadd.sendMsg(oos);

}

}**else** **if**(mudb.getMsg().getMsgType().equals("6")) {

//修改密码成功true

setUpSu(**true**);

JOptionPane.*showMessageDialog*(**null**,"修改成功");

System.***out***.println("修改成功");

}**else** **if**(mudb.getMsg().getMsgType().equals("7")) {

//修改密码失败false

setUpSu(**false**);

JOptionPane.*showMessageDialog*(**null**,"修改失败");

}**else** **if**(mudb.getMsg().getMsgType().equals("8")) {

//返回好友列表

**this**.setMngfriend(mudb.getFriends().getFriendlist());

}**else** **if**(mudb.getMsg().getMsgType().equals("11")) {

JOptionPane.*showMessageDialog*(**null**, "添加好友成功");

}**else** **if**(mudb.getMsg().getMsgType().equals("12")) {

JOptionPane.*showMessageDialog*(**null**, "抱歉，对方拒绝添加您为好友");

}

}**catch**(IOException e) {

e.printStackTrace();

} **catch** (ClassNotFoundException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

@Override

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

// **TODO** Auto-generated method stub

**while**(isRunning) {

receive(getS());

}

}

**public** Socket getS() {

**return** s;

}

**public** **void** setS(Socket s) {

**this**.s = s;

}

**public** User getUser() {

**return** user;

}

**public** **void** setUser(User user) {

**this**.user = user;

}

**public** Message getMsg() {

**return** msg;

}

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

**this**.msg = msg;

}

**public** **boolean** isUpSu() {

**return** isUpSu;

}

**public** **void** setUpSu(**boolean** isUpSu) {

**this**.isUpSu = isUpSu;

}

**public** List<User> getMngfriend() {

**return** mngfriend;

}

**public** **void** setMngfriend(List<User> mngfriend) {

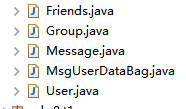
**this**.mngfriend = mngfriend;

}

}

客户端也需要管理好友聊天的面板，放入一个hashmap中

这是序列化的数据包，从客户端发送数据包给服务器，服务器判断数据包的Type，分类处理，注意：客户端与服务器的数据包序列号应该相同



ServerConSql.java是用来连接数据库及操作数据库的类