浙江大学计算机科学与技术学院

Java 程序设计课程报告

2018—2019 学年冬学期

题目	基于 Swing 的匿名即时通讯软件
学号	3160104734
学生姓名	沈子衿
所在专业	软件工程
所在班级	软件工程 1601 班

目　　录

1. 引言

本次，我们开发的是一款基于 Java Swing 图形库和 Java socket 的简单匿名即时通讯软件，其功能是将多个客户端连接到一个服务器上，利用 socket 技术实现多端同步和实时交流。

本项目分为两个模块：客户端和服务器端。其中，客户端基于 MVC 开发，简单分为 UI 实现和业务逻辑两个子模块。

1.1 设计目的

基于 Java Swing 图形库和 Java socket 的简单即时通讯软件的具体功能和设计思路如下：

(1) 用户登录客户端，客户端将自动把用户当前的 IP 地址作为账号，随必要信息上传到服务器，服务器返回一个已经登录的消息；

(2) 用户在文本框中输入要发送的文本，点击“send”，就可以将文本发送到聊天室；

(3) 与此同时，后台线程反复请求服务器，更新当前消息列表，一旦有新消息发出，则接收、处理并显示在图形界面上。

(4) 当关闭客户端时，向服务器发送一个 logout 信息，表示客户端同服务器断开连接；客户端安全退出，服务器端不退出；

(5) 使用 MySQL 数据库保存数据：当客户端热启动时，执行查询代码，从服务器端获取聊天记录；

1.2 设计说明

本程序采用 Java 程序设计语言，在 Jetbrains 基金会开发的 Intellij IDEA 平台下编辑、编译与调试。具体程序由我一人开发而成。

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
沈子衿	负责整个程序前期的需求分析和	报告的第 1 章、第 2 章和第 4 章

	整体功能的架构 程序后期的测试与运行	
沈子衿	负责程序中客户端模块的设计编码	报告的第 3 章
沈子衿	负责程序中服务器模块的设计编码	目录、总结和参考文献的整理 报告后期的格式设置

2. 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) 构建 SwingICQ 服务器端，实现信息的转发和存储；

(2) 构建 SwingICQ 客户端，实现信息的发送、接收和基本的 GUI 交互操作；

(3) 构建数据库连接，设计数据库，实现信息的静态存储；

程序的总体功能如图 1 所示：

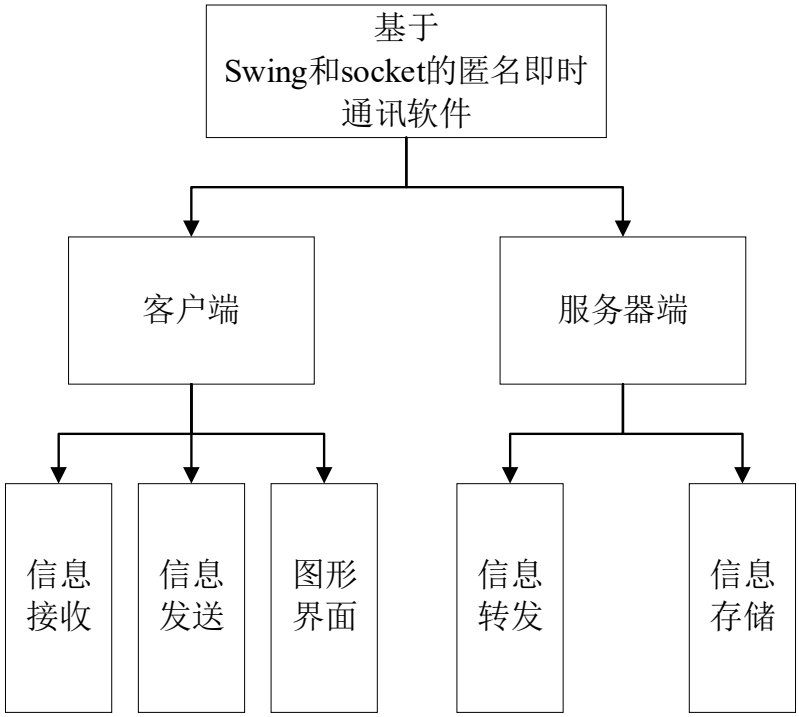


图 1 总体功能图

2.2 流程图设计

客户端消息发送总体逻辑如图 2 所示：

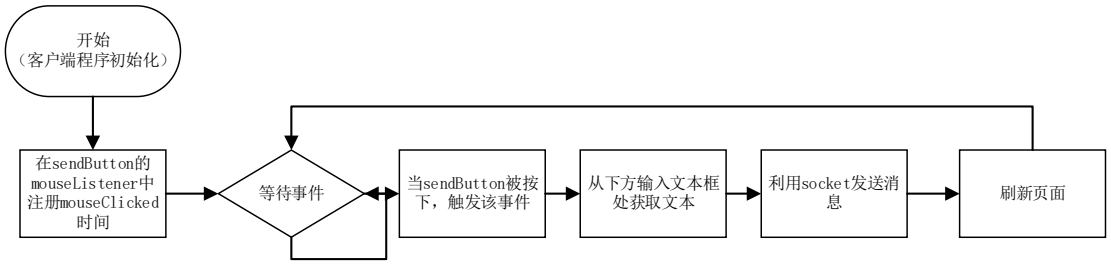


图 2 客户端消息发送总体流程图

客户端消息接收线程总体流程如图 3 所示：

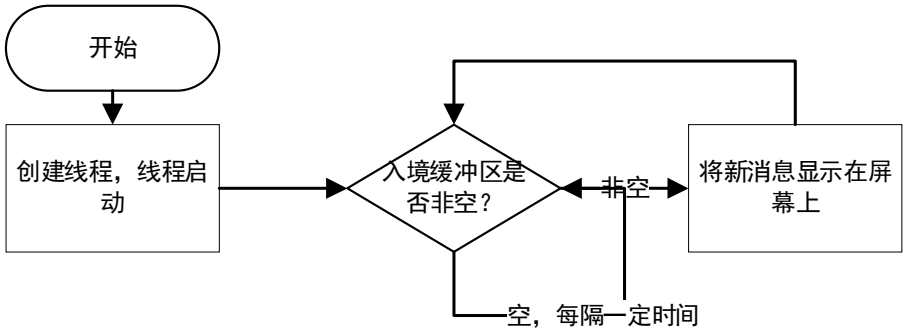


图 3 客户端消息接收总体流程图

服务端处理消息总体流程如图 4 所示：

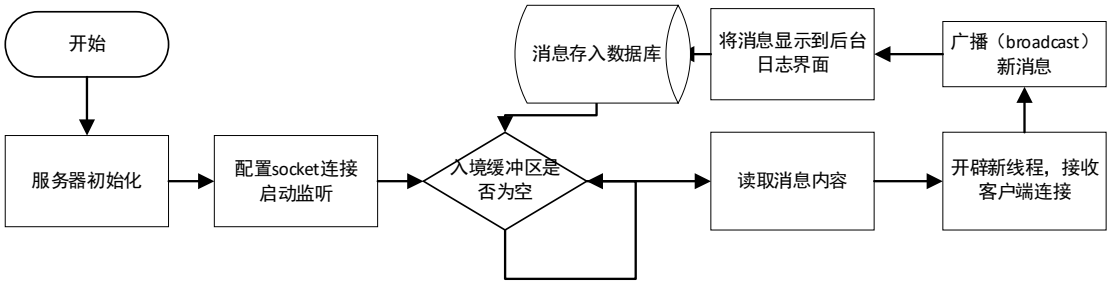


图 4 服务端处理消息总体流程图

服务端处理新连接总体流程如图 5 所示：

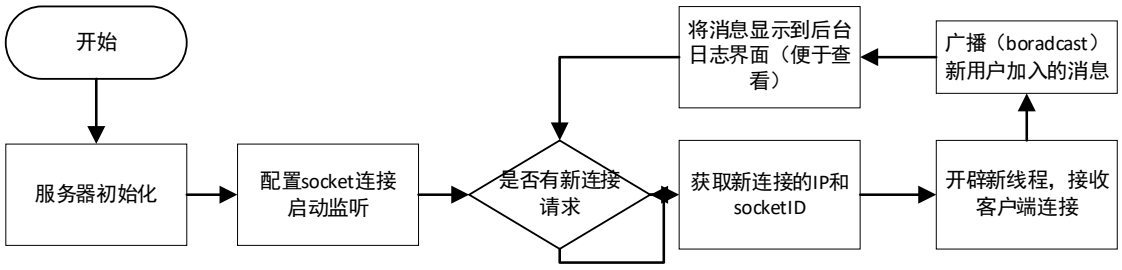


图 5 服务端处理新连接总体流程图

2.3 数据库设计

数据库地址：139.196.72.112（阿里云轻量级服务器）
数据库种类：MySQL
数据表结构：

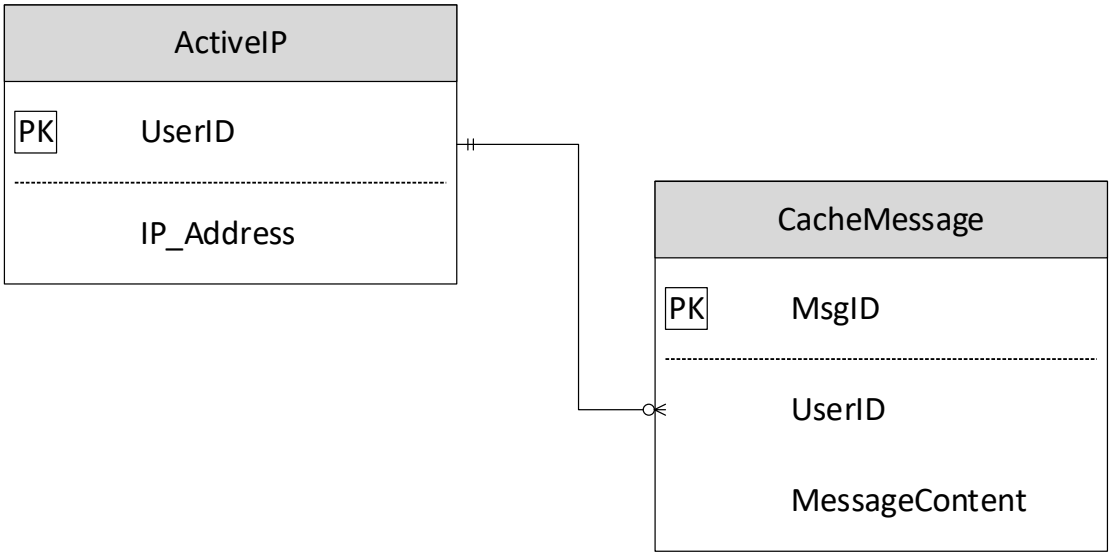


图 6 数据库关系图

3. 详细设计

3.1 客户端初始化

客户端初始化具体流程如图所示：

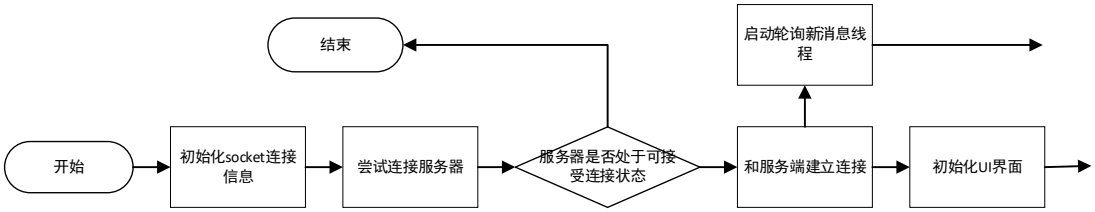


图 8 客户端初始化流程图

下面，我将对程序具体业务逻辑详细介绍如下：

首先，在客户端程序开始初始化的第一阶段，程序首先根据代码中设定的服务器 IP 地址对 socket 连接相关变量进行初始化：

```
1.  // 保存有关服务器连接的信息，这里使用的是本地的服务器
2.  public static final int PORT = 9999;
3.  public static final String serverAddr = "localhost";
4.  // 多线程实现
5.  // 客户端与服务端进行交流使用的 socket
6.  public static Socket socket = null;
7.  // 出境缓冲区，用于暂时保存需要发送给服务器的消息
8.  public static PrintWriter serverout = null;
9.  // 入境缓冲区，用于暂时保存需要从那个服务器读取的消息
10. public static BufferedReader serverin = null;
11. // 从服务器循环接收消息使用的子线程
12. public static Thread  rcvthread;
13. // 当前客户端是否处于正常运行状态
14. public static Boolean running;
```

客户端调用 srv_connect 函数和服务器建立 socket 连接。如果服务器未处于活跃状态或拒绝连接，则客户端返回一个错误提示，程序结束。否则，进入初始化的第二阶段：

```
1.  static void srv_connect(){
2.      try{
3.          // 初始化 socket 连接，入境和出境缓冲区
4.          socket = new Socket(serverAddr,PORT);
5.          serverout = new PrintWriter(socket.getOutputStream(), true);
```

```
            System.out.println(serverout);
            serverin = new BufferedReader(new InputStreamReader(socket.getInputS
    tream()));
            System.out.println(serverin);
        }
        catch (UnknownHostException e) {
            // 异常情况 1：找不到主机
            System.out.println("404: Host Not Found");
            System.exit(1);
        }
        catch  (IOException e) {
            // 异常情况 2：找到了主机但主机拒绝连接
            System.out.println("403: Forbidden (maybe server isn't active)");
            System.exit(1);
        }
        running = true;
    }
```

初始化的第二阶段，客户端初始化一个负责轮询新消息的线程，负责在实时更新客户端主界面上的消息列表，每次轮询，该线程都会执行一系列操作，包括检测 socket 连接是否还处于活跃状态，以及入境缓冲区是否非空。如果入境缓冲区非空，则直接将获得的数据显示在 UI 界面的 textArea1 上：

```
class GetInfoThread extends Thread{
    @Override
    public void run() {
        System.out.println("test");
        while(running)
        {
            // 每次轮询都检测 socket 连接是否还处于活跃状态
            System.out.println("socket is closed: " + socket.isClosed());
            // 如果 socket 连接关闭了
            if (socket.isClosed()) running = false;
            System.out.println("while the client is running: " + running);
            // 否则，尝试从入境缓冲区中读取数据
            rcv_data();
            try {
                // 每次尝试间隔 200 毫秒
                Thread.sleep(200);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
```

```
    }
}
// rcv_data 方法
// 从服务器 socket 连接中读取数据
private void rcv_data(){
    String line;
    try{
        // 尝试从入境缓冲区中读取字符串
        line = serverin.readLine();
        if(line!=null){
            // 如果读取到字符串
            System.out.println("rcv:"+line);
            // 将字符串写入 UI 界面的 textArea
            textArea1.append(line+"\n");
            textArea1.selectAll();
        }
    }
    // 异常处理
    catch (IOException e){
        System.out.println("Can't receive data, maybe you should check the I
nternet!");
        e.printStackTrace();
        System.exit(1);
    }
}
```

初始化的第三阶段，代码初始化 UI 各组件，并将 UI 界面展现出来，与此同时，启动消息轮询线程：

```
 static void InitUI(){
        JFrame frame = new JFrame("SwingICQ");
        ClientMainUI c = new ClientMainUI();
        frame.setContentPane(c.panel1);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.pack();
        c.startPolling();   // 启动消息轮询
        frame.setVisible(true);
    }
```

这样，客户端的初始化也就完成了，与此同时，在服务器端的日志界面可以看到一个新的连接进入的消息：

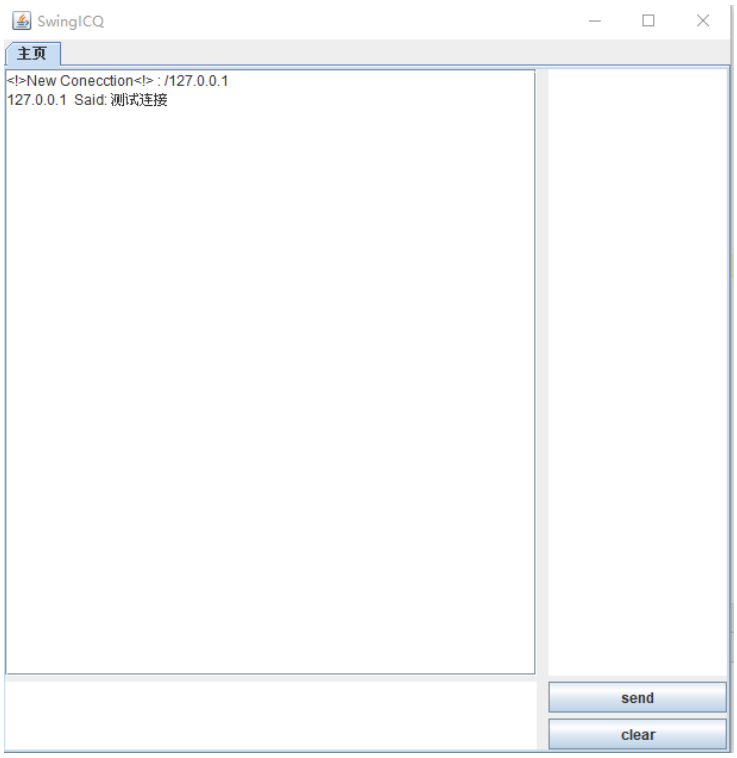


图 9 客户端初始化窗体

```
test
socket is closed: false
while the client is running: true
rcv:<!>New Conecction<!> : /127.0.0.1
socket is closed: false
while the client is running: true
测试连接
rcv:127.0.0.1  Said: 测试连接
socket is closed: false
while the client is running: true
```

图 10 服务器日志信息

3.2 客户端发送消息

客户端发送消息具体流程如图所示：

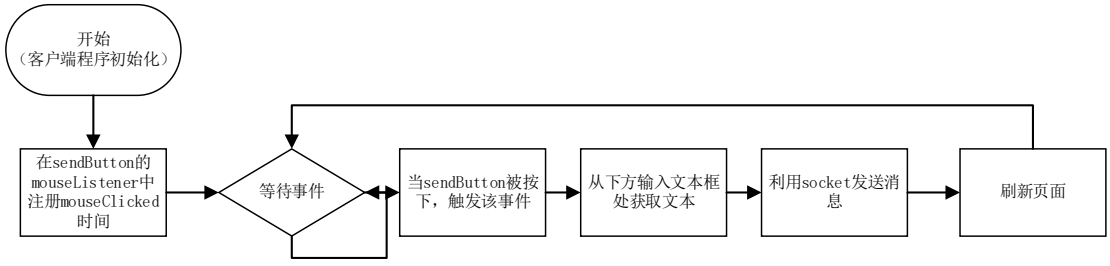


图 10 客户端发送消息流程

客户端发送消息涉及 Client 类和 ClientUI 类的互动，由绑定在 sendButton 上的 MouseAdapter 对象及其 mouseClicked 函数负责监听和实现:

```
发送 Button.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                super.mouseClicked(e);
                String t = editorPane1.getText();
                send_data(t);
                textArea1.selectAll();
                 // 清空输入区域
                editorPane1.setText("");
            }
        });
```

当按钮被按下时，mouseClicked 事件将被监听，该函数会从 editorPanel 中获取用户输入的文字，作为参数传给 send_data 函数：

```
// send_data 方法
    // 向往服务器发送消息的缓冲区中输入数据
    private void send_data(String text){
        try {
            System.out.println(text);
            serverout.println(text);
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
```

Send_data 会处理该字符串，将其序列化后写入出境缓冲区并发送给服务器。

具体效果如下：

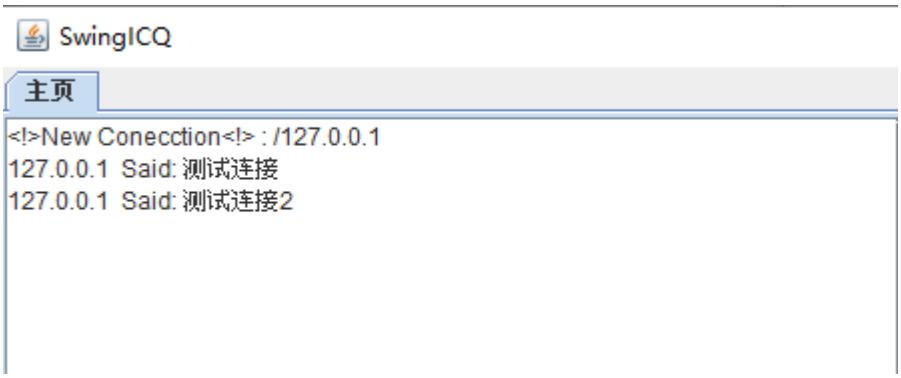


图 9 客户端发送消息示例

发现后台的消息轮询线程已经从服务器的广播信息中捕获到本客户端刚刚发出的消息：

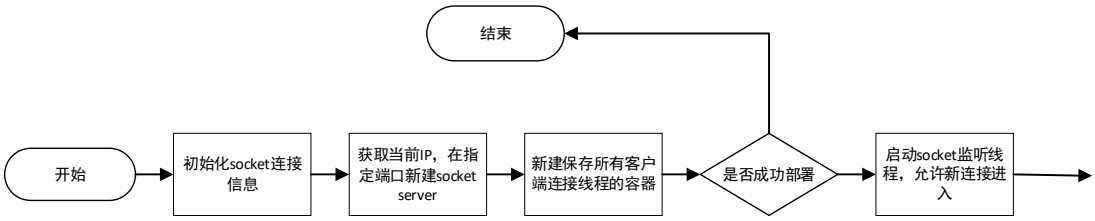
服务端后台：

```
测试连接2
rcv:127.0.0.1  Said: 测试连接2
socket is closed: false
while the client is running: true
```

说明消息成功发送给了服务器。多端同步的情况我们将在服务器部分进行测试。

3.3 服务器初始化

服务器端初始化代码逻辑如图所示：



服务器端初始化同样分为两个阶段。第一个阶段，服务器初始化需要绑定的 IP 地址和端口， 然后调用 ServerSocket 类构造函数在对应端口构造一个 Socket Server。如果这一过程没有出错，则服务器启动 socket 监听线程，允许多个远程客户端和服务器建立连接：

初始化 Socket Server：

```
1.  private void start_server() {
2.          try{
3.              // 获取 localhost 对应的 ip 地址，并依据这一 ip 地址初始化 socket server
```

```
4.              InetAddress ip = InetAddress.getByName(ADRESS);
5.              // 新建 socket server
6.              serversocket = new ServerSocket(PORT,0,ip);
7.              // 保存所有连接到服务器客户端线程的容器
8.              clients = new Vector<IndividualThread>();
9.          }
10.
11.         catch (UnknownHostException e) {
12.
13.         }
14.         catch (IOException e) {
15.             // 无法将 socket 服务部署到对应地址/端口后产生的异常
16.             System.out.println("Could not start conection on port "+PORT);
17.             System.exit(-1);
18.         }
19.         System.out.println("Server started on IP: "+serversocket.getInetAddr
    ess());
20.     }
```

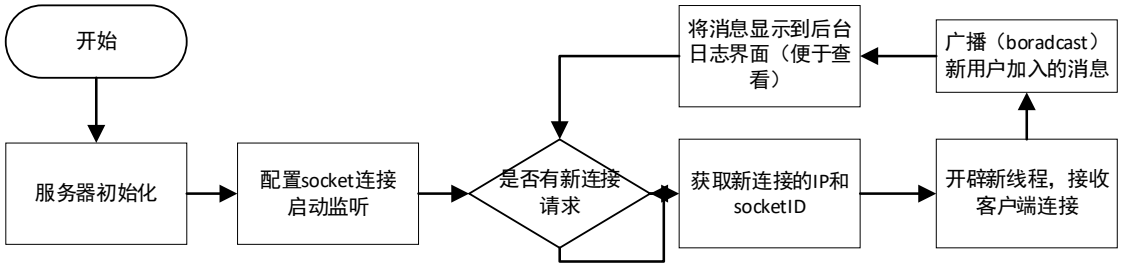
启动监听线程：

```
1.  public void listenSocket(){
2.
3.          try{
4.              while(true){
5.                  Thread.sleep(200);
6.                  // 启动监听线程
7.                  Socket new_conecction= serversocket.accept();
8.
9.                  IndividualThread w = new IndividualThread(new_conecction,thi
    s);
10.                 Thread t = new Thread(w);
11.                 t.start();
12.             }
13.         }
14.
15.         //2 Catch 1 para conection y otro para el sleep.
16.         catch (IOException e) {
17.             System.out.println("Accept failed: 4000");
18.             System.exit(-1);
19.         }
20.         catch (InterruptedException e) {
21.             e.printStackTrace();
22.         }
```

```
23.
24.     }
```

3.4 服务器接受客户端连接

服务器端接受客户端连接的代码逻辑如图所示：



在客户端试图同服务器建立新连接，也就是服务器的 accepted 函数监听到客户端的连接请求并且结束阻塞的时候：

```
1.  Socket new_conecction= serversocket.accept();
2.
3.                  IndividualThread w = new IndividualThread(new_conecction,thi
    s);
4.                  Thread t = new Thread(w);
5.                  t.start();
```

如上方代码所示，程序在同意建立连接后将会为该客户端新开一个单独的线程，专门处理同该客户端连接相关的一切事务。

注意 IndividualThread 对象的构造函数：

```
1.  public IndividualThread(Socket new_conecction, ServerFrame serverFrame) {
2.          client=new_conecction;
3.          running=true;
4.          this.serverFrame=serverFrame;
5.          this.serverFrame.registerClient(this);
6.
7.      }
```

该函数通过此线程对象保存的服务器对象引用（serverFrame）调用一个名为 registerClient 的函数：

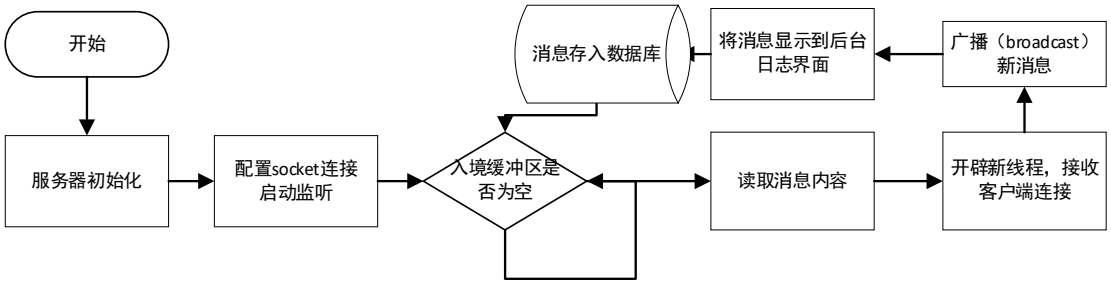
```
1.  public void registerClient(IndividualThread clientthread) {
```

```
2.          clients.add(clientthread);
3.      }
```

我们可以清楚地看出，该构造函数不仅创建了这样一个线程，而且负责将该线程放入 ServerFrame 对象的 client 列表（活跃线程列表）中，便于进一步管理。至此，针对一个单独客户端的管理线程创建完成。

3.5 服务器接收和转发客户端消息

服务器端接收和转发客户端消息的流程如图所示：



当某个客户端线程监听到来自对应客户端发过来的消息时，程序调用 receive 函数，试图从入境缓冲区中读出数据。当成功读出数据时，程序首先判断这是不是一个登出指令“/logout”，如果不是，则当作正常数据看待。对于正常数据，程序一方面会把日志打印到后台，另一方面会会调用 broadcast 广播函数对客户端线程列表中的所有客户端进行广播：

```
1.  public void recieve(){
2.      String line,parseline;
3.      while(running){
4.          try{
5.              Thread.sleep(100);
6.              line = in.readLine();
7.              if(line!= null){
8.                  if(check_logout(line))logout(); //如果收到的是 logout 信息，则登
    出
9.                  else{
10.                     parseline=client.getInetAddress()+"  Said: "+line;
11.                     parseline=parseline.substring(1);
12.                     System.out.println(parseline);
13.                     serverFrame.broadcast(parseline);
```

```
                }
            }
        }
        //CATCHS
        catch (IOException e) {
            System.out.println("IOERROR INDV THREAD.");
            e.printStackTrace();
            System.exit(-1);
        }
        catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

观察 broadcast 函数：

```
/**向所有客户端发送更新信息**/
    public void broadcast(String parseline) {

        String sendline = parseline;

        Iterator<IndividualThread> it;
        it = clients.iterator();
        int i = 0;
        while(it.hasNext()){
            IndividualThread temp_client = it.next();
            temp_client.send(sendline);

        }
    }
```

该函数接收一个类型为 String 的信息，对于客户端线程数组中的所有线程，程序依次调用各线程中的 send 函数，将更新的信息发送给所有处于活跃连接状态的客户端，达到同步的目的。

此外，服务端还将数据存入 MySQL 数据库，以便备份和恢复。

4. 测试与运行

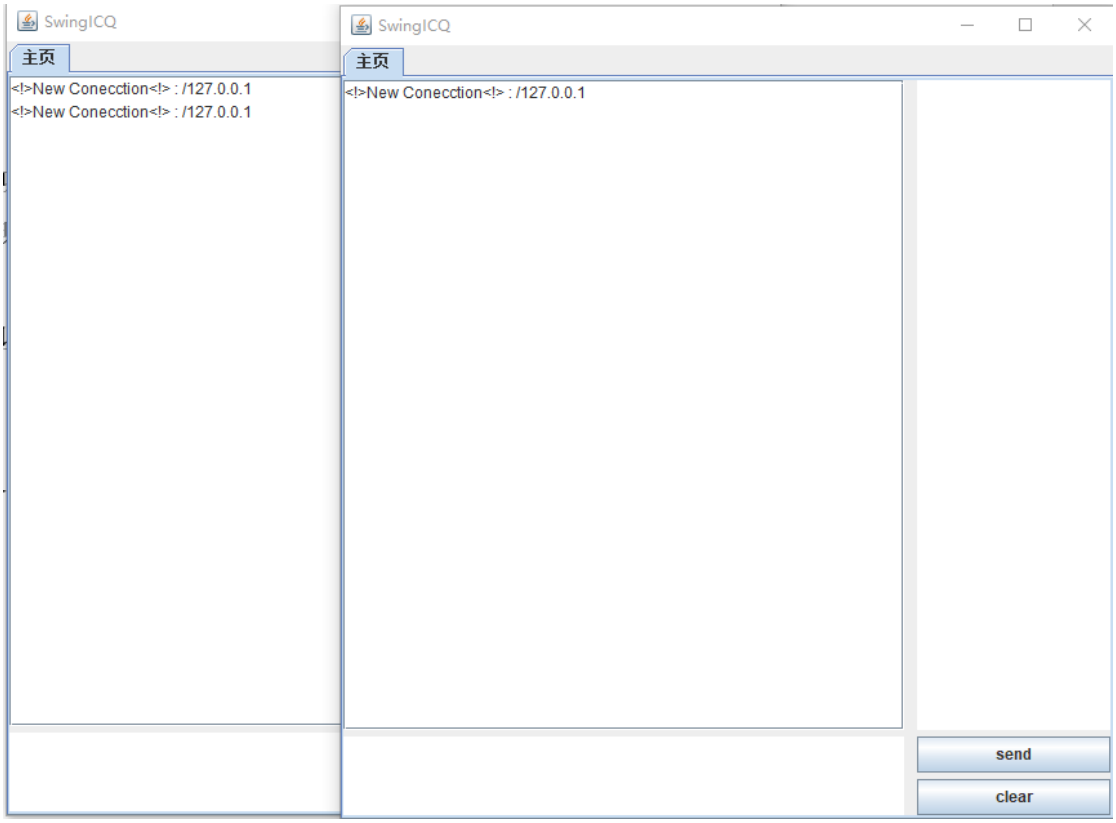
4.1 程序测试

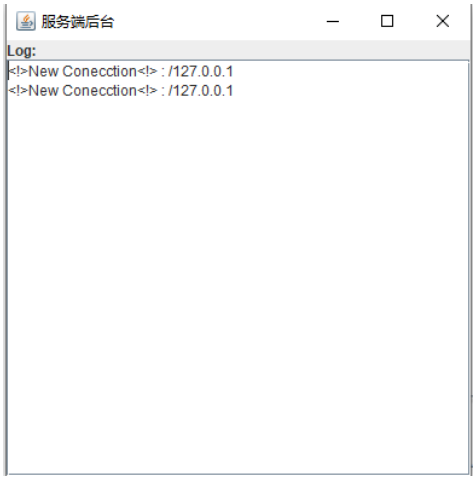
在程序的 1.0 版本编写完成后，我对程序的基本功能进行测试，发现其已经满足了作为匿名聊天室的基本要求，基本功能也都实现完全。唯一美中不足的是，服务端以及部分涉及资源共享和互斥的代码存在一定的安全风险，好在这一瑕疵不影响程序的总体性能。预计在未来的 1.1 版本中可以作进一步修正。

4.2 程序运行

【注：为了清楚地展现整个程序运行的过程，我为服务端撰写了一个简单的图形化日志界面】

服务端（后台日志界面）+两个客户端运行：

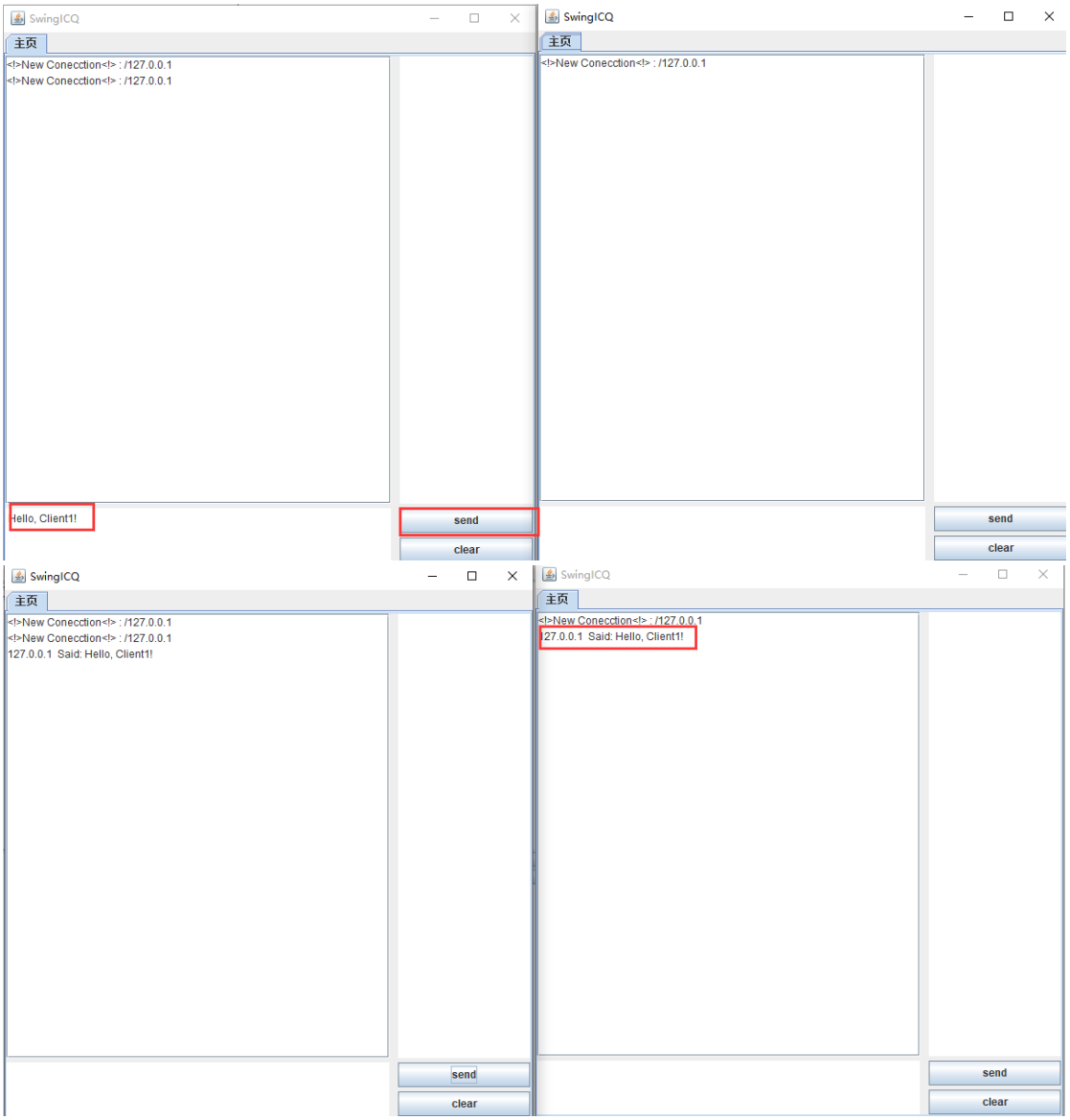




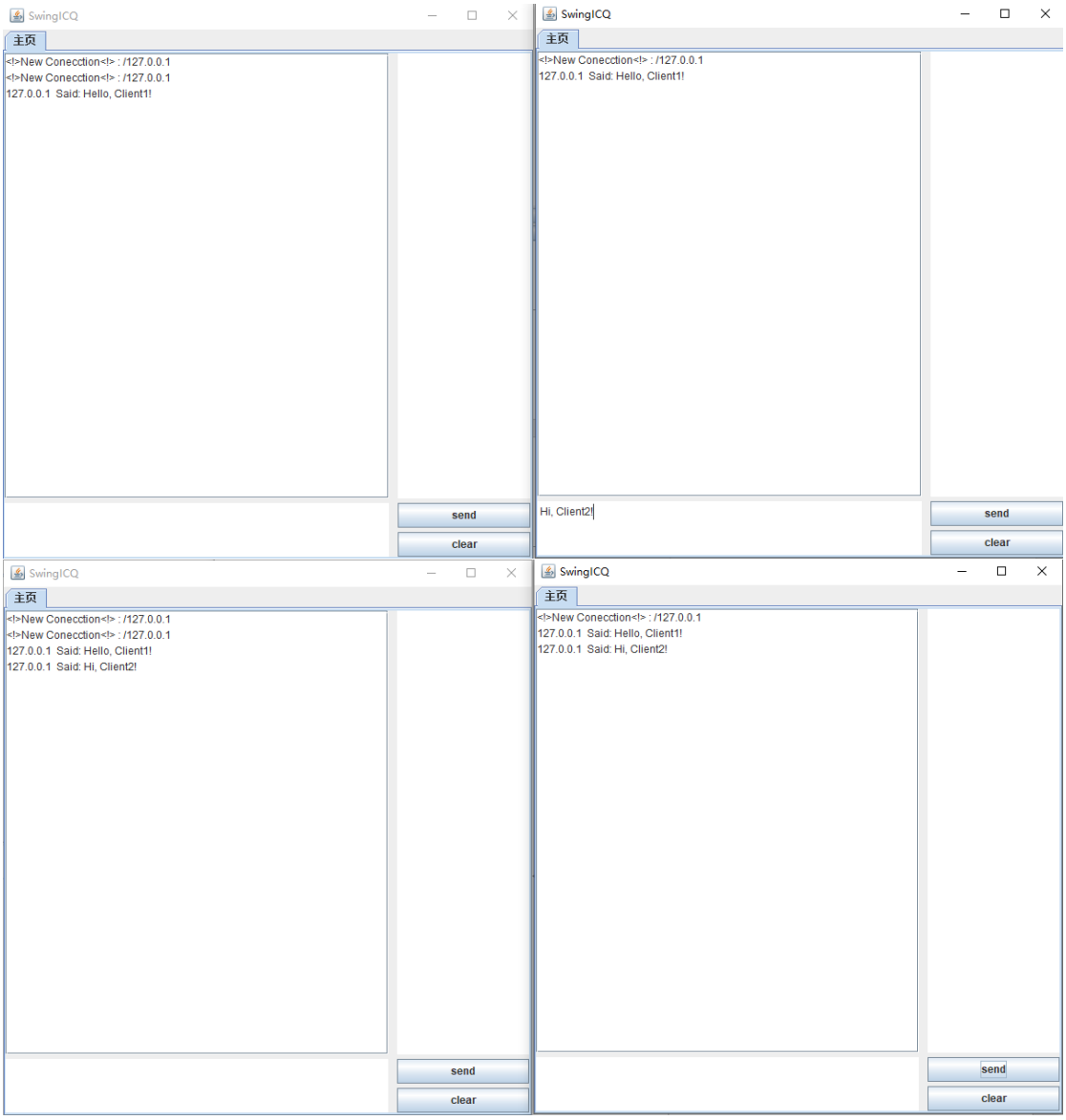
可以看到，先启动的客户端中出现了后启动客户端同服务器连接的信息。

客户端之间互相发送消息：

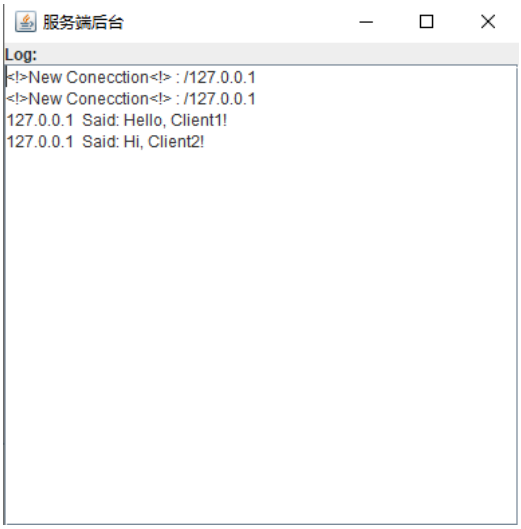
测试 1：



测试 2：



期间服务器端日志打印的信息：



5. 总结

本次作业内容丰富，工作量较大，我系统学习了使用 Swing 进行跨平台 GUI 程序编写和使用 Java Socket 进行服务器和客户端动态同步的方法，复习了计算机网络关于传输层的相关论述，巩固了 Java 应用技术课程中学习的基础知识，受益良多。

本次作业产品还有非常广阔的优化空间，在未来的学习和工作中我会充分吸取其中的经验和教训。

参考文献

[1] 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．