基于 Swing 和 Socket 的在线聊天程序

学号：2214010220
姓名：朱庆刚
指导教师：吕嘉

目录

1 分析和设计

1.1 需求分析

项目需求为实现一个基于 Swing 和 Socket 的在线聊天程序，大体需求如下：

1. 基于 Swing 的用户图形界面
2. 支持多个用户在线聊天
3. 支持两个用户点对点聊天
4. 聊天内容支持文字、图片、文件
5. 支持聊天记录的存储与回放

分析大体需求，将需求细化如下：

1. 基于即时聊天的需求，服务器与客户端建立长连接，并以此实现客户端向服务器发送消息以及服务器向客户端推送事件。
2. 基于发送图片和文件的非文本信息传输需求，模仿 HTTP 实现客户端到服务器的短连接请求，向服务器上传和下载文件。
3. 支持两种类型的聊天会话：所有在线用户的群聊，和两个在线用户之间的一对一私聊。
4. 在客户端实现聊天记录的保存与读取。
5. 实现客户端用户界面，使得操作相对简单友好。

1.2 总体设计思路

基于需求分析，系统分为三个主要模块：

1. 服务器（Server）：中心化的 Socket 服务器，负责保持与所有客户端的连接，接收客户端的消息发送，以及向相应客户端推送事件。同时，基于“临时通信协议”，接收并保存客户端上传的图片和文件，并提供下载服务。
2. 客户端（Client）：Socket 客户端，实现与服务器的通信协议，暴露向服务器发送消息的接口，接收服务器推送的事件并提供事件监听器。
3. 客户端 UI（UI）：基于 Swing 的用户图形界面，向用户提供易用的 UI，以图形化的方式操作客户端对通信协议的实现。

从基于模型-视图结构的视角来看，UI 模块将 Client 作为模型，实现视图，以事件驱动的方式（既有来自于服务器的事件，也有来自于用户界面的事件）操作模型聊天功能。

1.3 主要功能模块划分

1.3.1 服务器 Server

1. Server：服务器主类
   - 提供服务器程序入口
   - 运行 Socket 服务器，建立与客户端程序的连接
   - 对于每个建立的客户端连接，运行 Socket Handler 线程专门负责此客户端的消息接收
   - 对于临时连接请求，将连接控制权移交给 TCHandler 模块
2. Socket Handler：Socket 连接处理器
   - 提供一个线程供 Server 模块运行
   - 负责保持与客户端的长连接
   - 接收客户端发送的消息，触发处理函数
   - 实现处理函数，调用 Client 模块实现业务动作
3. Client：客户端业务类
   - 包装客户端 Socket 连接的输出流
   - 维护当前所有在线客户端的用户信息、包装输出流以及连接状态

- 依据业务动作向相应客户端推送事件
- 定时检查客户端连接状态，关闭连接中断的客户端连接

4. TCHandler：临时连接（Temporary Connection）处理器
   - 提供临时连接处理线程供 Server 模块运行
   - 实现临时通信协议
   - 基于“临时通信协议”接收用户文件上传并在服务器文件系统上保存
   - 基于“临时通信协议”响应用户文件下载请求，向客户端发送文件

1.3.2 客户端 Client

1. Client：客户端主类
   - 建立与服务器的 Socket 连接
   - 包装并暴露向服务器发送消息的协议实现，供 UI 调用
   - 建立连接后运行 Socket Handler 模块的线程
2. Socket Handler：Socket 连接处理器
   - 提供一个线程供 Client 模块运行
   - 负责保持与服务器的长连接
   - 接收服务器推送的事件，触发 Event Listener 中的事件处理器
3. Event Listener：提供事件处理器的一系列接口
   - 暴露事件侦听器接口（Interface）供 UI 实现
   - Socket Handler 接收到服务器的事件推送后，将调用对本接口的实现
4. TCRequest：临时连接（Temporary Connection）请求模块
   - 实现临时通信协议
   - 基于“临时通信协议”将本地文件上传到服务器
   - 基于“临时通信协议”从服务器下载文件并缓存/保存到本地

1.3.3 客户端图形界面 UI

1. Main：客户端图形界面主类
   - 提供图形化客户端程序入口
   - 实例化 Client，提供客户端协议实现
   - 中心化维护客户端全局状态
   - 实现登陆逻辑，登陆完成后启动 Main Frame
   - 在程序结束前关闭连接
2. Event Handler：事件处理器
   - 实现 Client 提供的 Event Listener 接口，在服务器发送事件时触发进行处理
   - 收到服务器聊天消息后调用 Message Record 模块对聊天记录进行持久化
   - 收到服务器事件时更新相应窗口视图，向用户展示
3. Message Record：消息记录
   - 用类表示消息记录对象
   - 实现消息记录对象的序列化及反序列化
   - 实现序列化消息记录写入磁盘，以及持久化消息记录的读取
4. Chat Panel：聊天面板的封装
   - 封装群聊以及私聊中，相同的聊天逻辑
   - 提供一个面板组件供 Main Frame 和 Direct Message Frame 使用
   - 消息框中展示消息记录
   - 实现文本、图片、文件消息的发送
   - 实现消息展示逻辑，包括展示图片消息前需要缓存图片
   - 实现内嵌文件下载功能，下载其他用户发送的文件消息

图 1　Chat Panel 面板界面

5. Direct Message Frame：私聊窗口
   - 视图上，直接将 Chat Panel 用 JFrame 包装
   - 逻辑上，指定 Chat Panel 发送消息的目标为对应欲私聊用户

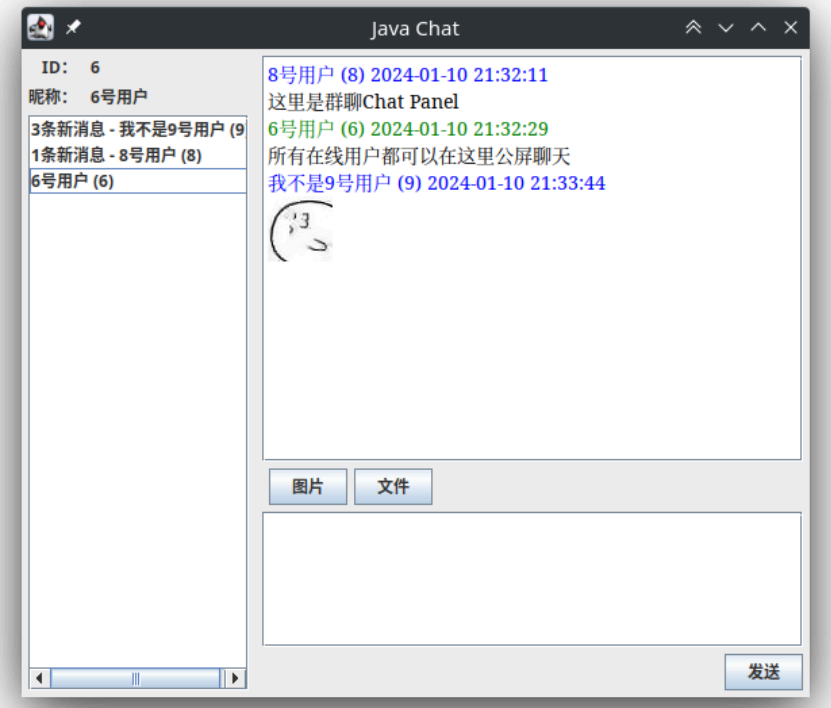

图 2　Main Frame 窗口界面

6. Main Frame：主窗口
   - 引入 Chat Panel，实现群聊功能

- 左侧顶部展示当前登陆用户的 id 和昵称
- 左侧列表展示在线用户列表以及未读消息数量（如有）
- 双击左侧列表列表项可以打开私聊窗口
- 特别地，允许和自己私聊

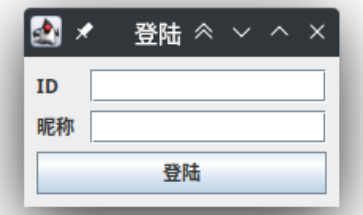

图 3　Login Frame 窗口界面

7. Login Frame：登陆窗口
   - 提供登陆功能
   - 点击登陆，使用 Client 发送登陆请求，关闭自身，调用 Main 进行下一步操作

1.4 主要连接（长连接）协议设计

1.4.1 数据包结构约定

字节数	类型	说明
1	uint8	数据包类型
		数据包内容

1.4.2 聊天消息类型约定

编号	内容	说明
0	文本消息	文本消息
1	图片 md5	图片消息
2	md5+文件名	文件消息

注：md5 长度固定，为 32 字节

1.4.3 对象

1. 用户 User

字节数	类型	说明
2	uint16	用户 id
1	uint8	用户名字节数
	string	用户名

2. 消息 Message

字节数	类型	说明
2	uint16	发送者 id
4	uint32	unix 时间戳（秒）
1	uint8	消息类型
2	uint16	消息内容字节数
	string	消息内容

1.4.4 客户端发往服务器的数据包

标题开头的整数表示数据包类型，此处将只列出数据包内容。数据包将使用 1.4.1 中描述的结构组合起来。

0 登陆

字节数	类型	说明
2	uint16	用户 id
1	uint8	用户名字节数
	string	用户名

1 发送群消息

字节数	类型	说明
1	uint8	消息类型
2	uint16	消息内容字节数
	string	消息内容

2 发送私聊消息

字节数	类型	说明
2	uint16	目标用户 id
1	uint8	消息类型
2	uint16	消息内容字节数
	string	消息内容

127 客户端下线

1.4.5 服务器发往客户端的数据包

0 登陆失败/强制下线

发送完该数据包后，服务器将立即断开与客户端的连接

字节数	类型	说明
1	uint8	错误信息长度
	string	错误信息内容

1 登陆成功

登陆成功后，服务器将发送当前在线的用户列表

字节数	类型	说明
2	uint16	当前在线的用户数量
	User	用户对象

2 用户上线

有用户上线时发送给全体客户端

字节数	类型	说明
	User	用户对象

3 用户离线

有用户离线时发送给全体客户端

字节数	类型	说明
2	uint16	用户 id

4 群消息

群消息将发送给全体客户端

字节数	类型	说明
	Message	消息对象

5 私聊消息

私聊消息将只发送给私聊对象

字节数	类型	说明
	Message	消息对象

1.5 临时连接（短连接）协议设计

模仿 HTTP 工作方式，由客户端发送请求，服务器接受请求并发送完结果之后，连接将被立即关闭。

1.5.1 上传

Request

字节数	类型	说明
1	Uint8	数据包类型，值为 128
32	string	文件 md5
4	uint32	文件字节数
	bytes	文件内容

Response

字节数	类型	说明
1	uint8	数据包类型，值为 128
1	uint8	错误信息字节数，0 表示成功
	string	错误信息

1.5.2 下载

Request

字节数	类型	说明
1	Uint8	数据包类型，值为 129
32	string	文件 md5

成功 Response

字节数	类型	说明
1	uint8	数据包类型，值为 129
1	uint8	值为 0
4	uint32	文件字节数
	bytes	文件内容

失败 Response

字节数	类型	说明
1	uint8	数据包类型，值为 129
1	uint8	错误信息字节数
	string	错误信息

1.6 系统过程分析与设计

1.6.1 服务器 Server

主类 Server 主线程

1. 程序从 Server 主类的 main 函数开始执行
2. 打开线程池和 Socket 服务器
3. 开始服务循环，接受客户端连接
4. 每当有客户端连接，创建新的 Socket Handler，并将其线程提交进线程池启动线程

Socket Handler 连接处理线程

1. 启动 guard 线程，如果在一段时间之后当前连接仍未进入已登陆装态或是交由 Temporary Connection Handler 处理，则发送超时错误信息并直接关闭连接
2. 从输入流读取包类型。如果是临时连接协议的包则启动 Temporary Connection Handler 处理当前连接，否则处理登陆包
3. 查询 Client 模块中是否已经存在当前登陆 id 的实例，如果存在则将之前的实例下线并销毁。（下线实例会通过 Client 的静态方法向全体 Client 实例广播下线数据包）
4. 登陆成功，打断 guard 线程
5. 创建 Client 实例，注册到 Client 模块中，并通过 Client 的静态方法向全体 Client 实例广播上线数据包
6. 开始 IO 循环，每次读取数据包先判断包类型，再依据包类型，通过 Client 类的静态方法广播数据包或者在 Client 模块中查询目标 Client 实例并向这个实例的连接发送数据包

Client 客户端镜像模块

- 设置静态属性存储当前在线所有在线连接的 Client 实例，提供查询和操作接口
- 使用静态语句块启动 Check Connection 线程，每隔一段时间遍历全部 Client 实例，检查其是否数据包发送超时，并强制下线超时的 Client 实例对应的客户端
- 提供静态方法实现与特定 Client 实例无关的操作，如广播数据包
- 对于 Client 实例，提供成员方法向其对应的客户端发送数据包

TCHandler 临时连接处理模块 Event Handler： 事件处理器

- 提供两个静态函数，上传和下载，在 Socket Handler 识别到临时连接数据包时接过连接的控制权（但实际上还是运行在 Socket Handler 的线程中）
- 对于上传文件请求，接受客户端发送的文件并以文件 MD5 为关键字存储在文件系统中
- 对于下载文件请求，从文件系统中读取对应 MD5 的文件并通过 Socket 发送
- 以上过程处理完毕后，关闭 Socket 连接以及整个线程，释放线程池资源

1.6.2 客户端 Client

主类 Client

1. 构造函数接收用户 id 和用户名参数，构造 Client 实例
2. 打开 Socket 连接连接服务器并发送登陆数据包，成功登陆
   - 登陆成功不代表开始运行，详见下一条
3. 外部程序（UI）注册事件处理接口（Event Listenre）后，主类将启动 Socket Handler 线程，将事件处理接口传递给 Socket Handler

- 本类暴露成员方法，操作 Client 实例以当前登陆用户的身份向服务器发送消息，运行在调用者（UI）的线程中

Socket Handler 连接处理线程

1. 构造函数接收 Event Listenre 接口的实例
2. 线程开始运行，启动 IO 循环，接收服务器发来的数据包

2. 根据数据包，调用对应的事件处理函数

TCRequest 临时连接请求模块

- 提供两个静态函数，上传和下载，运行在调用者（UI）的线程中
- 对于上传文件请求，读取磁盘上的文件，计算 MD5 值，并连接服务器发送文件
- 对于下载文件请求，通过 MD5 请求服务器，将文件下载到指定位置

1.6.3 客户端图形界面 UI

Main 客户端图形界面主类

1. 主函数启动，打开登陆窗口，线程控制权交给 Login Frame
2. 用户输入 id 和用户名，登陆窗口将其传递给 Main 主类并销毁自身，线程控制权交给 Main 主类
3. 创建客户端 Client 实例，进行登陆并注册 Event Handler 事件处理器
4. 打开主窗口 Main Frame，控制权交给 Main Frame

- Main 主类还负责存储全局状态，如当前打开了哪些私聊窗口，每个用户有多少未读消息，存储在静态属性中

Event Handler 事件处理器

- 登陆成功、用户上下线事件将触发主窗口用户列表视图更新
- 接收到群聊消息或私聊消息
  1. 调用 Message Record 模块将消息持久化，写入消息记录文件
  2. 如果是群消息，调用主窗口上群聊 Chat Panel 显示消息
  3. 如果是私聊消息，先在 Main 中的全局状态中查询发送者的私聊窗口是否打开，若打开则直接将消息显示在窗口的 Chat Panel 上，否则将未读消息数+1

Main Frame 主窗口

- 主窗口构造时，创建群聊 Chat Panel
- 双击左侧的用户列表中的列表项将打开私聊窗口，并更新主类 Main 中的全局状态

Direct Message Frame 私聊窗口

- 私聊窗口构造时，创建私聊 Chat Panel，并更新主类 Main 中的全局状态
- 窗口关闭时也要更新全局状态

Chat Panel

- 当需要显示图片消息时，调用 TCRequest 模块从服务器下载图片，然后再显示出来
- 当点击聊天框中内嵌的文件下载链接时，先打开文件选择窗口，选择想要下载到的目录，然后调用 TCRequest 模块从服务器下载文件保存到指定位置
- 点击发送按钮，调用 Client 实例的方法向服务器发送文本消息
- 点击图片/文本按钮，首先打开文件选择窗口选择要上传的文件，然后调用 TCRequest 模块向服务器上传文件，最后再调用 Client 实例的方法向服务器发送消息

2 系统实现

本章将介绍各个主要模块的关键功能具体实现。

2.1 通用关键模块实现

2.1.1 PacketBuffer 数据包缓冲区

本类是对 ByteArrayOutputStream 的包装，用于构造数据包。以下是各个 public 方法的介绍。

`new PacketBuffer()` 构造函数

`void write(int x)` 写入 8 位无符号整数

`void writeUint16(int x)` 写入 16 位无符号整数

`void writeUint32(long x)` 写入 32 位无符号整数

`void writeBytes(byte[] bytes)` 写入字节数组的全部字节

`void writeTo(OutputStream out) throws IOException` 将之前写入自身的所有字节数据写入到输出流中，用于发送构造完成的数据包

2.2 Server 服务器关键功能实现

2.2.1 线程池

由于对于每一个连接的客户端都需要一个线程为其提供服务，于是使用线程池提高资源利用效率。这里使用定长线程池，限制总线程并发数，线程一旦结束立即回收。

```
ExecutorService threadPool = Executors.newFixedThreadPool(64);
threadPool.submit(/*Runnable*/);
```

2.2.2 guard 线程

用户 Socket 连接后如果在规定时间内没有登陆将被立即关闭，释放线程资源。

具体实现方法为，在接收登陆数据包之前启动 guard 线程。guard 线程先睡眠一段时间，然后检查是否已经登陆，如果未登录则说明登陆超时，直接关闭 Socket 即可。

关闭 Socket 将导致阻塞在读取 Socket 输入流的 Socket Handler 线程立刻抛出 IOException 异常从而结束，释放线程资源。

如果该线程是临时连接，只要打断 guard 线程，guard 线程内部的 Thread.sleep 函数将抛出 InterruptedException，这时直接结束 guard 即可。

```
// 开一个 guard 线程，LOGIN_TIMEOUT 时间之后判断是否登陆成功，如果登陆超时直接踢下线
Thread guard = new Thread(() -> {
  // 先睡眠 LOGIN_TIMEOUT 时间
  try {
    Thread.sleep(LOGIN_TIMEOUT);
  } catch (InterruptedException e) {
    // 打断说明遇到了临时连接包，可以下班了
    return;
  }
  if (!isLogin) { // 判断是否已经登陆
    // 未登录，直接踢下线
    close("连接超时");
  }
});
```

```
guard.start(); // 启动 guard 线程

// 如果该线程是临时连接，或者登陆已经完成
isLogin = true; // 设置为已经登陆，防止 guard 关闭连接
guard.interrupt(); // 打断 guard，使得 guard 退出
```

2.2.3 读取数据包

使用 DataInputStream 即可按照协议读出数据包的各个字段，下面以群聊消息数据包为例：

```
private void groupMsg() throws IOException {
  int msgType = input.readUnsignedByte(); // 消息类型
  int msgLen = input.readUnsignedShort(); // 消息长度
  byte[] msgBytes = new byte[msgLen]; // 根据长度创建 byte 数组
  input.readFully(msgBytes); // 读取消息内容字节

  client.sendGroupMsg(msgType, msgBytes);
}
```

2.2.4 检查连接超时

Client 客户端模块每隔一段时间都会对所有已连接客户端进行检查，如果发现有连接阻塞在接收数据包中间达到一定时间，则直接关闭 Socket 连接。

关闭 Socket 将导致阻塞在读取 Socket 输入流的 Socket Handler 线程立刻抛出 IOException 异常从而结束，释放线程资源。

这里使用了 java 的 Timer 工具。其实质是一个线程定时执行某个函数。

```
private static void checkConn() {
  for (Client client : clients.values()) {
    if (client.status.isTimeout()) { // 这个客户端超时了
      client.close("数据包超时");
    }
  }
}
static { // 从静态代码块启动
  Timer timer = new Timer(true); // timer 作为后台线程
  timer.schedule(new TimerTask() {
    public void run() {
      checkConn();
    }
  }, 0, CONN_CHECK_PERIOD); // 每隔 CONN_CHECK_PERIOD 执行一次
}
```

在 TCHandler 模块中，检测文件上传超时中也使用了类似的机制，只不过检查对象从所有连接变成仅仅检查当前上传连接，这里不再赘述。

2.2.5 向客户端发送数据包

服务器中的多个线程可能会对同一个连接同时发送数据包，造成数据包错误。为了避免这种情况发生，应该给输出流对象加锁，在所有发送数据包的操作前必须等待锁释放。比如：

```
synchronized (output) { // 等待并获得输出流的对象锁
  output.write(PACKET_LOGIN_SUCCESS);
  getUserList().writeTo(output);
}
```

2.3 Client 客户端关键功能实现

因为 Client 和 Server 相似，都是基于 Socket 编程，重要的部分有很大重合，因此在前文中已经讲述过的内容不再重复。

2.3.1 EventListener 接口

模仿 Swing 中 EventListener 的实现方式，client 提供 EventListener 接口用于辅助实现事件处理函数。

```
public interface EventListener {
  /**
    * 收到私聊消息
    * @param userId 发送者 id
    * @param time 时间戳 (s)
    * @param msgType 消息类型
    * @param msg 消息内容
    */
  void dmMsg(int userId, long time, int msgType, String msg);

  // 仅举一例，其他函数声明在此省略
}
```

2.3.2 文件 MD5 计算

使用 java 的 MessageDigest 库实现 MD5 字节的计算。为了将字节转换为字符串，这里使用了 Bigint 类，将字节数组看作大整数的表示转换为 16 进制字符串。

```
MessageDigest md5Digest = null;
try {
    md5Digest = MessageDigest.getInstance("MD5");
} catch (Exception ignored) {} // MD5 肯定存在啊
// 文件输入流
DigestInputStream din = new DigestInputStream(new FileInputStream(file), md5Digest);
// 循环将文件全部过一遍，计算 md5 同时统计文件总长度
byte[] fileBuf = new byte[1024];
int curLen;
while ((curLen = din.read(fileBuf)) != -1) totLen += curLen;
assert md5Digest != null;
// md5Digest.digest()得到 md5 字节，利用 Bigint 类将其转换为 16 进制字符串字节
byte[] md5 = new BigInteger(1, md5Digest.digest()).toString(16).getBytes();
```

2.4 UI 客户端图形界面关键功能实现

2.4.1 退出程序前关闭客户端连接

使用 Java Runtime 的钩子，在程序退出前执行函数

```
Runtime.getRuntime().addShutdownHook(new Thread(() -> {
  if (client == null) return;
  try {
    // 关闭客户端
    client.close();
    // 关闭消息记录文件
    groupRecord.close();
  } catch (IOException ignored) {}
}));
```

2.4.2 消息记录的持久化

Java 自带的序列化带有头信息，因此在写入聊天记录文件时不能在已经写入的部分尾部追加。为了能够直接将新消息记录追加在文件结尾，笔者自行实现了一套类似于数据包协议的序列化方法用于持久化消息记录。具体操作方法和 Socket 编程一样，只是把写入目标从 Socket 输出流变成文件输出流，Socket 输入流变成文件输入流。具体操作方法前面已经讲述过，这里不再重复。

2.4.3 支持富文本的消息框

Swing 有一个支持富文本的编辑框组件 JTextPane。本项目中采用笔者最熟悉的 HTML 方式操作富文本。

```
JTextPane msgPane;
// 设置消息框为 html 格式，不可编辑
msgPane.setContentType("text/html");
msgPane.setEditable(false);

// 向 TextPane 尾部中加入 html 标签
HTMLDocument doc = (HTMLDocument)msgPane.getStyledDocument();
doc.insertAfterEnd(doc.getCharacterElement(doc.getLength()), htmlTag);
```

这里的 htmlTag 支持文字图片链接和 css，能够满足本项目的需求，如：

```
<span>这是一段文本</span>
<span style="color: blue;">这是一段蓝色的文本</span>
<img src="file:path/to/img" alt="这是一张图片" />
<a href="http://some_information_here">这是一个链接，点击可以触发事件</a>
```

2.4.4 内嵌在消息框中的文件下载链接

在 JTextPane 中插入 a 标签（即链接），可以添加 addHyperlinkListener 监听到点击链接的事件。

```
msgPane.addHyperlinkListener(e -> {
  // 仅处理链接激活（点击）事件
  if (HyperlinkEvent.EventType.ACTIVATED.equals(e.getEventType())) {
    String url = String.valueOf(e.getURL()); // 读取 url 字符串
    // ...
  }
});
```

经测试，这里的 url 必须带有合法的协议头（如 `http://`）才能被正确读取。

2.4.5 消息框滚动到底部

为了使得消息框能够滚动，需要在外面套一层 JScrollPane，然后可以使用以下代码将 JTextPane 滚动到底部：

```
msgPane.setCaretPosition(msgPane.getStyledDocument().getLength());
```

3 系统部署

3.1 获取源代码

从 Github 仓库克隆

```
git clone https://github.com/CmdBlockZQG/java-chat.git
cd java-chat
```

3.2 指定服务器端口和客户端链接 IP 地址

修改 `common/Config.java`，文件内容如下：

```
package common;

public class Config {
    public static final int PORT = 1145; // 服务端口
    public static final String HOST = "127.0.0.1"; // 服务器地址
}
```

将服务端口和服务器地址改成需要的设置。

3.3 编译

在 `java-chat` 目录下执行以下命令编译（或仅编译特定模块）：

```
javac server/Server.java # 编译服务器
javac ui/Main.java # 编译客户端
```

3.4 运行

```
java server.Server # 运行服务器
java ui.Main # 运行客户端
```