基于WebSocket的实时聊天室系统报告

项目源码：<https://github.com/pei0908/simple-chat.git>

[1. 制作需求 2](#_Toc200802448)

[1.1功能需求 2](#_Toc200802449)

[1.2 非功能需求 2](#_Toc200802450)

[1.3 技术栈 3](#_Toc200802451)

[2. 系统分析 3](#_Toc200802452)

[2.1 系统架构设计 3](#_Toc200802453)

[2.2 核心代码 4](#_Toc200802454)

[2.3 关键技术点 5](#_Toc200802455)

[3. 系统运行与功能测试 7](#_Toc200802456)

[3.1运行截图 7](#_Toc200802457)

[3.2 功能测试用例 8](#_Toc200802458)

[4. 计算机网络知识应用 8](#_Toc200802459)

[4.1 WebSocket协议分析 8](#_Toc200802460)

[4.2 TCP协议保障机制 9](#_Toc200802461)

[4.3 性能优化 10](#_Toc200802462)

[5. 总结 10](#_Toc200802463)

[6. 附录：使用说明 11](#_Toc200802464)

# 制作需求

## 1.1功能需求

本系统基于计算机网络课程知识，设计并实现了一个实时聊天室系统，主要功能包括：

### 1.1.1用户身份管理：

用户首次连接需设置唯一用户名

支持在线修改用户名

用户名冲突检测

### 1.1.2实时通信功能：

全群组广播聊天

用户间私密点对点通信

系统消息通知

### 1.1.3在线状态管理：

实时显示在线用户列表

用户加入/离开通知

用户更名广播

## 1.2 非功能需求

实时性：基于WebSocket协议实现毫秒级消息传递

可靠性：利用TCP协议保证消息可靠传输

可扩展性：线程安全设计支持高并发

兼容性：支持主流现代浏览器

易用性：简洁直观的用户界面

## 1.3 技术栈

服务器端：Java WebSocket API (Tyrus实现)

客户端：HTML5 + JavaScript (WebSocket API)

通信协议：WebSocket over TCP

开发环境：JDK 8, Maven 3.9.4

网络协议：遵循RFC 6455 WebSocket标准S

# 系统分析

## 2.1 系统架构设计

本系统采用典型的C/S架构，结合计算机网络分层模型：

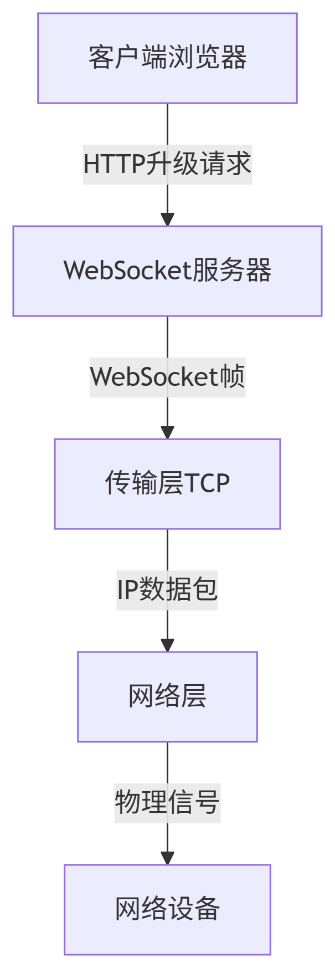


图 1系统架构设计图

## 2.2 核心代码

### 2.2.1 WebSocket连接管理

|  |
| --- |
| **WebSocket 服务器启动类 WebSocketServer**  public class WebSocketServer {  public static void main(String[] args) {  Server server = new Server("localhost", 9000, "/", Collections.*emptyMap*(), ChatServer.class);  try {  server.start();  System.*out*.println("WebSocket 服务器已启动，端口：9000");  System.*in*.read(); *// 阻塞等待退出* } catch (Exception e) {  e.printStackTrace();  } finally {  server.stop();  }  } }  **WebSocket 业务逻辑处理类 ChatServer**  @ServerEndpoint("/chat") public class ChatServer {  private static final Map<String, Session> *userSessionMap* = new ConcurrentHashMap<>();  private static final Map<Session, String> *sessionUserMap* = new ConcurrentHashMap<>();   @OnOpen  public void onOpen(Session session) {  *// 连接后提示客户端发送 SETNAME* sendMessage(session, "系统：欢迎，请发送 'SETNAME:xxx' 来设置您的用户名");  }   @OnMessage  public void onMessage(String message, Session session) {  *// 处理用户名设置或更改* if (message.startsWith("SETNAME:")) {  String newName = message.substring("SETNAME:".length()).trim();   *// 名称冲突检查* if (*userSessionMap*.containsKey(newName)) {  sendMessage(session, "系统：该用户名已被占用，请换一个");  return;  }   String oldName = *sessionUserMap*.get(session);  if (oldName != null) {  *// 移除旧绑定  userSessionMap*.remove(oldName);  }   *// 绑定新用户名  userSessionMap*.put(newName, session);  *sessionUserMap*.put(session, newName);   *// 根据是否已有旧名，分别广播“加入”或“更名”事件* if (oldName == null) {  *// 首次设置用户名* broadcast("系统：" + newName + " 加入了聊天室");  } else {  *// 更改用户名* sendMessage(session, "系统：用户名更改成功，你的用户ID是 " + newName);  broadcast("系统：" + oldName + " 更改了名字为 " + newName);  }   *// 更新并广播在线用户列表* broadcastOnlineUsers();  return;  }   *// 未设置用户名则拒绝其他消息* if (!*sessionUserMap*.containsKey(session)) {  sendMessage(session, "系统：请先发送 'SETNAME:你的名字' 来设置用户名");  return;  }   *// 处理私信功能* if (message.startsWith("to")) {  int colonIndex = message.indexOf(":");  if (colonIndex == -1) {  sendMessage(session, "系统：格式错误，正确格式是 to用户名:消息内容");  return;  }   String targetUser = message.substring(2, colonIndex).trim();  String msgContent = message.substring(colonIndex + 1).trim();  Session targetSession = *userSessionMap*.get(targetUser);   if (targetSession != null && targetSession.isOpen()) {  String senderUser = *sessionUserMap*.get(session);  sendMessage(targetSession, "私信来自 " + senderUser + "：" + msgContent);  sendMessage(session, "你发给 " + targetUser + "：" + msgContent);  } else {  sendMessage(session, "系统：用户 " + targetUser + " 不在线或不存在");  }  return;  }   *// 处理群聊消息* String sender = *sessionUserMap*.get(session);  broadcast(sender + "：" + message);  }   @OnClose  public void onClose(Session session) {  String name = *sessionUserMap*.remove(session);  if (name != null) {  *userSessionMap*.remove(name);  broadcast("系统：" + name + " 离开了聊天室");  broadcastOnlineUsers();  }  }   private void sendMessage(Session session, String message) {  session.getAsyncRemote().sendText(message);  }   private void broadcast(String message) {  *userSessionMap*.values().forEach(s -> s.getAsyncRemote().sendText(message));  }   private void broadcastOnlineUsers() {  String list = "\_\_userlist\_\_:" + String.*join*(",", *userSessionMap*.keySet());  broadcast(list);  } } |

### 2.2.2 消息处理流程

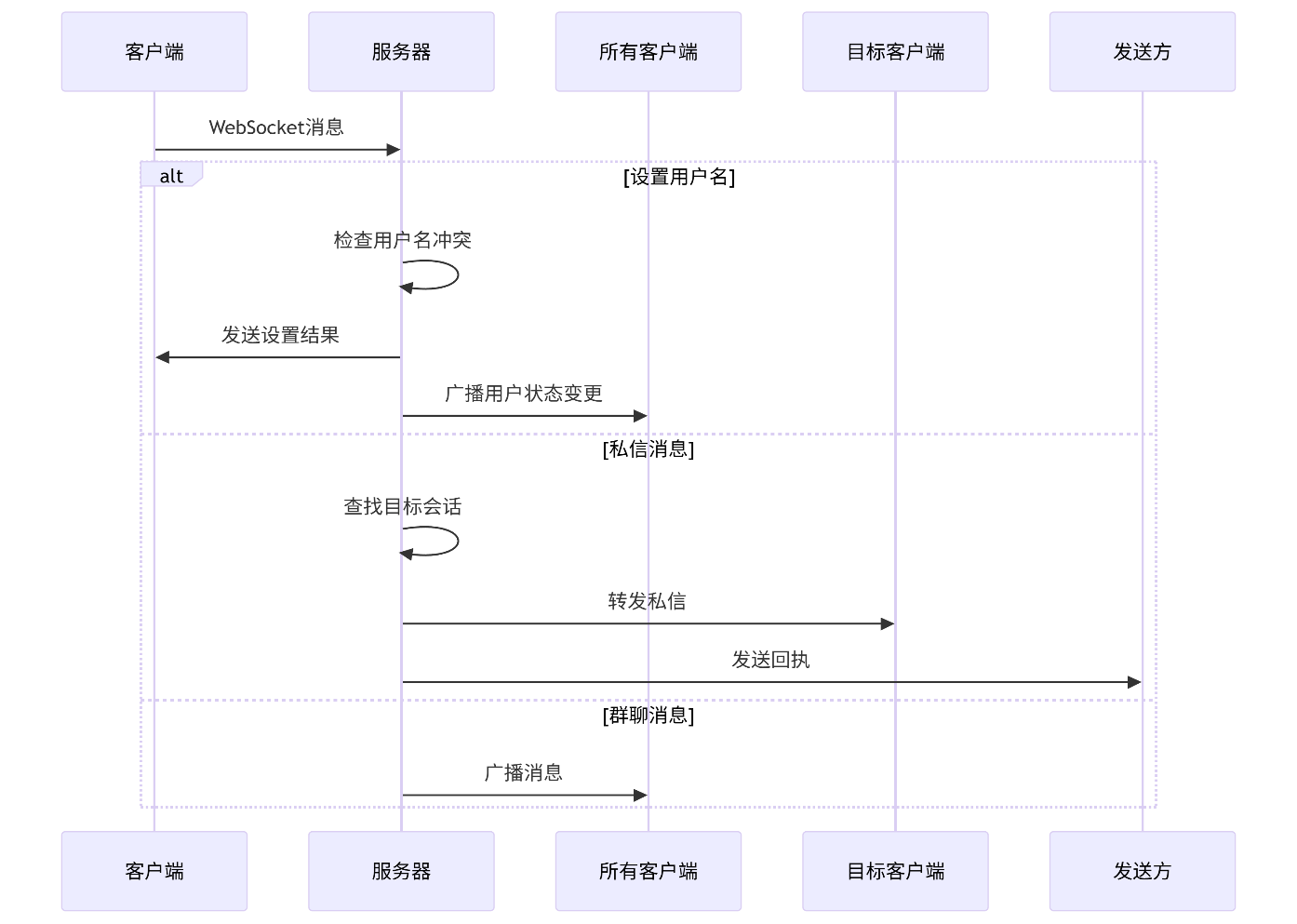


图 2消息处理流程图

## 2.3 关键技术点

### 2.3.1 WebSocket协议优势

|  |  |  |
| --- | --- | --- |
| **特性** | **HTTP轮询** | **WebSocket** |
| 连接方式 | 短连接 | 持久连接 |
| 通信模式 | 半双工 | 全双工 |
| 头部开销 | 每次请求携带完整头部 | 初始握手后仅2-14字节帧头 |
| 延迟 | 高(100ms+) | 低(10-50ms) |
| 服务器压力 | 高 | 低 |

### 2.3.2 并发处理机制

**1.线程安全设计：**

使用ConcurrentHashMap管理用户会话

避免同步锁带来的性能瓶颈

**2.异步消息发送：**

session.getAsyncRemote().sendText(message);

非阻塞I/O提高吞吐量

避免消息发送阻塞消息处理

### 2.3.3 协议设计

**控制指令：**

SETNAME:用户名 - 设置/更改用户名

to用户名:消息内容 - 发送私信

**系统消息：**

\_\_userlist\_\_:用户1,用户2,... - 在线用户列表

系统：xxx - 系统通知

# 系统运行与功能测试

## 图形用户界面, 应用程序 AI 生成的内容可能不正确。3.1运行截图

图 3运行截图1

图形用户界面, 应用程序, Word

AI 生成的内容可能不正确。

图 4运行截图2

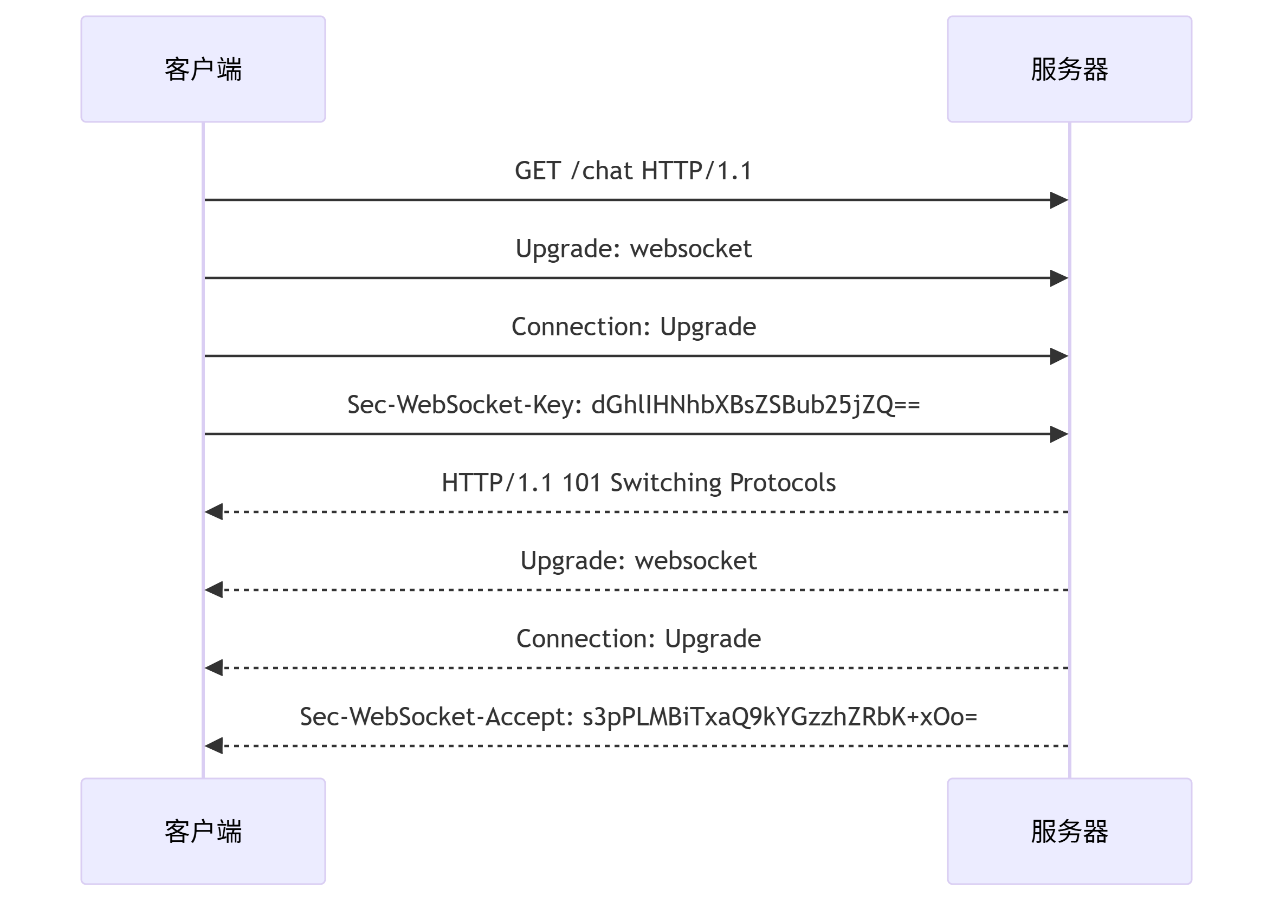
## 3.2 功能测试用例

|  |  |  |  |
| --- | --- | --- | --- |
| 测试场景 | 输入 | 预期输出 | 实际结果 |
| 设置用户名 | SETNAME:TonyStark | 用户名设置成功 | ✅ |
| 群发消息 | Hello everyone | 所有在线用户收到消息 | ✅ |
| 私信功能 | toSpiderMan:Check roof | 仅SpiderMan收到消息 | ✅ |
| 用户离开 | 关闭浏览器 | 广播离开通知 | ✅ |
| 在线列表 | 多用户在线 | 实时更新排序列表 | ✅ |

# 计算机网络知识应用

## 4.1 WebSocket协议分析

### 4.1.1 握手过程



### 4.1.2 数据帧结构

|  |
| --- |
| 0 1 2 3  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1  +-+-+-+-+-------+-+-------------+-------------------------------+  |F|R|R|R| opcode|M| Payload len | Extended payload length |  |I|S|S|S| (4) |A| (7) | (16/64) |  |N|V|V|V| |S| | (if payload len==126/127) |  | |1|2|3| |K| | |  +-+-+-+-+-------+-+-------------+ - - - - - - - - - - - - - - - +  | Extended payload length continued, if payload len == 127 |  + - - - - - - - - - - - - - - - +-------------------------------+  | |Masking-key, if MASK set to 1 |  +-------------------------------+-------------------------------+  | Masking-key (continued) | Payload Data |  +-------------------------------- - - - - - - - - - - - - - - - +  : Payload Data continued ... :  + - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +  | Payload Data continued ... |  +---------------------------------------------------------------+ |

## 4.2 TCP协议保障机制

**1.可靠传输：**

序列号与确认应答

超时重传机制

数据完整性校验

**2.流量控制：**

滑动窗口协议

动态调整发送速率

**3.拥塞控制：**

慢启动算法

拥塞避免策略

快速重传与恢复

## 4.3 性能优化

**1.异步I/O模型：**

使用getAsyncRemote()非阻塞发送

避免线程阻塞提高并发能力

**2.消息压缩：**

支持permessage-deflate扩展

减少网络传输量

**3.心跳机制：**

Ping/Pong帧保持连接

自动检测断开连接

# 总结

本系统实现了一个基于WebSocket的实时聊天室，应用了计算机网络课程中的关键知识：

协议分析：深入理解WebSocket协议在HTTP基础上的升级机制

传输层：利用TCP协议实现可靠数据传输

应用层设计：自定义简洁高效的应用层协议

并发处理：使用线程安全数据结构处理高并发

性能优化：异步I/O模型提高系统吞吐量

通过本项目，实践了从协议理解、系统设计到编码实现的全过程，加深了对实时网络通信机制的理解。系统具有良好的扩展性，未来可考虑添加消息持久化、房间分组、文件传输等功能进一步增强。

# 附录：使用说明

启动服务器：java WebSocketServer

打开chat.html前端页面

输入用户名加入聊天室

发送消息：

普通消息：直接输入内容

私信：to用户名:消息内容

更改用户名：SETNAME:新用户名

项目源码：<https://github.com/pei0908/simple-chat.git>