南昌大学

系统分析与设计报告

课程名称：WEB 程序设计

学　　院：信息工程学院　系　计算机科学与技术系

专　　业：计算机科学与技术

班　　级：计算机科学与技术 151 班

姓　　名：黄嘉诚

项目名称：基于 SSM 框架和 WebSocket 技术的在线聊天系统

任课教师：文喜

授课学期：2017　年~~~~　2018　年　2　学期

分数_____

项目组编号	CS15102
项目名称	基于 SSM 框架和 WebSocket 技术的在线聊天系统
同组成员	黄嘉诚（6103115036）温名敏（6103115035） 李浩然（6103115038）肖维　（6103115015）
项目分工	**1. 黄嘉诚** 主要负责后端部分的工作，还负责整个项目总体构建技术架构方案的设计。在对需求分析与数据库模式设计完成后，负责定义项目不同层级之间的交互接口。在与数据库进行交互 DAO 层中，负责使用 Mybatis 框架来编写数据库与各个实体类之间的 XML 映射文件。在 Service 层中，负责处理用户的主要业务逻辑和事务操作。在与前端交互的 Controller 层，使用了 Spring MVC 框架。并使用 DTO 层在 MVC 层与 Service 服务进行数据交互。在前端中，使用了 SockJS 和 STOMP 技术通过 WebSocket 与后端进行实时通信。 **2. 温名敏** **3. 李浩然** **4. 肖维**

《基于 SSM 框架和 WebSocket 技术的在线聊天系统》

系统分析及设计报告

一、需求分析

在线聊天系统是一个多人实时的社交平台。该系统提供包括注册登陆、实时聊天、管理好友、印象评价和聊天记录管理等多种功能。用户首先通过使用邮箱的方式注册，并使用相应的口令和密钥登陆系统。登陆进入系统之后可以通过用户名和注册邮箱的方式来寻找好友，并发送添加好友的请求。另一位用户收到他人的好友请求之后，可以选择接受请求，成为好友。双方成为好友了之后，用户就可以在系统中的好友列表中看到对方，并相互发送消息进行实时聊天。并且有管理聊天记录的功能，可以下载历史聊天记录。用户还可以填写自己的资料卡片，以及为好友评价自己对其的印象。只需要把鼠标移动到好友列表中的好友名字上面，就会自动弹出小提示窗，显示好友的详细资料卡片以及其好友的印象评价。

图 1 基于 SSM 框架和 WebSocket 技术的在线聊天系统整体架构描述

二、数据库设计

1. 数据库整体设计

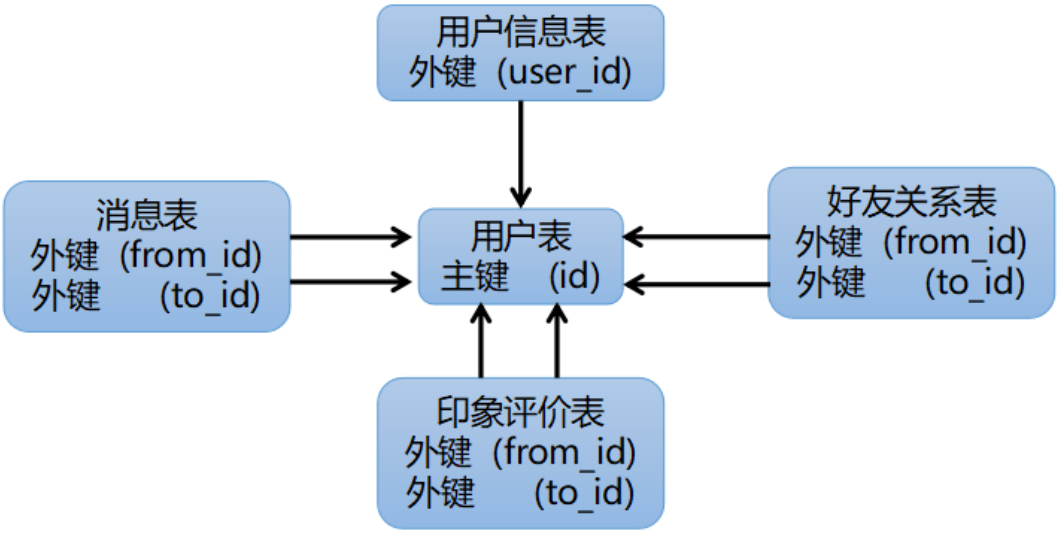

图 2 数据库关联概况

2. 具体设计详情

2.1 用户表

```
create table user (
  id       VARCHAR(50) NOT NULL, -- 用户ID
  email    VARCHAR(50), -- 用户邮箱
  password VARCHAR(50), -- 用户用MD5计算散列后的密码
  PRIMARY KEY (id)       -- 设置主键
);
```

2.2 用户信息表

```
create table impression (
  to_id       VARCHAR(50) NOT NULL,
  from_id     VARCHAR(50) NOT NULL,  -- from_id对to_id的印象评价
  description VARCHAR(100),           -- 印象内容
  PRIMARY KEY (to_id, from_id)
);
```

2.3 好友关系表

```
create table relation (
  to_id   VARCHAR(50) NOT NULL,
  from_id VARCHAR(50) NOT NULL, -- from_id有to_id的好友
  state   INT, -- 0未验证 --1已验证
  PRIMARY KEY (to_id, from_id)
);
```

2.4 消息表

```
create table message (
  to_id   VARCHAR(50) NOT NULL,
  from_id VARCHAR(50) NOT NULL,  -- from_id向to_id发送消息
  content VARCHAR(300),           -- 消息内容
  time    TIMESTAMP   NOT NULL,  -- 消息的时间
  PRIMARY KEY (to_id, from_id, time)
);
```

2.5 印象评价表

```
create table impression (
  to_id       VARCHAR(50) NOT NULL,
  from_id     VARCHAR(50) NOT NULL,  -- from_id对to_id的印象评价
  description VARCHAR(100),           -- 印象内容
  PRIMARY KEY (to_id, from_id)
);
```

三、架构设计

1. 总体技术方案

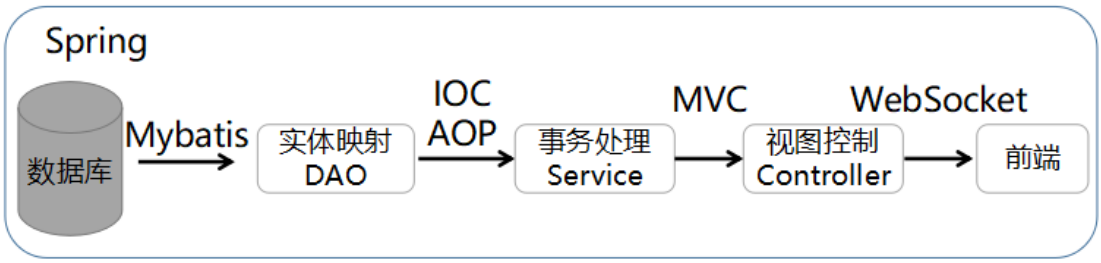

图 3 整体系统技术方案

2. 代码工程架构

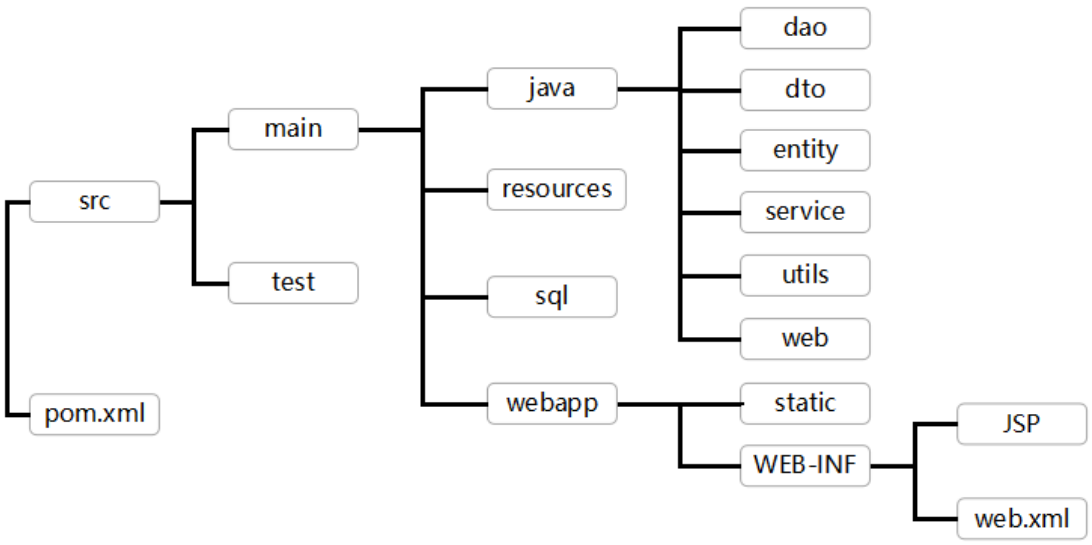

图 4 程序代码分包管理层次结构

项目的分包方式如图 4 所示，采用了典型的 maven 工程架构。主要包的作用：

文件名	作用
src	根目录，没什么好说的，下面有 main 和 test
main	主要目录，可以放 java 代码和一些资源文件
java	存放我们的 java 代码，这个文件夹要使用 Build Path -> Use as Source Folder，这样看包结构会方便很多，新建的包就相当于在这里新建文件夹
resources	存放资源文件，譬如各种的 spring，mybatis，log 配置文件
sql	存放 SQL 文件
webapp	用来存放我们前端的静态资源，如 jsp js css
WEB-INF	部浏览器无法访问，只有内部才能访问，可以把 jsp 放在这里，另外就是 web.xml
dao	数据访问层，与数据打交道，可以是数据库操作，也可以是文件读写操作，甚至是 redis 缓存操作，总之与数据操作有关的都放在这里，也有人叫做 dal 或者数据持久层都差不多意思。为什么没有 daoImpl，因为我们用的是 mybatis，所以可以直接在配置文件中实现接口的每个方法
entity	实体类，一般与数据库的表相对应，封装 dao 层取出来的数据为一个对象，也就是我们常说的 pojo，一般只在 dao 层与 service 层之间传输
dto	数据传输层，用于 service 层与 web 层之间传输
service	业务逻辑，实现业务逻辑，一般事务控制是写在这里
web	springmvc 就是在这里发挥作用的，一般人叫做 controller 控制器，相当于 struts 中的 action

四、功能实现

1. 实时聊天功能

```
@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig extends AbstractWebSocketMessageBrokerConfigurer {
    @Override
    public void configureMessageBroker(MessageBrokerRegistry registry) {
        registry.enableSimpleBroker( ...destinationPrefixes: "/topic");
        registry.setApplicationDestinationPrefixes("/app");
    }

    @Override
    public void registerStompEndpoints(StompEndpointRegistry stompEndpointRegistry) {
        stompEndpointRegistry.addEndpoint( ...strings: "/chat");
        stompEndpointRegistry.addEndpoint( ...strings: "/chat").withSockJS();
    }
}
```

图 5 WebSocket 的中间人配置

```java
@Controller
public class WebSocketController {
    @Autowired
    private SimpMessagingTemplate simpMessagingTemplate;

    @Autowired
    private WebSocketService webSocketService;

    @MessageMapping("/chat/{toId}")
    public void chat(@DestinationVariable("toId") String toId, WebSocketInMessage message) {
        WebSocketOutMessage out = webSocketService.processMessage(toId, message);
        simpMessagingTemplate.convertAndSend( destination: "/topic/chat/" + toId, out);
    }
}
```

图 6 WebSocket 的控制器

```javascript
function sendMessage(toId) {
    var textarea = document.getElementById('textarea' + toId);
    text = textarea.value;
    stompClient.send("/app/chat/"+toId, {},
        JSON.stringify({'from':'<%=currentId%>', 'text':text}));
    stompClient.send("/app/chat/<%=currentId%>", {},
        JSON.stringify({'from':'<%=currentId%>', 'text':text}));
    textarea.value = "";
}
```

图 7 前端页面中通过 STOMP 向后端监听 WebSocket 的路径发送数据

```javascript
function connect() {
    var socket = new SockJS('/webfinal/chat'); // yes
    stompClient = Stomp.over(socket);
    stompClient.connect({}, function(frame) {
        console.log('Connected: ' + frame);
        stompClient.subscribe('/topic/chat/<%=session.getAttribute("currentId")%>',
            function(messageOutput) {
            showMessageOutput(JSON.parse(messageOutput.body));
        });
    });
}
```

图 8 前端与后端建立 WebSocket 连接

2. 登陆注册功能

```
@RequestMapping(value = "/login", method = RequestMethod.POST)
private String login(@RequestParam("email") String email,
                     @RequestParam("password") String password,
                     HttpServletRequest request) {
    String idKey = "currentId";
    String nameKey = "currentName";
    Result<LoginExecution> result = userService.login(email, password);
    request.getSession().setAttribute( s: "result", result);
    if (result.isSuccess()) {  // 成功把currentId放到session中
        request.getSession().setAttribute(idKey, result.getData().getId());
        request.getSession().setAttribute(nameKey, result.getData().getName());
        return "redirect:/view/home";
    }
    return "redirect:/view/login";
}
```

图 9 登陆控制器

```
@RequestMapping(value = "/register", method = RequestMethod.POST)
private String register(@RequestParam("email") String email,
                        @RequestParam("password1") String password1,
                        @RequestParam("password2") String password2,
                        HttpServletRequest request) {
    Result result;
    if (password1 == null || !password1.equals(password2)) {
        result = new Result<>( success: false,  error: "两次密码不一致");
    } else {
        result = userService.register(email, password1);
    }
    request.getSession().setAttribute( s: "result", result);
    if (!result.isSuccess()) {
        return "redirect:/view/register";
    }
    return "redirect:/view/login";
}
```

图 10 注册控制器

3. 好友管理功能

```
@RequestMapping(value = "/findNewFriend", method = {RequestMethod.POST, RequestMethod.GET})
private String findNewFriend(@RequestParam("key") String key, HttpServletRequest request) {
    String currentId = (String) request.getSession().getAttribute( s: "currentId");
    Result<FindNewFriendExecution> result = homeService.findNewFriend(currentId, key);
    request.getSession().setAttribute( s: "searchResult", result);
    return "redirect:/view/managerFriend";
}
```

图 11 查询好友

```
@RequestMapping(value = "/addNewFriendRequest",
        method = {RequestMethod.POST, RequestMethod.GET})
private String addNewFriendRequest(@RequestParam("friendId") String friendId,
                                   HttpServletRequest request) {
    String currentId = (String) request.getSession().getAttribute( s: "currentId");
    homeService.addNewFriendRequest(currentId, friendId);
    return "redirect:/view/home";
}
```

图 12 提交好友请求

```
@RequestMapping(value = "/agreeNewFriend",
        method = {RequestMethod.POST, RequestMethod.GET})
private String agreeNewFriend(@ModelAttribute("currentId") String currentId,
                              @RequestParam("friendId") String friendId,
                              HttpServletRequest request) {
    homeService.agreeNewFriend(friendId, currentId);
    return "redirect:/view/home";
}
```

图 13 同意好友请求

```
@RequestMapping(value = "/deleteFriend",
        method = {RequestMethod.POST, RequestMethod.GET})
private String deleteFriend(@ModelAttribute("currentId") String currentId,
                            @RequestParam("friendId") String friendId,
                            HttpServletRequest request) {
    homeService.deleteFriend(currentId, friendId);
    return "redirect:/view/managerFriend";
}
```

图 14 删除好友

4. 好友印象功能

```
@RequestMapping(value = "/label", method = {RequestMethod.POST, RequestMethod.GET})
private String label(@ModelAttribute("currentId") String currentId,
                     @RequestParam("friendId") String friendId,
                     @RequestParam("content") String content,
                     HttpServletRequest request) {
    homeService.label(currentId, friendId, content);
    return "redirect:/view/home";
}
```

图 15 对好友进行印象评价

5. 用户资料卡功能

```
@RequestMapping(value = "/updateSelfInformation",
        method = {RequestMethod.POST, RequestMethod.GET})
private String updateSelfInformation(
        @ModelAttribute("currentId") String currentId,
        @RequestParam("name") String name,
        @RequestParam("gender") String gender,
        @RequestParam("motto") String motto,
        @RequestParam("school") String school,
        @RequestParam("phone") String phone,
        HttpServletRequest request
) {
    Integer intGender = null;
    if (gender != null) {
        if("男".equals(gender)) {
            intGender = 1;
        } else if ("女".equals(gender)) {
            intGender = 0;
        }
    }
    SelfInformationExecution execution = new SelfInformationExecution(
            currentId, name, email: null, intGender, motto, school, phone
    );
    // 修改session缓存中的用户名
    request.getSession().setAttribute( s: "currentName", name);
    homeService.updateSelfInformation(execution);
    return "redirect:/view/home";
}
```

图 16 修改资料卡

6. 下载聊天记录功能

```
@RequestMapping(value = "/chatHistory",
        method = {RequestMethod.POST, RequestMethod.GET})
private String chatHistory(@ModelAttribute("currentId") String currentId,
                           @RequestParam("friendId") String friendId,
                           HttpServletRequest request,
                           HttpServletResponse response) throws IOException {
    String content = homeService.chatHistory(currentId, friendId);
    String filename = "chatHistory-" + currentId + "-" + friendId;
    // 设置response的头文件下载的方式
    response.setContentType("text/plain");
    response.setHeader( s: "Content-Disposition",
            s1: "attachment; filename=" + filename + ".txt");
    ServletOutputStream resOut = response.getOutputStream();
    // 使用缓冲IO流，下载
    BufferedOutputStream out = new BufferedOutputStream(resOut);
    out.write(content.getBytes( charsetName: "UTF-8")); // 这里一定要设置编码方式
    out.flush();
    out.close();
    return "redirect:/view/downloadChat";
}
```

图 17 下载聊天历史纪录

五、界面成果展示

1. 实时聊天功能

图 18 实时聊天功能，可以在窗口实时显示。如果另一个好友发送消息，会有小红点智能提示

2. 登陆注册功能

图 19 登陆

图 20 注册

3. 好友管理功能

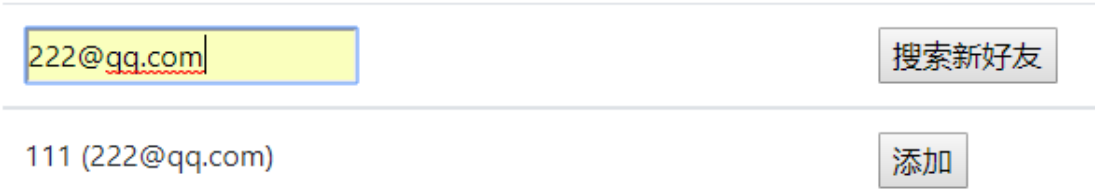

图 21 查找新好友，并提交好友请求

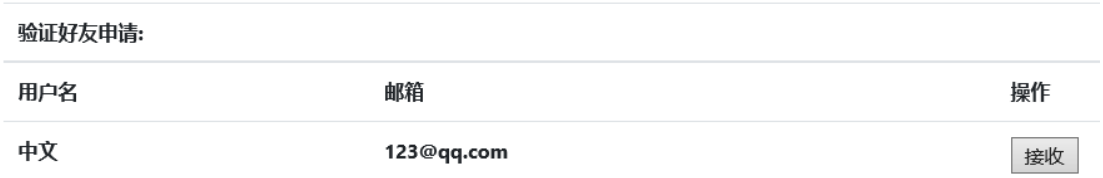

图 22 同意好友申请

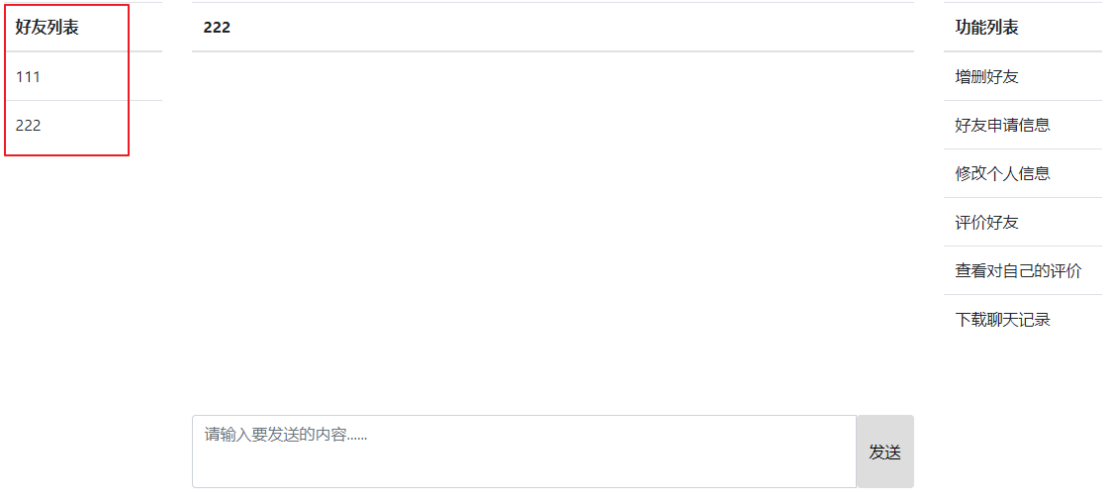

图 23 左侧会显示好友列表

图 24 删除好友

4. 好友印象功能

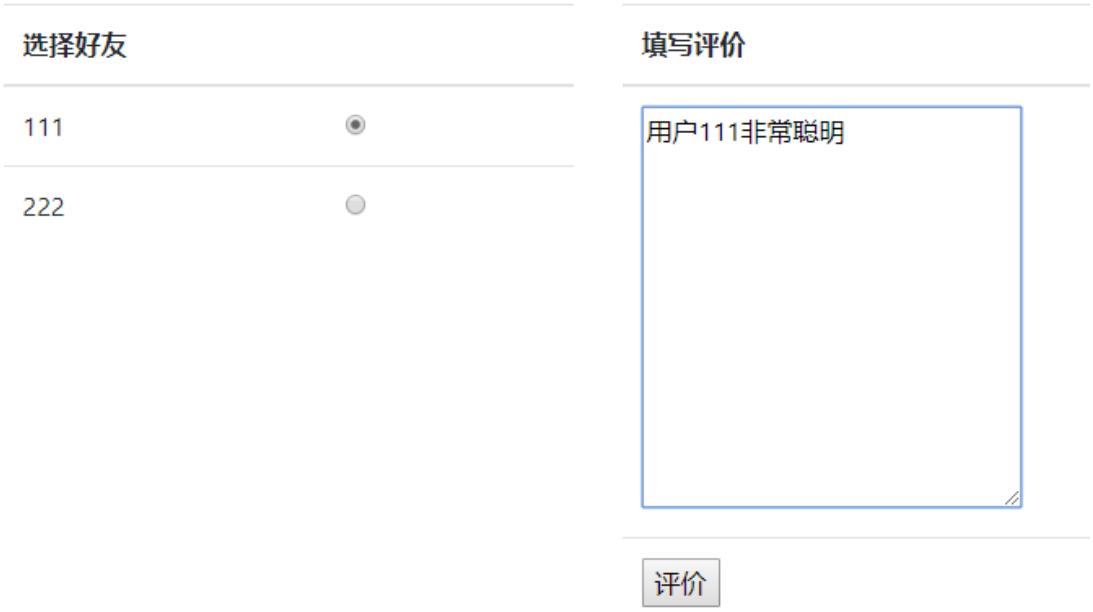

图 25 评价对好友的印象

5. 用户资料卡功能

图 26 修改个人资料

图 27 把鼠标放在好友的名字上，就会弹出好友资料卡，并包含其好友的评价

6. 下载聊天记录功能

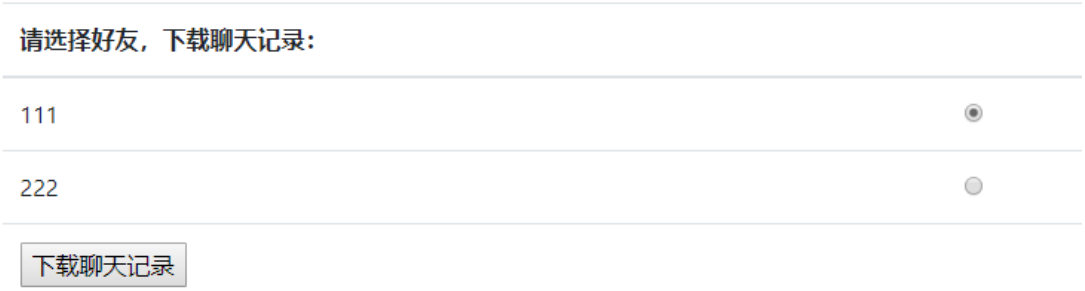

图 28 选择下载聊天记录的对象

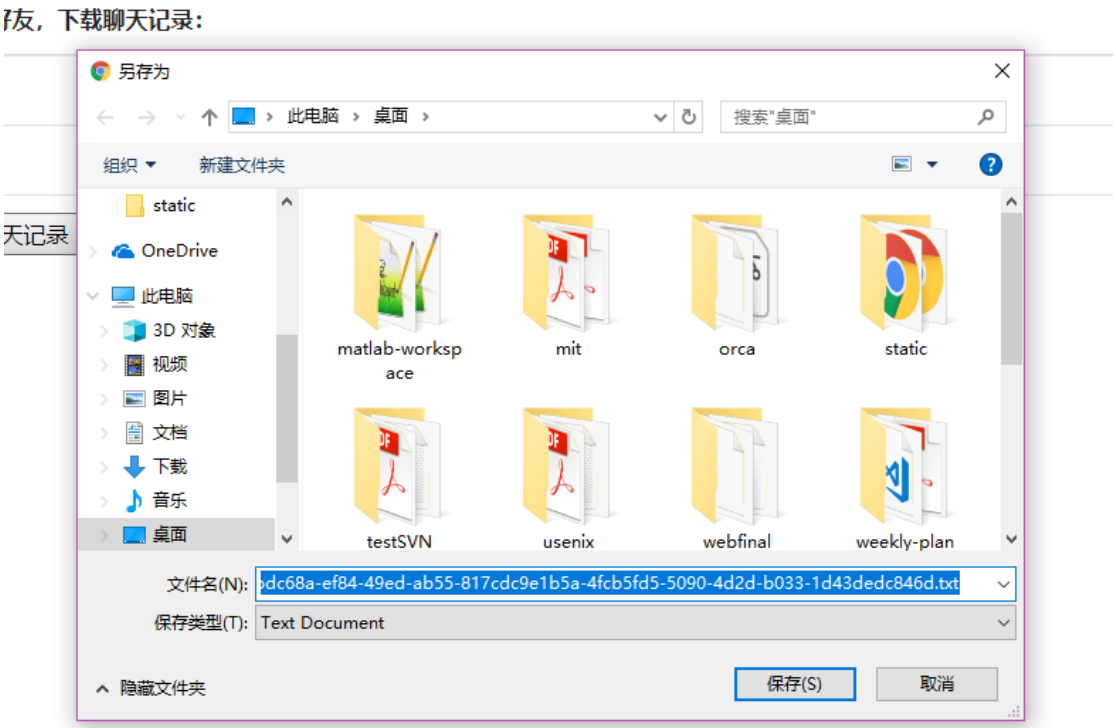

图 29 点击下载聊天记录后，浏览器自动弹出下载框

图 30 下载的聊天记录

六、心得体会

在本次 Java Web 的大作业中，我们整合了这个学期学到的所有知识包括前端的 HTML、CSS、JavaScript 和后端的 Java 技术。除此之外，我们还使用了非常多的课本之外的技术。前端与后端需要实时全双工通讯，因此我们使用了 WebSocket 技术。WebSocket 主要通过前端的 SockJS 和 STOMP 来与后端建立连接，利用 Spring 框架对 WebSocket 的支持设置好相关消息订阅和发布的路径，就可以非常好地实现聊天系统地实时通讯需求。在前端我们还使用了 Boostrap 框架， 通过使用已经定义好了的各个样式，从而较为简单地设计并实现出漂亮简约的前端页面。我们在后端使用了 Spring、Spring MVC、Mybatis 框架，大大提高了代码质量和程序效率。相比于原本的 Servlet 于 JSP 的方案，SSM 框架的使用虽然刚开始学起来感觉有点困难。但是用多了以后，就发现使用框架可以极大的提高效率。使用 Mybatis 框架通过 xml 配置建立与数据库映射大大简化了原本的 JDBC 操作，而且还可以防止 SQL 注入。利用 Spring 框架提供的依赖注入和控制反转的功能，极大地降低了整个系统的各个类之间的耦合度。Spring 提供的 AOP 的功能，减少了不必要的重复代码，特别是在数据库的事务控制过程中。Spring MVC 在 Spring 框架的基础上为与前端页面和数据交互提供了非常方便的渠道。

在这个项目中我们使用 SSM 框架和 WebSocket 的方案，较好地实现了初始的项目需求。但是在 Java Web 这个方向上还有太多的东西需要学习，我们现在学会了如何使用框架，但

是我觉得更重要的是理解框架中非常重要的功能是如何实现的。比如，Spring 通过动态代理的方式实现了 IoC 和 AOP，WebSocket 基于的网络协议等等。我们不仅要会使用框架，更要想办法探求其中的原理。框架的变化速度非常快，每年都有许多新的技术方案。但是其中的原理是永远不会过时的。在今后的学习中，我想在熟悉使用框架等工具的同时，多看看深层次的东西。例如，Spring 源码、设计模式、IO/NIO、多线程、JVM 等方面。