**审大聊天室开发项目报告**

**[项目要求]**

1、完善聊天室的基本功能（注册、登录、群发、私聊、聊天记录导出等功能）。

2、聊天室要求 文明用语，要求对不文明用语进行过滤。可以用‘ \* ’号代替禁用语，也可以用提示框 提醒用户完成修订后 发出。要有过滤词的添加与删除页面。

3、其他个性化设计（美观，前后作业的连贯，新功能）

**[最终实现功能]**

1. 聊天室登录界面，并使用**动态效果**和渐变颜色进行美化
2. 注册界面，使用企业级最严格的验证方式对用户的注册信息（身份证、手机号、邮编、出生日期）进行验证，同时利用下拉框实现全国所有**省市区三级联动**。最后在注册提交的时候需要通过**验证码**进行校验，如果输入失败则拒绝注册。
3. 通过密码错误重试次数对系统的登陆进行**访问控制**，超过三次即锁定15分钟无法访问
4. 聊天室的公共聊天区域以及和各个用户间的私聊窗口，并使**公聊和私聊**界面分离。
5. 普通用户对聊天记录的**查看与导出**功能（支持txt、csv、excel三种格式）。并能根据日期进行查询。
6. 在聊天功能中加入了**表情包**，用户可以在发送消息时加入表情包。
7. 接入**大语言模型API**。在聊天室中，一共接入了chatGPT、通义千问在内的**5种**不同语言的大模型，用户可以在发送消息的时候根据对方的内容利用**AI大语言模型**进行帮写。
8. 实现对**文件的上传和显示以及下载**功能。聊天室支持对txt、zip、pdf、png、excel、py、ppt、mp4等19种文件的传输功能，接收方还能对文件进行下载保存。
9. 利用**屏蔽词**对用户发送消息进行**过滤**，维护聊天室的聊天环境，去除带有恶意的词。
10. **管理员端**界面设计，使其能够实现对用户的基本信息的查看和修改；并能对**屏蔽词库进行维护**，修改删除或增加屏蔽词；并且可以对**聊天记录进行管理**，实现删除聊天记录和筛选聊天记录中带敏感词的记录的功能。
11. 网站的发布，通过**反向代理+动态域名绑定+内网穿透**实现外网用户对聊天室的访问，从而实现真正意义上的多人在线聊天。
12. 页面的**整体美化**。通过参考第一节课的高级网页，设计聊天室的界面，尽量逼近于QQ、微信等主流社交软件。

**[数据库设计]**

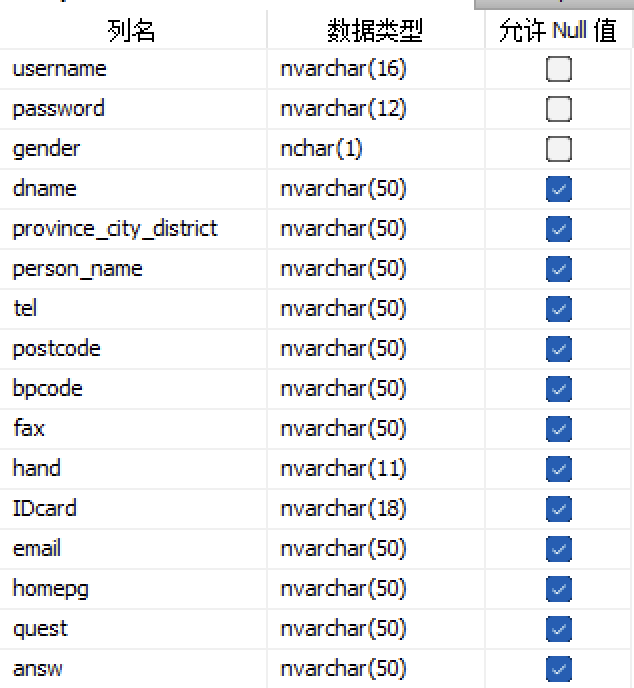
**Login表的结构设计**

username表示注册用户名；password表示注册密码



**UserInfo表的结构设计**

username表示用户名；password表示注册密码；gender表示性别；dname表示单位名称；province\_city\_district表示省市区；person\_name表示个人姓名；tel表示电话号码；postcode表示邮政编码；bpcode表示寻呼号码；fax表示传真号码；hand表示手机号码；IDcard表示身份证号码；email表示电子邮箱；homepg表示个人主页地址；quest表示忘记密码时的验证问题；answ表示验证问题的答案。



**SensitiveWords表的结构设计**

word表示需要屏蔽的敏感词



**Message表的结构设计**

date表示日期;username表示发送者用户名;message表示消息内容;receiver表示接收者用户名;file\_data表示文件数据;file\_name表示文件名;file\_type表示文件类型;message\_id表示消息编号。



**[功能实现详细说明]**

**聊天室网站的关键部分文件结构：**

├── DataBase

│   ├── webChat.mdf

│   └── webChat\_log.ldf

├── src

│   └── main

│   └── java

│   ├── DownloadFileServlet.java

│   ├── FileUploadServlet.java

│   └── back.java

└── web

├── AdminChat.jsp

├── ChangePassword.jsp

├── Chat.jsp

├── ChatRecordManagement.jsp

├── ExportRecord.jsp

├── Loginbysql.jsp

├── RegisterToSql.jsp

├── SaveToSql.jsp

├── SensitiveWordsMaintenance.jsp

├── UserInfoMaintenance.jsp

├── UserManage.jsp

├── WebChat\_AdminListMsg.jsp

├── WebChat\_AdminSendMsg.jsp

├── WebChat\_HistoryRecord.jsp

├── WebChat\_ListMsg.jsp

├── WebChat\_Logout.jsp

├── WebChat\_Maintenance.jsp

├── WebChat\_SendMsg.jsp

├── data.js

└── web

└── WEB-INF

└── web.xml

**注册界面(带验证码校验)**

最终效果图：



验证逻辑：

用户名验证：验证用户名是否为空，并确保用户名符合一定的格式规则（字母开头，后续可以包含字母、数字和下划线，长度为4到16个字符）。正则表达式如下：/^[a-zA-Z][a-zA-Z0-9\_]{3,15}$/

密码验证：密码不能为空，长度不能超过12位，并且确认密码与输入密码必须一致。

单位名称验证：验证单位名称是否为空。

联系人姓名验证：验证联系人姓名是否为空，并且姓名只能包含中文或英文字符。正则表达式：/^([\u4e00-\u9fa5]+|([a-zA-Z]+\s?)+)$/

电话号码验证：如果电话号码输入不为空，则验证其格式是否正确（如中国区号的座机号码）。正则表达式：/^0\d{2,3}-\d{7,8}$/

手机号验证：验证手机号码格式是否正确（如中国手机号）。正则表达式：/^1[34578]\d{9}$/

身份证验证：验证身份证号码格式是否合法，包括校验位的正确性。正则表达式：/^[1-9]\d{5}(19|20)\d{2}(0[1-9]|1[0-2])(0[1-9]|[12]\d|3[01])\d{3}(\d|X)$/

邮箱验证：验证邮箱地址格式是否符合国际标准。正则表达式：/^(?:[a-z0-9!#$%&'\*+/=?^\_{|}~-]+(?:.[a-z0-9!#$%&'\*+/=?^\_{|}~-]+)\*|"(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21\x23-\x5b\x5d-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f])\*")@(?:(?:[a-z0-9](?:[a-z0-9-]\*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]\*[a-z0-9])?|\[(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?|[a-z0-9-]\*[a-z0-9]:(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21-\x5a\x53-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f])+)\])$/

邮政编码验证：验证邮政编码是否符合规范（六位数字）。正则表达式：/^[1-9]\d{5}(?!\d)$/

寻呼号码验证：如果输入了寻呼号码，验证其是否为7到10位的数字。正则表达式：/^\d{7,10}$/

传真号码验证：如果输入了传真号码，验证其格式是否符合（如带区号的传真号码）。正则表达式：/^\d{1,3}-\d{1,3}-\d{4}$/

忘记密码问题验证：验证忘记密码问题是否为空，并且长度在5到100字符之间，且不包含非法字符。正则表达式：/^[a-zA-Z0-9\u4e00-\u9fa5\s,.\?!:;'"-]+$/

忘记密码答案验证：验证答案是否为空，长度是否在5到100字符之间，并且不包含非法字符。正则表达式：/^[a-zA-Z0-9\u4e00-\u9fa5\s,.\?!:;'"-]+$/

主页地址验证：验证主页地址是否为有效的URL。正则表达式：/^(https?|ftp):\/\/([a-zA-Z0-9.-]+(\:[0-9]+)?)(\/[a-zA-Z0-9%\_.~+-]+)\*\/?(\?[a-zA-Z0-9%\_.,~+-=&]\*)?(#[a-zA-Z0-9\_-]+)?$/

地址验证：验证省、市、区的选择是否完整。

验证码验证：验证用户输入的验证码是否为空，并且与生成的验证码一致。

验证码的生成过程：

从字符集 charsArray 中随机选取 6 个字符并保存在 captcha 数组中，确保字符不重复。最终生成的验证码字符串赋值给 generatedCaptcha。var charsArray = "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";

var lengthOtp = 6; // 验证码的长度

var captcha = [];

for (var i = 0; i < lengthOtp; i++) {

var index = Math.floor(Math.random() \* charsArray.length);

if (captcha.indexOf(charsArray[index]) == -1)

captcha.push(charsArray[index]);

else i--;

}

generatedCaptcha = captcha.join(""); // 生成验证码字符串

清空画布填充内容：  
ctx.clearRect(0, 0, canvas.width, canvas.height);

ctx.fillStyle = "#f2f2f2"; // 背景颜色

ctx.fillRect(0, 0, canvas.width, canvas.height);

添加干扰线：

for (var i = 0; i < 5; i++) {

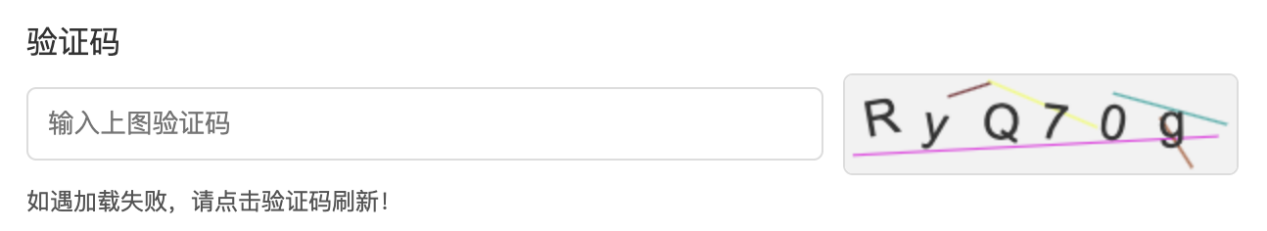
ctx.strokeStyle = getRandomColor(); // 随机颜色

ctx.beginPath();

ctx.moveTo(Math.random() \* canvas.width, Math.random() \* canvas.height);

ctx.lineTo(Math.random() \* canvas.width, Math.random() \* canvas.height);

ctx.stroke();

}

保存注册数据过程：  
注册数据保存过程首先通过 request.getParameter 获取用户提交的表单数据，并使用辅助方法 trimAndNullIfEmpty 对数据进行处理，确保空值和无效数据被清除。然后，使用 JDBC 连接到数据库，构建插入语句将用户的个人信息和登录信息分别插入 UserInfo 和 Login 表中。通过执行 SQL 插入语句，将数据写入数据库，如果两个表的插入操作都成功，页面会重定向到登录页面，否则会显示错误提示。最后，关闭数据库连接和相关资源，确保系统资源得到有效释放。

主要代码为：  
// 构建插入UserInfo表的语句

String userInfoSql = "INSERT INTO webChat.dbo.UserInfo (username, password, gender, person\_name, province\_city\_district, tel, postcode, bpcode, fax, hand, IDcard, email, homepg, quest, answ) VALUES ('"

+ username + "', '" + password + "', '" + gender + "', '" + person\_name + "', '"

+ (province != null ? province : "")

+ (city != null ? city : "")

+ (district != null ? district : "") + "', '"

+ tel + "', '" + postcode + "', '" + bpcode + "', '" + fax + "', '" + hand + "', '"

+ IDcard + "', '"

+ email + "', '" + homepg + "', '" + quest + "', '" + answ + "')";

// 构建插入Login表的语句

String loginSql = "INSERT INTO webChat.dbo.Login (username, password) VALUES ('" + username + "', '" + password + "')";

// 执行更新

int rowsAffectedUserInfo = 0;

int rowsAffectedLogin = 0;

// 先执行UserInfo表的插入

if (username != null && password != null) {

rowsAffectedUserInfo = stmt.executeUpdate(userInfoSql);

}

// 然后执行Login表的插入

if (rowsAffectedUserInfo > 0) {

rowsAffectedLogin = stmt.executeUpdate(loginSql);

}

// 检查是否两个表都成功插入

if (rowsAffectedUserInfo > 0 && rowsAffectedLogin > 0) {

response.sendRedirect("Loginbysql.jsp");

} else {

out.println("<script>alert('Failed to insert data into one or both tables.');</script>");

}

**登陆界面(带访问控制)**

最终效果图：

****

登录逻辑主要分为账户锁定机制和用户名密码验证两部分。首先，系统会检查用户在当前会话中是否有连续错误登录的记录（即errorCount和lockTime）。如果错误次数超过限制（3次），系统会锁定账户并记录锁定时间，阻止用户继续登录，直到锁定时间过了15分钟。如果账户没有被锁定，系统将继续验证用户输入的用户名和密码。如果用户名或密码为空，会给出相应提示；如果不为空，则通过JDBC连接到数据库，检查该用户名和密码是否存在匹配的记录。如果验证通过，系统会清空错误计数并重定向到聊天页面，如果验证失败，错误次数会增加，达到限制时账户将被锁定。

访问控制部分根据用户的身份来决定访问权限。如果用户名和密码验证成功，系统会检查是否是管理员账号（YYF222090140），如果是管理员，则重定向到管理员页面；否则，重定向到普通用户的聊天页面。在登录过程中，还包含了一个错误提示机制，如果用户输入错误的用户名或密码，系统会提示剩余可尝试的次数，并在多次失败后锁定账户，防止暴力破解攻击。

代码如下：

// 获取用户名和密码

String username = request.getParameter("username");

String password = request.getParameter("password");

String errorMessage = null;

Integer errorCount = (Integer) session.getAttribute("errorCount");

Long lockTime = (Long) session.getAttribute("lockTime");

if (errorCount == null) {

errorCount = 0;

}

// 检查账户是否被锁定

if (lockTime != null) {

long elapsed = System.currentTimeMillis() - lockTime;

if (elapsed > 900000) { // 锁定时间超过 15 分钟

errorCount = 0;

session.removeAttribute("lockTime");

session.setAttribute("errorCount", errorCount);

} else {

long remainingTime = 900000 - elapsed;

long seconds = remainingTime / 1000;

long minutes = seconds / 60;

seconds = seconds % 60;

errorMessage = "账户被锁定，请等待 " + minutes + " 分 " + seconds + " 秒 后再试。";

}

}

// 如果账户没有被锁定，继续进行用户名和密码验证

if (errorMessage == null) {

if (username != null && password != null) {

if (username.trim().isEmpty()) {

errorMessage = "用户名不能为空";

} else if (password.trim().isEmpty()) {

errorMessage = "密码不能为空";

} else {

// 使用 SQL Server JDBC 驱动

Security.setProperty("jdk.tls.disabledAlgorithms", "");

System.setProperty("https.protocols", "TLSv1");

Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");

// 数据库连接

Connection conn = DriverManager.getConnection(

"jdbc:sqlserver://10.211.55.7:1433;databaseName=webChat;" +

"encrypt=true;" +

"trustServerCertificate=true;" +

"sslProtocol=TLSv1;" +

"disableStatementPooling=true;" +

"cancelQueryTimeout=0;" +

"socketTimeout=120",

"sa", "123456");

// 建立连接并执行查询

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery("SELECT \* FROM webChat.dbo.Login WHERE username='" + username + "' AND password='" + password + "'");

// 检查是否有匹配的结果

if (rs.next()) {

session.setAttribute("errorCount", 0);

session.removeAttribute("lockTime");

rs.close();

stmt.close();

conn.close();

session.setAttribute("user", username);

// 判断是否为管理员账号

if ("YYF222090140".equals(username)) {

response.sendRedirect("AdminChat.jsp");

} else {

response.sendRedirect("Chat.jsp");

}

return;

} else {

errorCount++;

session.setAttribute("errorCount", errorCount);

if (errorCount >= 3) {

session.setAttribute("lockTime", System.currentTimeMillis()); // 错误次数过多，设置锁定时间

errorMessage = "密码错误，尝试次数过多，请等待 15 分钟后再试。";

} else {

errorMessage = "密码错误，还有 " + (3 - errorCount) + " 次机会";

}

}

rs.close();

stmt.close();

conn.close();

}

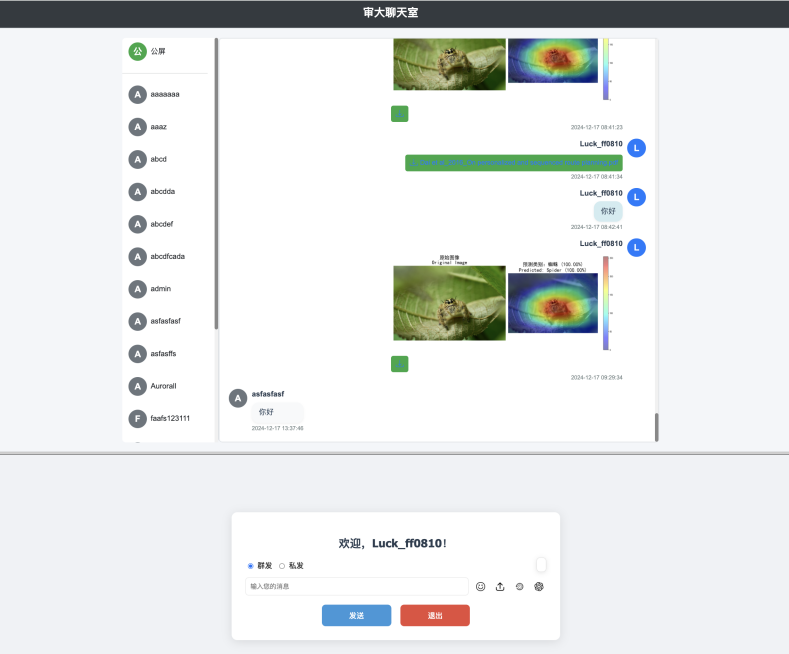
}

}

**聊天室界面(布局展示)**

最终效果图：

公共聊天区域的内容



与其他用户间私聊的内容：



聊天室总体界面分为两部分，上半部分为聊天记录展示区，下半部分为聊天记录发送区域。其中聊天记录展示区左边为选择展示与哪个用户的聊天记录。发送区域可以选择的功能按钮：群发、私发、表情包、上传文件、聊天记录管理、AI对话。

实现的思路步骤为：

1. 获取当前登录用户信息：首先从会话（session）中获取当前登录的用户名（currentUsername），以此来确定是哪位用户在查看聊天记录，后续基于该用户名筛选出与之相关的聊天消息。

String currentUsername = (String) session.getAttribute("user");

String selectedUser = request.getParameter("selectedUser");

1. 确定聊天对象及查询条件：通过获取请求参数（selectedUser）判断当前选择的聊天对象是谁。如果是 “everyone” 则表示公屏聊天，查询公屏消息的相关记录；如果是具体某个用户名，则查询与之对应的私聊消息记录。通过构建不同的 SQL 查询语句来适配公屏和私聊这两种场景。
2. 连接数据库并执行查询：根据前面确定的查询条件（不同的 SQL 语句），通过PreparedStatement预编译语句执行查询操作，获取包含聊天记录相关信息的结果集（ResultSet）。

4.筛选并显示聊天记录：遍历结果集，针对每条记录判断是否应该显示（根据公屏还是私聊以及发送者、接收者与当前登录用户的关系等条件判断），如果满足显示条件，则根据消息是当前用户发送的还是其他用户发送的，对于包含文件的消息，还会根据文件类型进一步处理，比如图片文件会进行 Base64 编码后展示预览图并提供下载按钮，其他类型文件直接提供下载链接。

关键代码如下：  
if (selectedUser!= null &&!selectedUser.isEmpty()) {

if ("everyone".equals(selectedUser)) {

// Query public messages

sql = "SELECT message\_id, username, message, date, receiver, file\_data, file\_name, file\_type FROM Message " +

"WHERE receiver = 'everyone' " +

"ORDER BY date ASC";

pstmt = conn.prepareStatement(sql, ResultSet.TYPE\_SCROLL\_INSENSITIVE, ResultSet.CONCUR\_READ\_ONLY);

} else {

// Query private chat messages

sql = "SELECT message\_id, username, message, date, receiver, file\_data, file\_name, file\_type FROM Message " +

"WHERE (username =? AND receiver =?) OR (username =? AND receiver =?) " +

"ORDER BY date ASC";

pstmt = conn.prepareStatement(sql, ResultSet.TYPE\_SCROLL\_INSENSITIVE, ResultSet.CONCUR\_READ\_ONLY);

pstmt.setString(1, currentUsername);

pstmt.setString(2, selectedUser);

pstmt.setString(3, selectedUser);

pstmt.setString(4, currentUsername);

}

}

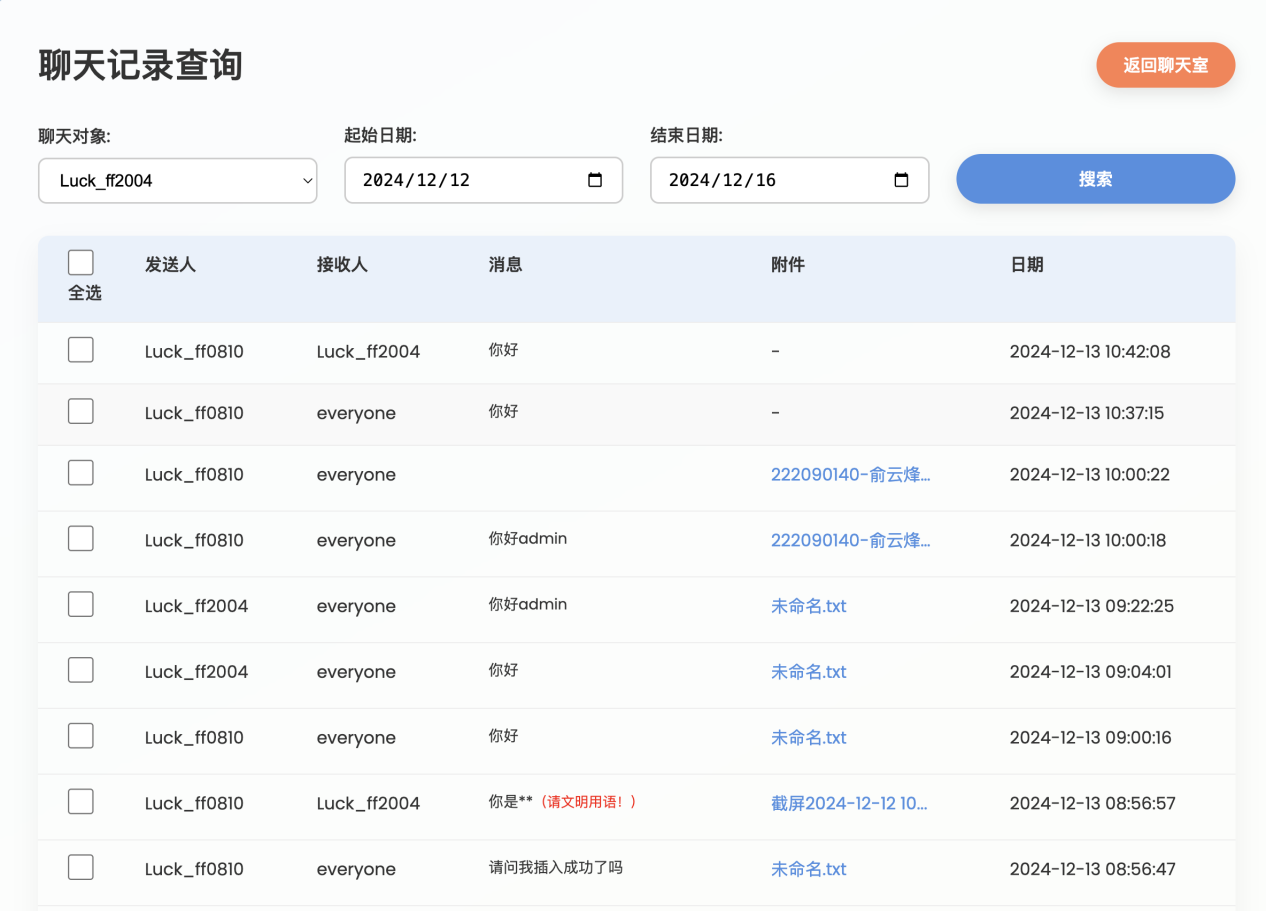
**聊天记录查询和导出**

最终效果图：

初始界面，选择查询用户记录。



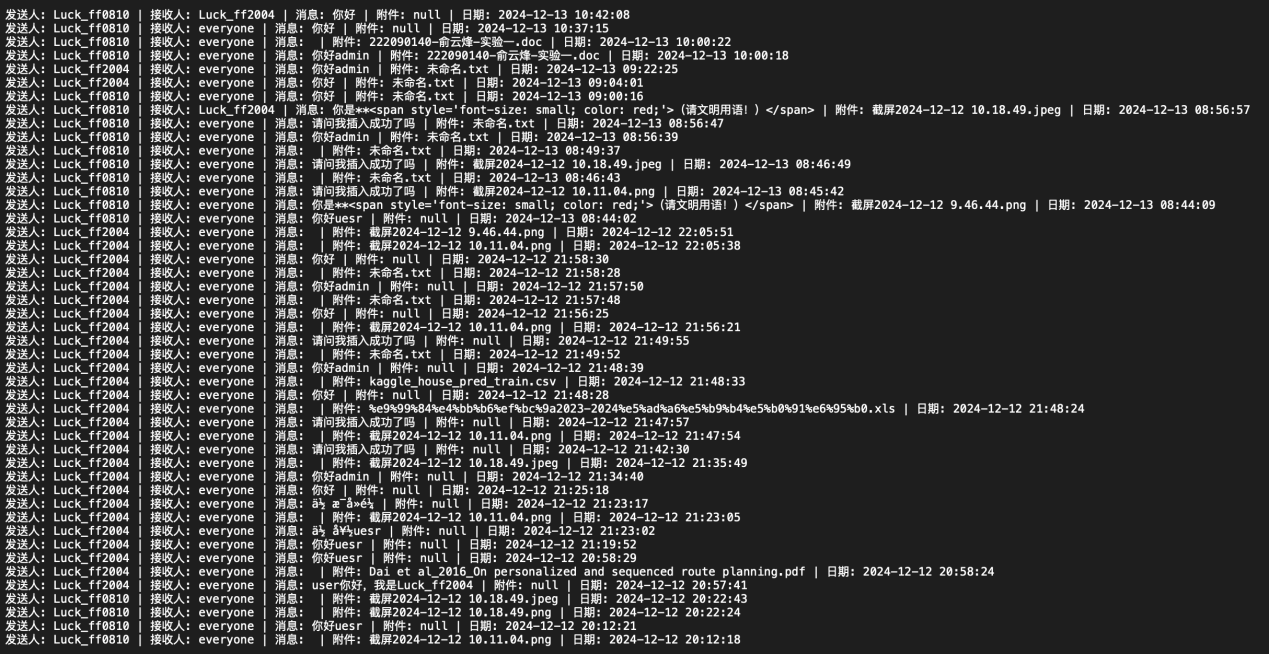
查询到的记录展示界面



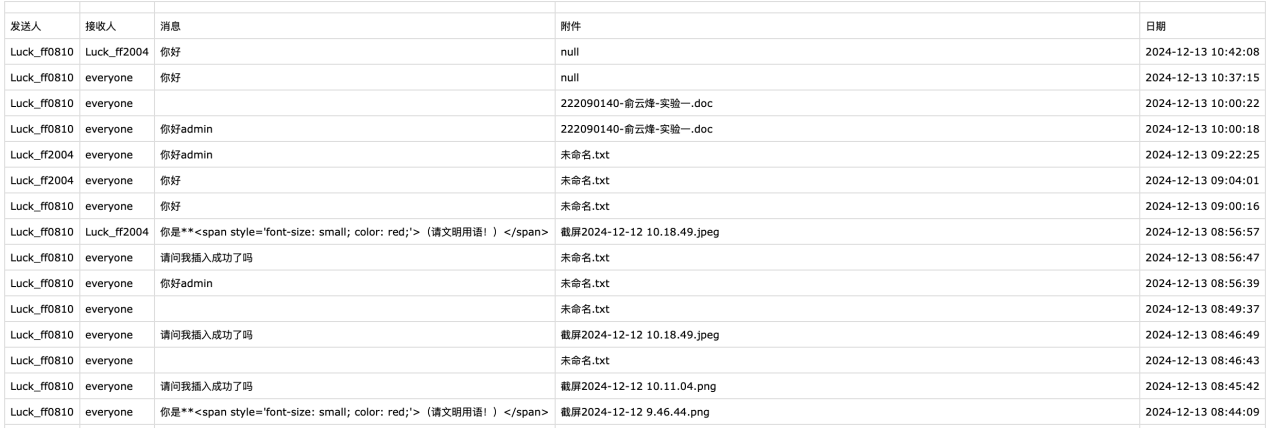
聊天记录的三类导出格式的按钮



以txt格式导出的聊天记录情况：



以excel和csv格式导出的聊天记录情况：



实现步骤：

首先，在页面加载时会进行一些初始化操作与前置判断。通过session.getAttribute("user")获取当前用户信息，若当前用户名（currentUsername）为空，则直接重定向到登录页面（Loginbysql.jsp），终止后续流程。

接着，处理查询及显示相关逻辑。一是获取查询相关的请求参数，包括exportType（用于判断是否是导出操作）、sender（聊天对象）、startDate（起始日期）和endDate（结束日期），这些参数将用于构建查询条件筛选聊天记录。

然后进入数据库操作部分：

1.尝试加载com.microsoft.sqlserver.jdbc.SQLServerDriver数据库驱动，建立与指定配置（如数据库地址、加密、协议等相关设置）的 SQL Server 数据库连接。

2.根据获取到的sender参数情况动态构建查询语句的条件部分。若sender不为空且已选择聊天对象，构建的条件会筛选出涉及当前用户与所选聊天对象之间（包括公屏消息情况）的聊天记录；若sender为空，则构建筛选与当前用户相关所有消息的条件。同时，还会结合startDate和endDate进一步细化日期范围的筛选条件，最终形成完整的 SQL 查询语句，用于从Message表中获取符合条件的记录，查询结果包含消息编号、发送人、消息内容、接收人、日期和文件名等字段信息。

在获取到查询结果集（ResultSet）后：

3.循环遍历结果集，将每条记录的各字段信息提取出来，封装到Map中，再将这些Map添加到List<Map<String, String>>类型的chatHistory集合里，以此构建出便于后续操作处理的聊天记录数据结构，同时把这个聊天记录集合以及exportType存放到session中（用于后续可能的导出操作）。

之后，开始构建页面展示部分：

1.根据请求方法是否为POST且exportType为null来判断是否需要展示查询结果（避免在导出等其他操作时重复展示）。若满足展示条件，则再次建立数据库连接，按照之前同样的规则构建查询语句并执行查询获取结果集。

2.创建 HTML 表格（<table>）用于展示聊天记录，在表格头部（<thead>）定义各列标题，如全选框、发送人、接收人、消息、附件、日期等列。

3.循环遍历结果集进行表格内容填充。

4.若遍历完结果集发现没有符合条件的记录，则在表格中显示一行提示“没有查询到聊天记录”的内容。

5.最后，在表格下方创建“导出”按钮区域，提供“导出TXT”“导出CSV”“导出Excel”等不同格式导出的按钮，点击按钮将触发相应的导出操作（实际导出逻辑在后续ExportRecord.jsp等相关页面中处理），并且通过 JavaScript 代码实现全选框（selectAll）的功能，即点击全选框能控制所有记录对应的复选框同步选中或取消选中状态。

关键代码：

String currentUsername = (String) session.getAttribute("user");

if (currentUsername == null) {

response.sendRedirect("Loginbysql.jsp");

return;

}

String exportType = request.getParameter("exportType");

if (exportType!= null) {

// 后续处理导出相关逻辑

return;

}

String sender = request.getParameter("sender");

String startDate = request.getParameter("startDate");

String endDate = request.getParameter("endDate");

StringBuilder sb = new StringBuilder();

List<Object> params = new ArrayList<>();

if (sender!= null &&!sender.isEmpty()) {

sb.append("((username =? AND (receiver =? OR receiver = 'everyone')) OR (username =? AND (receiver =? OR receiver = 'everyone')))");

params.add(sender);

params.add(currentUsername);

params.add(currentUsername);

params.add(sender);

} else {

sb.append("((username <>? AND (receiver =? OR receiver = 'everyone')) OR (username =?))");

params.add(currentUsername);

params.add(currentUsername);

params.add(currentUsername);

}

if (startDate!= null &&!startDate.isEmpty()) {

sb.append(" AND date >=?");

params.add(startDate);

}

if (endDate!= null &&!endDate.isEmpty()) {

sb.append(" AND date <=?");

params.add(endDate);

}

String query = "SELECT message\_id, username, message, receiver, date, file\_name FROM Message WHERE " + sb.toString() + " ORDER BY date DESC";

PreparedStatement pstmt = conn.prepareStatement(query);

for (int i = 0; i < params.size(); i++) {

pstmt.setObject(i + 1, params.get(i));

}

ResultSet rs = pstmt.executeQuery();

List<Map<String, String>> chatHistory = new ArrayList<>();

while (rs.next()) {

Map<String, String> record = new HashMap<>();

record.put("message\_id", String.valueOf(rs.getInt("message\_id")));

record.put("username", rs.getString("username"));

record.put("message", rs.getString("message"));

record.put("receiver", rs.getString("receiver"));

record.put("date", rs.getString("date"));

record.put("file\_name", rs.getString("file\_name"));

chatHistory.add(record);

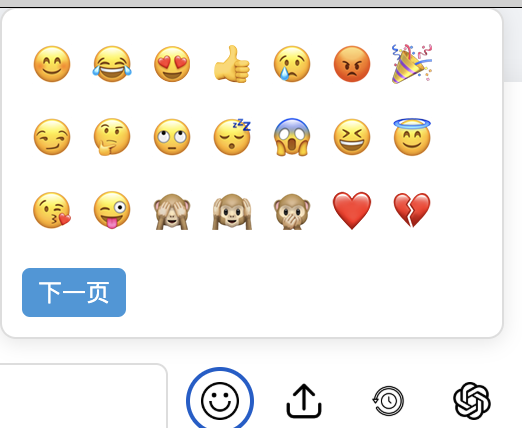
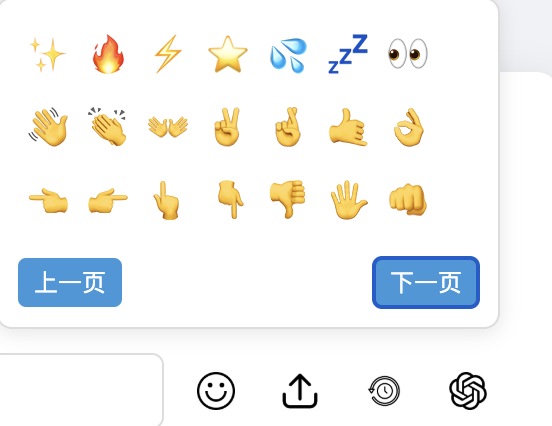
}

session.setAttribute("chatHistoryExport", chatHistory);

**表情包**

最终效果图：

选择表情包

** **

把表情包放到消息中



消息框中显示表情包



实现细节：

首先，通过定义一个名为allEmojis的数组，将各式各样的表情符号罗列其中，构成了整个表情包功能的基础数据来源。

核心的分页展示表情页面部分，设定了currentPage（初始为 0 标记页码）和pageSize（为 21 代表每页表情数量）两个变量。showEmojiPage函数按页码展示表情，先获取emojiGrid元素清空内容，依当前页码和pageSize算出索引截取当前页表情数组，遍历数组为表情创建<span>元素并绑定点击事件（点击调用insertEmoji函数插入表情到输入框），再添加到emojiGrid展示，同时根据页码控制上一页、下一页按钮显示隐藏。

翻页操作靠prevEmojiPage（当前页码大于 0 就减 1 并调用showEmojiPage往前翻页）和nextEmojiPage（先算出最大页码，当前页码小于它就加 1 并调用showEmojiPage往后翻页）两个函数实现。

表情插入功能通过insertEmoji函数实现，先获取消息输入框元素，将传入表情添加到其值后，再隐藏表情面板，让输入框重获焦点，保障输入操作流畅。

关键代码：  
// 表情数据准备

var allEmojis = [

//

];

// 分页展示表情页面相关变量

var currentPage = 0;

var pageSize = 21;

function showEmojiPage(page) {

var grid = document.getElementById('emojiGrid');

grid.innerHTML = '';

var start = page \* pageSize;

var end = start + pageSize;

var pageEmojis = allEmojis.slice(start, end);

pageEmojis.forEach(function (e) {

var span = document.createElement('span');

span.textContent = e;

span.onclick = function () { insertEmoji(e) };

grid.appendChild(span);

});

var shouldShowPrev = (page > 0);

var shouldShowNext = (end < allEmojis.length);

updatePaginationButtonsVisibility(shouldShowPrev, shouldShowNext);

}

function updatePaginationButtonsVisibility(showPrev, showNext) {

var prevPageBtn = document.getElementById('prevPageBtn');

var nextPageBtn = document.getElementById('nextPageBtn');

if (showPrev) {

prevPageBtn.style.display = 'inline-block';

} else {

prevPageBtn.style.display = 'none';

}

if (showNext) {

nextPageBtn.style.display = 'inline-block';

} else {

nextPageBtn.style.display = 'none';

}

}

// 向前翻页函数

function prevEmojiPage() {

if (currentPage > 0) {

currentPage--;

showEmojiPage(currentPage);

}

}

// 向后翻页函数

function nextEmojiPage() {

var maxPage = Math.ceil(allEmojis.length / pageSize) - 1;

if (currentPage < maxPage) {

currentPage++;

showEmojiPage(currentPage);

}

}

// 表情插入函数

function insertEmoji(emoji) {

var messageInput = document.getElementById('message');

messageInput.value += emoji;

var panel = document.getElementById('emojiPanel');

panel.style.display = 'none';

messageInput.focus();

}

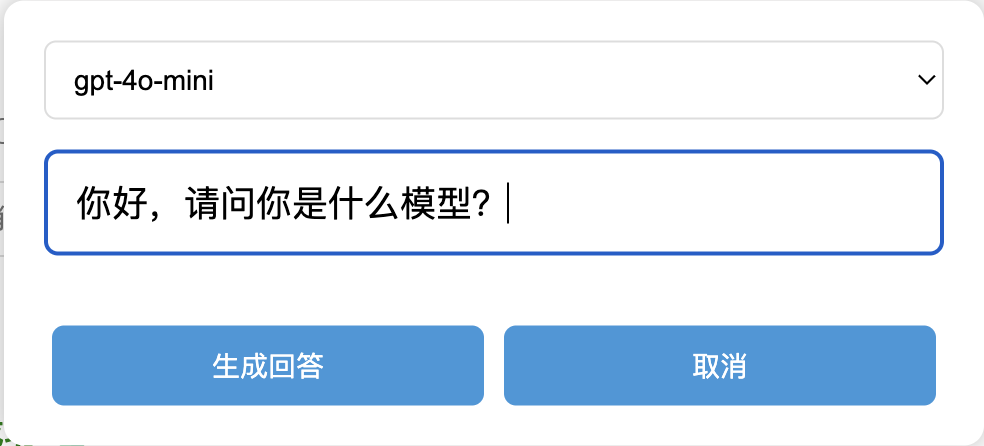
**大语言模型接入实现**

最终效果图：

选择模型界面

****

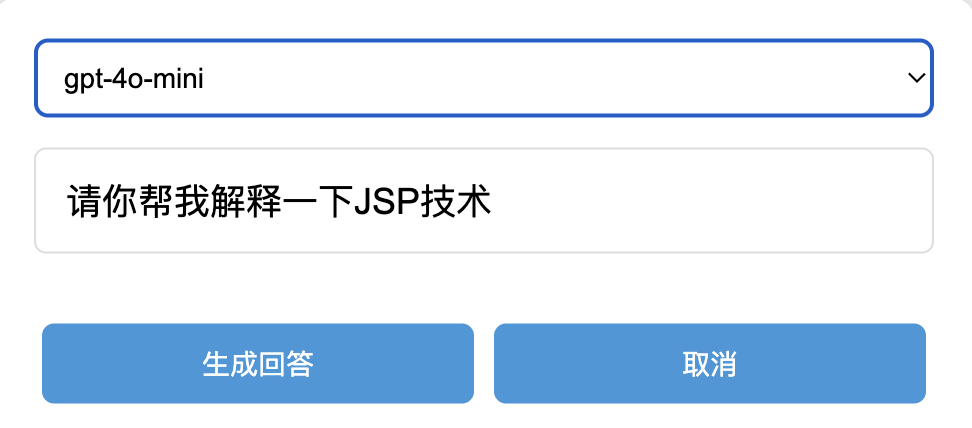
**输入需要询问的问题**

****

**生成回答后自动填入聊天室**

****

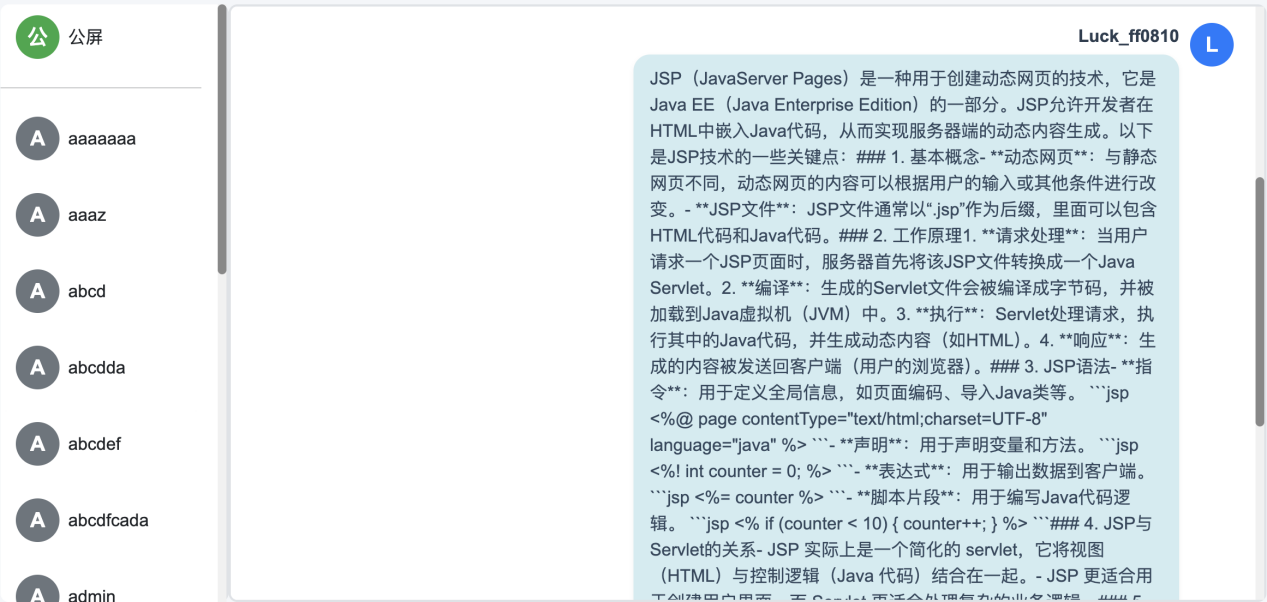
**让gpt帮我们解释专有名词并解释**

****

**生成回答并选择私发对象**

****

**发送后在聊天记录中显示**

****

思路：

先定义包含不同 AI 模型配置信息的AI\_MODELS对象。通过openAIInput函数显示 AI 输入界面并动态生成模型选择选项，即遍历AI\_MODELS对象为下拉框添加对应选项并展示相关容器。closeAIInput函数负责隐藏该容器。核心的generateAIResponse函数，先获取用户输入和所选模型，若输入为空或没选模型则提示相应错误，然后按所选模型类型（如qwen或openai）构建请求体，用fetch向对应 API 端点发 POST 请求，若响应正常就解析提取 AI 回复填入聊天输入框并隐藏界面，若响应异常则抛出对应错误提示，catch块统一处理错误，以此实现多模型的 AI 对话及回复展示功能。关键代码涉及AI\_MODELS对象定义、各函数内的 DOM 操作（如获取元素、设置显示隐藏等）以及发请求、处理响应的相关代码部分。

关键代码：  
const AI\_MODELS = {

// 通义千问模型

'qwen-plus': {

name: 'qwen-plus',

type: 'qwen',

endpoint: 'https://dashscope.aliyuncs.com/compatible-mode/v1/chat/completions',

apiKey: 'sk-6098629a2074b2792357a1'

},

'qwen-turbo': {

name: 'qwen-coder-turbo-latest',

type: 'qwen',

endpoint: 'https://dashscope.aliyuncs.com/compatible-mode/v1/chat/completions',

apiKey: 'sk-6098750241629a2074b2792357a1'

},

'qwen-max': {

name: 'qwen-coder-turbo-0919',

type: 'qwen',

endpoint: 'https://dashscope.aliyuncs.com/compatible-mode/v1/chat/completions',

apiKey: 'sk-609875241629a2074b2792357a1'

},

'gpt-4o-mini': {

name: 'gpt-4o-mini',

type: 'openai'

}

};

function openAIInput() {

const modelSelect = document.getElementById('modelSelect');

modelSelect.innerHTML = '<option value="">选择AI模型</option>';

Object.keys(AI\_MODELS).forEach(modelKey => {

const option = document.createElement('option');

option.value = modelKey;

option.textContent = AI\_MODELS[modelKey].name;

modelSelect.appendChild(option);

});

document.getElementById('aiInputContainer').style.display = 'block';

}

async function generateAIResponse() {

const userInput = document.getElementById('aiUserInput').value;

const selectedModelKey = document.getElementById('modelSelect').value;

if (userInput.trim() === "") {

alert('请输入问题');

return;

}

if (!selectedModelKey) {

alert('请选择AI模型');

return;

}

const model = AI\_MODELS[selectedModelKey];

try {

if (model.type === 'qwen') {

const requestBody = {

model: selectedModelKey,

messages: [

{ role: "system", content: "你是一个有帮助的助手。" },

{ role: "user", content: userInput }

]

};

const response = await fetch(model.endpoint, {

method: 'POST',

headers: {

'Content-Type': 'application/json; charset=utf-8',

'Authorization': 'Bearer sk-6098750b24422792357a1'

},

body: JSON.stringify(requestBody)

});

if (response.ok) {

const data = await response.json();

const aiResponse = data.choices[0].message.content;

document.getElementById('message').value = aiResponse;

closeAIInput();

} else {

const errorData = await response.text();

throw new Error(`通义千问API调用失败：${errorData}`);

}

}

// 处理其他模型类型（此处省略部分代码，如OpenAI模型处理逻辑）

} catch (error) {

console.error('AI响应生成错误', error);

alert(`发生错误：${error.message}`);

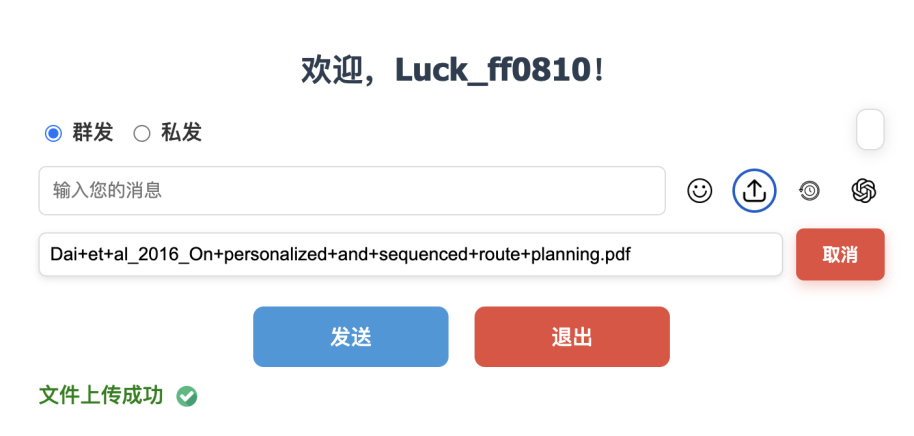
}

}

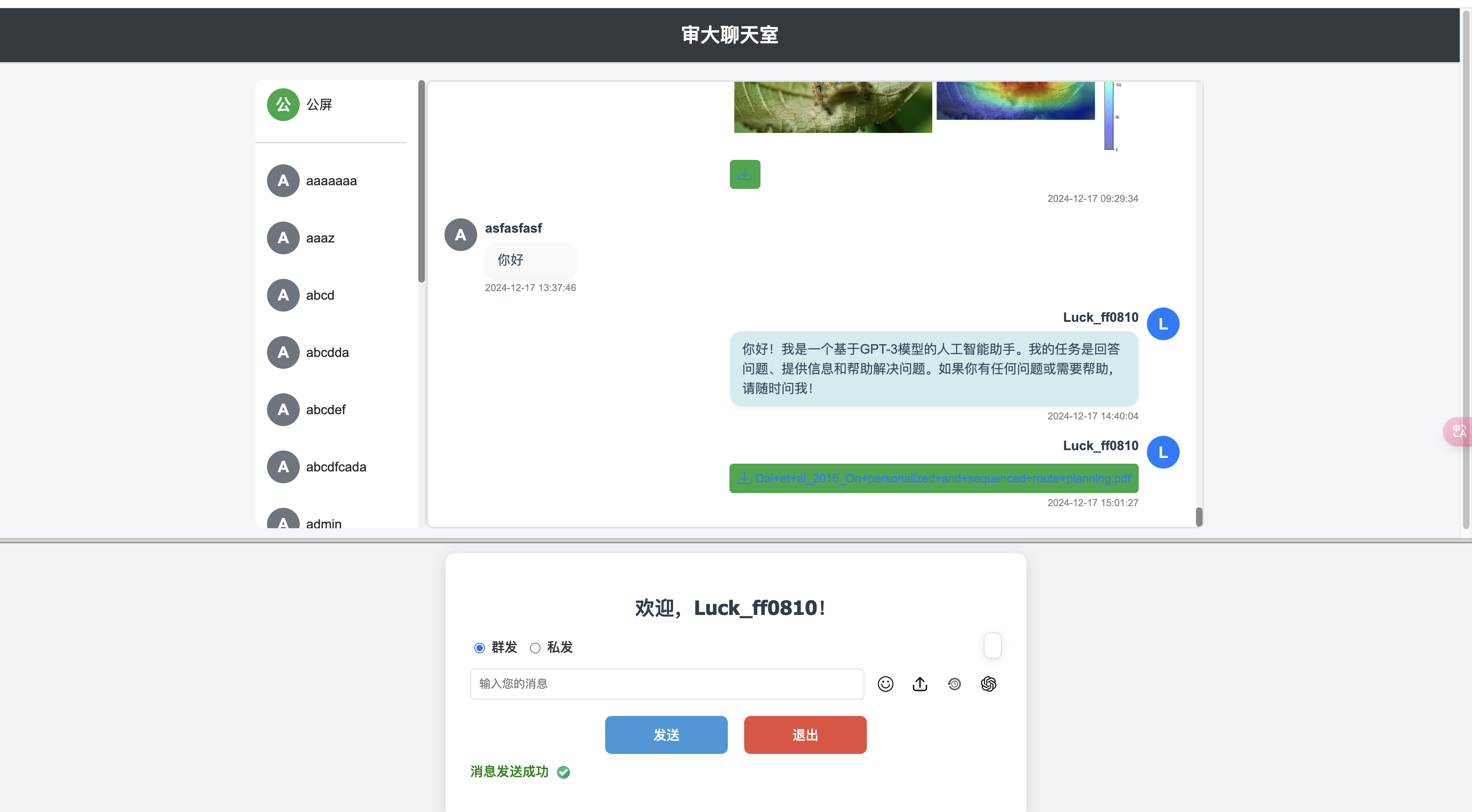
**文件传输和下载**

最终效果图：

**上传本地的pdf文件**

****

**单独发送一个pdf文件，并且正确发送到数据库中**

****

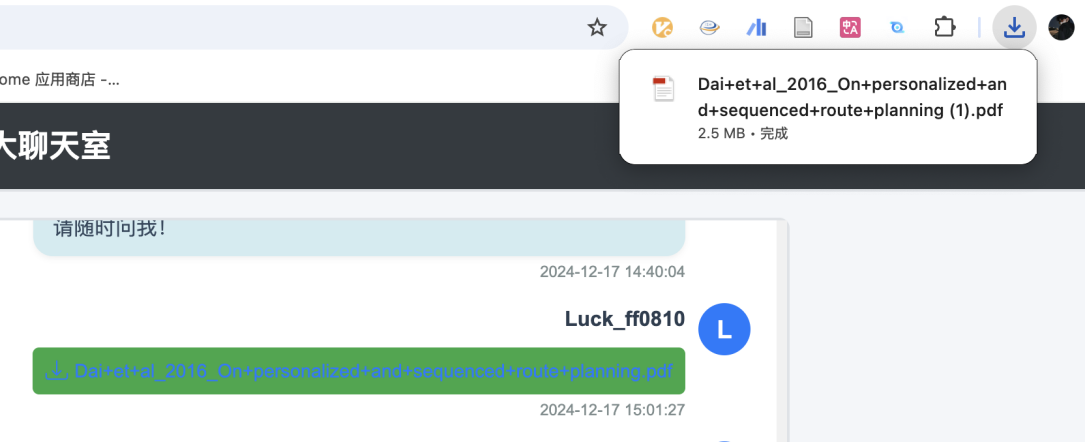
**让文件和消息同时发送**

****

**同时发送文件和消息**

****

**下载聊天记录中的文件**

****

**实现思路：**

首先是获取请求参数action，判断其值是否为"uploadFile"，以此来区分当前请求是进行文件上传操作还是普通消息发送操作（在这段代码中，这里虽然区分了但主要聚焦在文件发送相关逻辑，消息发送部分涉及的敏感词过滤等更多是附带处理）。如果是文件上传相关操作，就调用handleFileUpload方法来处理文件上传逻辑；如果不是，则可能是其他类型操作（此处代码逻辑后续未完整展示其他分支处理情况）。

然后是文件上传处理（handleFileUpload方法内）

1.文件存在性验证：

从请求中获取文件部分（Part对象），若获取到的文件对象为null或者文件大小小于等于 0，意味着用户没有选择要上传的文件，此时向客户端返回400 Bad Request状态码，并提示 “未选择文件！”，结束该方法执行。

String action = request.getParameter("action"); if ("uploadFile".equals(action)) { handleFileUpload(request, response, user); } else { handleMessageSend(request, response, user); }

2.文件类型验证：

获取文件的原始文件名、文件类型（通过Part对象的相关方法获取）以及文件大小。接着定义一个允许的文件类型数组，使用Arrays.asList将数组转换为列表后，通过contains方法检查获取到的文件类型是否在允许列表中。若文件类型不在允许列表内，向客户端返回400 Bad Request状态码，并提示 “不支持的文件类型！”，终止方法执行。

String[] allowedTypes = { "image/png", "image/jpeg", "application/pdf", "application/msword", "application/vnd.openxmlformats-officedocument.wordprocessingml.document", "application/vnd.ms-excel", "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet", "text/csv", "text/plain", "application/x-python-code", "application/vnd.ms-powerpoint", "application/vnd.openxmlformats-officedocument.presentationml.presentation", "application/zip", "application/x-rar-compressed", "image/gif", "video/mp4" }; boolean isAllowed = Arrays.asList(allowedTypes).contains(fileType); if (!isAllowed) { response.setStatus(HttpServletResponse.SC\_BAD\_REQUEST); response.getWriter().println("不支持的文件类型！"); return; }

3.文件大小验证：

设定最大允许文件大小（这里为 60MB），将获取到的文件大小与最大允许大小进行比较，如果文件大小超过了限制，同样向客户端返回400 Bad Request状态码，并提示 “文件大小超过 60MB 限制！”，不再继续后续流程。

long maxSize = 60 \* 1024 \* 1024; // 60MB if (fileSize > maxSize) { response.setStatus(HttpServletResponse.SC\_BAD\_REQUEST); response.getWriter().println("文件大小超过60MB限制！"); return; }

4.文件数据读取及数据库插入准备：

若文件通过了上述类型和大小验证，获取文件的输入流，并将其内容读取为字节数组。然后建立与数据库的连接（使用DriverManager.getConnection方法并传入配置好的数据库连接 URL、用户名和密码），构建一个用于向Message表插入数据的SQL语句，通过PreparedStatement预编译该语句，设置相应的参数（如用户名、消息内容这里为空字符串、当前日期时间、接收者为 “everyone”、文件字节数据、文件名、文件类型等），准备将文件相关信息插入数据库。

InputStream fileContent = filePart.getInputStream(); byte[] fileData = fileContent.readAllBytes(); try (Connection connection = DriverManager.getConnection(DB\_URL, DB\_USER, DB\_PASSWORD)) { String sql = "INSERT INTO Message(username, message, date, receiver, file\_data, file\_name, file\_type) VALUES (?,?,?,?,?,?,?)"; PreparedStatement pstmt = connection.prepareStatement(sql); pstmt.setString(1, user); pstmt.setString(2, ""); // 消息为空 pstmt.setString(3, new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new Date())); pstmt.setString(4, "everyone"); pstmt.setBytes(5, fileData); pstmt.setString(6, fileName); pstmt.setString(7, fileType); //... 后续执行插入操作等 }

5.数据库插入操作及结果反馈：

执行PreparedStatement的executeUpdate方法执行插入操作，根据受影响的行数判断插入是否成功。如果插入成功（受影响行数大于 0），向客户端返回200 OK状态码，并提示 “文件上传成功”；若插入失败，返回500 Internal Server Error状态码，并提示 “文件上传失败”。

int rows = pstmt.executeUpdate(); if (rows > 0) { response.setStatus(HttpServletResponse.SC\_OK); response.getWriter().println("文件上传成功"); } else { response.setStatus(HttpServletResponse.SC\_INTERNAL\_SERVER\_ERROR); response.getWriter().println("文件上传失败"); }

**屏蔽词过滤**

最终效果图：

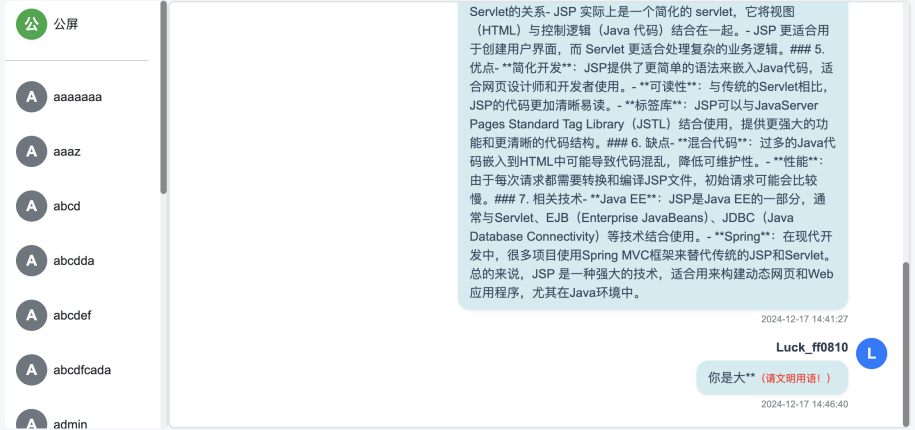
屏蔽词列表：

****

**输入带有屏蔽词的内容并发送：**

****

**发送后在聊天记录中显示为过滤后的记录，同时带有颜色标注的提醒。**

****

**实现思路：**在消息发送的相关处理逻辑中，当用户输入消息内容后，会通过与数据库中存储的敏感词列表进行比对，若消息中包含敏感词，则将敏感词替换为相应数量的 “\*” 字符进行屏蔽，同时若有敏感词被替换，还会在消息末尾添加提示语提醒用户文明用语，最终返回过滤后的消息内容用于后续的插入数据库等操作。

### **关键代码及步骤**

#### 敏感词数据获取：在filterSensitiveWords方法中，首先会构建一个 SQL 查询语句，目的是从名为SensitiveWords的数据库表中获取所有的敏感词记录。然后使用PreparedStatement预编译该查询语句，并执行查询操作获取结果集ResultSet。

String query = "SELECT word FROM SensitiveWords";

PreparedStatement stmt = null;

ResultSet rs = null;

try {

stmt = conn.prepareStatement(query);

rs = stmt.executeQuery();

while (rs.next()) {

sensitiveWords.add(rs.getString("word"));

}

} finally {

if (rs!= null) try { rs.close(); } catch (SQLException e) { }

if (stmt!= null) try { stmt.close(); } catch (SQLException e) { }

}

#### 敏感词比对与替换：在获取到敏感词列表后，通过循环遍历该列表中的每个敏感词，使用contains方法检查用户输入的消息（message变量）中是否包含当前敏感词。若包含，则设置一个标志位flag为true，表示消息中存在敏感词需要处理；接着创建一个StringBuilder对象，根据敏感词的长度生成对应数量的 “” 字符作为替换内容，然后使用replace方法将消息中的敏感词替换为这些 “” 字符，实现屏蔽效果。

boolean flag = false;

for (String word : sensitiveWords) {

if (message.contains(word)) {

flag = true;

StringBuilder replacement = new StringBuilder();

for (int i = 0; i < word.length(); i++) {

replacement.append("\*");

}

message = message.trim();

message = message.replace(word, replacement.toString());

}

}

#### 添加提示语（若有敏感词被替换）：在完成敏感词的替换后，再次检查标志位flag，如果flag为true，意味着消息中原本存在敏感词且已经进行了替换操作，此时会在消息末尾添加一段带有样式设置的提示语 “（请文明用语！）”，提醒用户要文明发言，然后返回经过处理后的消息内容。

if (flag) {

message += "<span style='font-size: small; color: red;'>（请文明用语！）</span>";

}

return message;

4.调用过滤方法的位置：在handleMessageSend方法中，当获取到用户输入的消息内容（message变量）后，会判断消息不为空时，调用filterSensitiveWords方法对消息进行敏感词过滤处理，将过滤后的消息再继续用于后续的数据库插入等操作流程。

// 过滤敏感词

if (message!= null &&!message.isEmpty()) {

message = filterSensitiveWords(connection, message);

}

**管理员界面设计**

最终效果图：

管理员登陆后的界面：可以看到之前别人发的消息

****

**管理员端实现对用户信息的维护**

****

**对用户密码的维护**

****

**对聊天记录管理的维护**

****

**实现查询相关聊天记录中的敏感词的记录**

****

**布局设计：**

整体上，此聊天室维护界面旨在为具有相应权限的用户（通过登录验证确保）提供便捷且集中的功能操作入口，方便管理与维护聊天室相关的各类事务。界面布局上，采用了经典的顶部导航栏搭配主体内容区域的结构。顶部导航栏利用 Bootstrap 的navbar组件进行构建，使其具备响应式特点，适应不同屏幕尺寸。在导航栏内设置了多个功能链接，分别对应如用户信息修改、密码修改、聊天记录管理以及敏感词维护等关键功能，清晰地划分出不同维护操作的入口，方便用户快速定位并选择所需功能。同时设置 “返回聊天室” 按钮，便于随时返回聊天界面。主体内容区域通过flex布局实现居中展示，内部的半透明卡片式容器包裹iframe元素，用于承载各个具体维护功能对应的 JSP 页面，以一种简洁又直观的方式来展示不同功能模块的内容。

在交互设计方面，重点在于实现页面切换功能以及视觉反馈。通过 JavaScript 编写的loadPage函数，响应用户点击导航栏链接的操作，实现iframe中页面的动态切换，让用户能流畅地在不同维护功能页面间进行转换。并且在切换页面时，合理地处理导航栏链接的激活状态显示，使得用户能清晰知晓当前处于哪个功能页面下。另外，还运用了丰富的 CSS 样式来美化界面，从整体的背景渐变效果，到各个元素如导航栏、按钮、iframe等的颜色、阴影、圆角、悬停效果等细节样式设置，全方位提升界面的美观度与易用性，增强用户使用体验。

关键代码：  
 顶部导航栏及各功能链接的定义

<nav class="navbar navbar-expand-lg navbar-light bg-white">

<div class="container-fluid">

<a class="navbar-brand" href="#">聊天室维护</a>

<button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarMaintenance"

aria-controls="navbarMaintenance" aria-expanded="false" aria-label="Toggle navigation">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="navbarMaintenance">

<ul class="navbar-nav me-auto mb-2 mb-lg-0">

<!-- 修改用户的基本信息 -->

<li class="nav-item">

<a class="nav-link active" aria-current="page" onclick="loadPage('UserInfoMaintenance.jsp')">修改用户信息</a>

</li>

<!-- 修改用户密码 -->

<li class="nav-item">

<a class="nav-link" onclick="loadPage('ChangePassword.jsp')">修改用户密码</a>

</li>

<!-- 聊天记录管理（含导出功能） -->

<li class="nav-item">

<a class="nav-link" onclick="loadPage('ChatRecordManagement.jsp')">聊天记录管理</a>

</li>

<!-- 敏感词维护（添加、删除、修改） -->

<li class="nav-item">

<a class="nav-link" onclick="loadPage('SensitiveWordsMaintenance.jsp')">敏感词维护</a>

</li>

</ul>

<!-- 返回聊天室按钮 -->

<form class="d-flex">

<button type="button" class="return-button" onclick="location.href='AdminChat.jsp'">返回聊天室</button>

</form>

</div>

</div>

</nav>

展示不同维护页面的iframe元素定义

<div class="content-container">

<div class="card-container">

<!-- 使用iframe显示对应的维护界面 -->

<iframe id="maintenanceFrame" src="UserInfoMaintenance.jsp"></iframe>

</div>

</div>

实现页面切换以及导航栏链接激活状态管理的核心loadPage函数

function loadPage(pageUrl) {

var frame = document.getElementById('maintenanceFrame');

frame.src = pageUrl;

// 移除之前的active状态

var links = document.querySelectorAll('.navbar-nav .nav-link');

links.forEach(function(link) {

link.classList.remove('active');

});

// 给当前点击的链接添加active状态

event.target.classList.add('active');

}

**基本信息维护实现思路：**

初次加载页面时会先尝试从请求参数中获取当前选中的用户（selectedUser）以及操作类型（action）等关键信息。若操作类型为 “update”，意味着是提交了用户信息更新请求，此时会收集表单中除密码外各个字段的新值，构建 UPDATE 语句并执行数据库更新操作，根据更新结果反馈相应提示信息。而在页面展示过程中，当有选中的用户（selectedUser 不为空）时，会通过查询数据库获取该用户除密码外的详细信息，填充到右侧的表单各对应字段中进行展示，方便查看与后续编辑。另外，用户列表通过查询数据库获取所有用户的用户名和姓名信息并动态生成表格展示，点击列表中的用户行可触发 selectUser 函数，实现选中用户并刷新页面展示对应用户详情的效果，整个过程围绕数据库交互与页面展示更新来实现用户基本信息的维护管理。

#### 关键代码： 获取页面相关参数及处理更新操作：

request.setCharacterEncoding("UTF-8");

String selectedUser = request.getParameter("selectedUser");

String action = request.getParameter("action");

// 其他获取表单字段的代码省略

if ("update".equals(action)) {

selectedUser = request.getParameter("username");

Connection updateConn = null;

PreparedStatement updatePstmt = null;

try {

Security.setProperty("jdk.tls.disabledAlgorithms", "");

System.setProperty("https.protocols", "TLSv1");

Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");

updateConn = DriverManager.getConnection(

"jdbc:sqlserver://10.211.55.7:1433;databaseName=webChat;" +

"encrypt=true;trustServerCertificate=true;sslProtocol=TLSv1;" +

"disableStatementPooling=true;cancelQueryTimeout=0;socketTimeout=120",

"sa", "123456");

// 构建UPDATE语句

String updateSql = "UPDATE webChat.dbo.UserInfo SET gender=?, dname=?, province\_city\_district=?," +

" person\_name=?, tel=?, postcode=?, bpcode=?, fax=?, hand=?, IDcard=?, email=?," +

" homepg=?, quest=?, answ=? WHERE username=?";

updatePstmt = updateConn.prepareStatement(updateSql);

updatePstmt.setString(1, gender);

updatePstmt.setString(2, dname);

updatePstmt.setString(3, province\_city\_district);

updatePstmt.setString(4, person\_name);

updatePstmt.setString(5, tel);

updatePstmt.setString(6, postcode);

updatePstmt.setString(7, bpcode);

updatePstmt.setString(8, fax);

updatePstmt.setString(9, hand);

updatePstmt.setString(10, IDcard);

updatePstmt.setString(11, email);

updatePstmt.setString(12, homepg);

updatePstmt.setString(13, quest);

updatePstmt.setString(14, answ);

updatePstmt.setString(15, selectedUser);

int rows = updatePstmt.executeUpdate();

if (rows > 0) {

message = "用户信息更新成功！";

} else {

message = "用户信息更新失败，请确认用户是否存在。";

}

} catch (Exception e) {

e.printStackTrace();

message = "发生错误：" + e.getMessage();

} finally {

if (updatePstmt!= null) {

try { updatePstmt.close(); } catch (SQLException e) { e.printStackTrace(); }

}

if (updateConn!= null) {

try { updateConn.close(); } catch (SQLException e) { e.printStackTrace(); }

}

}

}

#### 根据选中用户加载用户信息到表单：

if (selectedUser!= null &&!selectedUser.isEmpty()) {

Connection userConn = null;

PreparedStatement userPstmt = null;

ResultSet userRs = null;

try {

Security.setProperty("jdk.tls.disabledAlgorithms", "");

System.setProperty("https.protocols", "TLSv1");

Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");

userConn = DriverManager.getConnection(

"jdbc:sqlserver://10.211.55.7:1433;databaseName=webChat;" +

"encrypt=true;trustServerCertificate=true;sslProtocol=TLSv1;" +

"disableStatementPooling=true;cancelQueryTimeout=0;socketTimeout=120",

"sa", "123456");

String userSql = "SELECT username, gender, dname, province\_city\_district, person\_name," +

" tel, postcode, bpcode, fax, hand, IDcard, email, homepg, quest, answ" +

" FROM webChat.dbo.UserInfo WHERE username=?";

userPstmt = userConn.prepareStatement(userSql);

userPstmt.setString(1, selectedUser);

userRs = userPstmt.executeQuery();

if (userRs.next()) {

db\_gender = userRs.getString("gender");

db\_dname = userRs.getString("dname");

db\_province\_city\_district = userRs.getString("province\_city\_district");

db\_person\_name = userRs.getString("person\_name");

db\_tel = userRs.getString("tel");

db\_postcode = userRs.getString("postcode");

db\_bpcode = userRs.getString("bpcode");

db\_fax = userRs.getString("fax");

db\_hand = userRs.getString("hand");

db\_IDcard = userRs.getString("IDcard");

db\_email = userRs.getString("email");

db\_homepg = userRs.getString("homepg");

db\_quest = userRs.getString("quest");

db\_answ = userRs.getString("answ");

}

} catch (Exception e) {

e.printStackTrace();

message = "加载用户信息时出错：" + e.getMessage();

} finally {

if (userRs!= null) {

try { userRs.close(); } catch (SQLException e) { e.printStackTrace(); }

}

if (userPstmt!= null) {

try { userPstmt.close(); } catch (SQLException e) { e.printStackTrace(); }

}

if (userConn!= null) {

try { userConn.close(); } catch (SQLException e) { e.printStackTrace(); }

}

}

}

#### 展示用户列表

Connection listConn = null;

Statement listStmt = null;

ResultSet listRs = null;

try {

Security.setProperty("jdk.tls.disabledAlgorithms", "");

System.setProperty("https.protocols", "TLSv1");

Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");

listConn = DriverManager.getConnection(

"jdbc:sqlserver://10.211.55.7:1433;databaseName=webChat;" +

"encrypt=true;trustServerCertificate=true;sslProtocol=TLSv1;" +

"disableStatementPooling=true;cancelQueryTimeout=0;socketTimeout=120",

"sa", "123456");

String listSql = "SELECT username, person\_name FROM webChat.dbo.UserInfo ORDER BY username ASC";

listStmt = listConn.createStatement();

listRs = listStmt.executeQuery(listSql);

%>

<table class="table table-sm table-hover">

<thead>

<tr>

<th>用户名</th>

<th>姓名</th>

</tr>

</thead>

<tbody>

<%

while (listRs.next()) {

String uName = listRs.getString("username");

String pName = listRs.getString("person\_name");

%>

<tr onclick="selectUser('<%= uName %>')"

style="cursor: pointer;<%= (selectedUser!= null && selectedUser.equals(uName))? "background-color:#e9ecef;" : "" %>">

<td><%= uName %></td>

<td><%= pName!= null? pName : "" %></td>

</tr>

<%

}

%>

</tbody>

</table>

<%

} catch(Exception e) {

e.printStackTrace();

} finally {

if (listRs!= null) try { listRs.close(); } catch (SQLException e) {}

if (listStmt!= null) try { listStmt.close(); } catch (SQLException e) {}

if (listConn!= null) try { listConn.close(); } catch (SQLException e) {}

}

**聊天记录维护实现思路：**

首先，页面加载时获取相关请求参数，根据不同操作（如搜索、删除、导出等）执行对应逻辑。搜索时，依据聊天对象、日期范围、是否含敏感词等条件构建数据库查询语句，获取符合条件的聊天记录展示在表格中。删除操作会根据消息 ID 删除对应记录并刷新页面。导出功能可按选择的格式（如 TXT、CSV、Excel 等）导出所选聊天记录，整体通过与数据库交互及前端交互实现功能。

页面布局上通过表单提供查询筛选条件输入，用表格展示聊天记录详情及操作按钮，搭配样式和脚本实现交互效果，如展开收起消息内容、全选记录、确认删除提示等，提升用户操作体验。

关键代码：

#### 查询记录构建查询条件及执行查询部分

List<String> queryConditions = new ArrayList<>();

List<Object> params = new ArrayList<>();

if (sender!= null &&!sender.isEmpty()) {

queryConditions.add("(username =? OR receiver =? OR receiver = 'everyone')");

params.add(sender);

params.add(sender);

}

if (startDate!= null &&!startDate.isEmpty()) {

queryConditions.add("date >=?");

params.add(startDate);

}

if ("true".equals(includeSensitive)) {

queryConditions.add("message LIKE?");

params.add("%<span style='font-size: small; color: red;'>（请文明用语！）</span>");

}

String whereClause = queryConditions.isEmpty()? "1=1" : String.join(" AND ", queryConditions);

String query = "SELECT message\_id, username, message, receiver, date, file\_name FROM Message WHERE " + whereClause + " ORDER BY date DESC";

PreparedStatement pstmt = conn.prepareStatement(query);

for (int i = 0; i < params.size(); i++) {

pstmt.setObject(i + 1, params.get(i));

}

ResultSet rs = pstmt.executeQuery();

#### 全选功能的 JavaScript 代码

document.getElementById('selectAll').addEventListener('change', function() {

const checkboxes = document.querySelectorAll('input[name="selectedRecords"]');

checkboxes.forEach(checkbox => {

checkbox.checked = this.checked;

});

});

**敏感词表维护实现思路：**

页面加载时先获取请求中的操作参数（action）来判断执行何种功能逻辑，比如添加操作时会获取输入的新敏感词插入数据库；删除操作依据指定的要删除敏感词从数据库移除相应记录；编辑操作则准备好对应表单展示原敏感词方便修改，更新操作将修改后的敏感词替换原有敏感词存到数据库。同时页面展示已有敏感词列表，并为每个敏感词提供编辑、删除的操作按钮，实现交互功能，操作后还会根据结果给出相应提示信息。

添加敏感词操作

if ("add".equals(action)) {

if (newWord!= null &&!newWord.trim().isEmpty()) {

Connection conn = null;

PreparedStatement pstmt = null;

try {

conn = DriverManager.getConnection(

"jdbc:sqlserver://10.211.55.7:1433;databaseName=webChat;" +

"encrypt=true;trustServerCertificate=true;sslProtocol=TLSv1;" +

"disableStatementPooling=true;cancelQueryTimeout=0;socketTimeout=120",

"sa", "123456");

String sql = "INSERT INTO webChat.dbo.SensitiveWords (word) VALUES (?)";

pstmt = conn.prepareStatement(sql);

pstmt.setString(1, newWord.trim());

int rows = pstmt.executeUpdate();

if (rows > 0) {

message = "敏感词添加成功！";

} else {

message = "敏感词添加失败！";

}

} catch (Exception e) {

e.printStackTrace();

message = "发生错误：" + e.getMessage();

} finally {

if (pstmt!= null) try {pstmt.close();} catch(SQLException e){}

if (conn!= null) try {conn.close();} catch(SQLException e){}

}

} else {

message = "请输入要添加的敏感词！";

}

}

删除敏感词操作

if ("delete".equals(action)) {

oldWord = request.getParameter("oldWord");

if (oldWord!= null &&!oldWord.trim().isEmpty()) {

Connection conn = null;

PreparedStatement pstmt = null;

try {

conn = DriverManager.getConnection(

"jdbc:sqlserver://10.211.55.7:1433;databaseName=webChat;" +

"encrypt=true;trustServerCertificate=true;sslProtocol=TLSv1;" +

"disableStatementPooling=true;cancelQueryTimeout=0;socketTimeout=120",

"sa", "123456");

String sql = "DELETE FROM webChat.dbo.SensitiveWords WHERE word=?";

pstmt = conn.prepareStatement(sql);

pstmt.setString(1, oldWord.trim());

int rows = pstmt.executeUpdate();

if (rows > 0) {

message = "敏感词删除成功！";

} else {

message = "敏感词删除失败，未找到该词。";

}

} catch (Exception e) {

e.printStackTrace();

message = "发生错误：" + e.getMessage();

} finally {

if (pstmt!= null) try {pstmt.close();} catch(SQLException e){}

if (conn!= null) try {conn.close();} catch(SQLException e){}

}

} else {

message = "未指定要删除的敏感词！";

}

}