**目录**

[一、引言 3](#_Toc187681351)

[二、需求分析 3](#_Toc187681352)

[1.今日头条用户部分主要包括以下功能模块 3](#_Toc187681353)

[2.系统处理的主要数据对象是用户和作品 3](#_Toc187681354)

[3.涉及的数据 3](#_Toc187681355)

[三、数据库和功能概念模型 4](#_Toc187681356)

[1.数据库部分设计的概念模型如下图所示 4](#_Toc187681357)

[2.功能部分设计的概念模型如下图所示： 5](#_Toc187681358)

[四、技术栈与核心组件 5](#_Toc187681359)

[五、数据库代码实现 6](#_Toc187681360)

[1.AT\_USERS\_DESC表(用户表) 6](#_Toc187681361)

[2.AT\_USERS\_COLLECT表(用户收藏表) 7](#_Toc187681362)

[3.AT\_USERS\_COMMENT表(用户评论表) 8](#_Toc187681363)

[4.AT\_USERS\_MESS表(用户私信表) 9](#_Toc187681364)

[5.AT\_WORK表(作品表) 10](#_Toc187681365)

[六、后端整体框架图 11](#_Toc187681366)

[七、后端代码实现 12](#_Toc187681367)

[1.后端整体目录 12](#_Toc187681368)

[2.执行流程 12](#_Toc187681369)

[3.Interceptor（拦截器） 14](#_Toc187681370)

[4.Controller（控制器） 15](#_Toc187681371)

[5.Service（服务层） 32](#_Toc187681372)

[6.Pojo（实体类） 38](#_Toc187681373)

[7.Mapper（数据访问层） 44](#_Toc187681374)

[八、前端代码实现和页面展示 46](#_Toc187681375)

[1.前端整体目录 46](#_Toc187681376)

[2.跳转页面（index.html） 47](#_Toc187681377)

[3.首页抬头（head\_include.html） 49](#_Toc187681378)

[4.登录和注册（userLogin.html和userRegister.html） 53](#_Toc187681379)

[5.用户部分 60](#_Toc187681380)

[6.功能部分（以作品（work）相关代码为例，其余功能前端代码与work类似） 69](#_Toc187681381)

[九、相关功能展示 79](#_Toc187681382)

[1.注册 79](#_Toc187681383)

[2.作品（还是以作品为例，剩下的已经验收过了） 81](#_Toc187681384)

[十、结论 84](#_Toc187681385)

# 一、引言

随着互联网技术的飞速发展，信息传播的方式日益多样化。今日头条作为一款广受欢迎的新闻资讯类应用，其背后复杂而高效的数据库系统是支撑其庞大用户群体和海量信息的关键。本课程设计旨在通过模拟今日头条用户部分的功能，深入探究数据库原理在实际应用中的运用，增强数据库设计、实现与管理能力。通过对用户注册、作品发布、收藏、评论及私信交流等功能的实现，全面展示数据库技术在现代互联网应用中的重要作用。

# 二、需求分析

## 1.今日头条用户部分主要包括以下功能模块

（1）用户注册

（2）用户发表作品、修改作品、删除作品，注意存储细节数据，比如作品的展现量、发表时间等

（3）用户可以收藏别人的作品

（4）用户可以评论别人的作品

（5）用户和用户之间可以私信交流

## 2.系统处理的主要数据对象是用户和作品

和用户相关的信息有：用户的基本信息

和作品相关的信息有：作品的基本信息、作品的展现量

和二者相关的信息有：用户收藏作品、用户评论作品、用户私信交流的信息

## 3.涉及的数据

用户表：用户编号、用户名、密码、性别、注册时间、联系方式等

作品表：作品编号、用户编号、作品标题、内容、发表时间、修改时间、展现量等

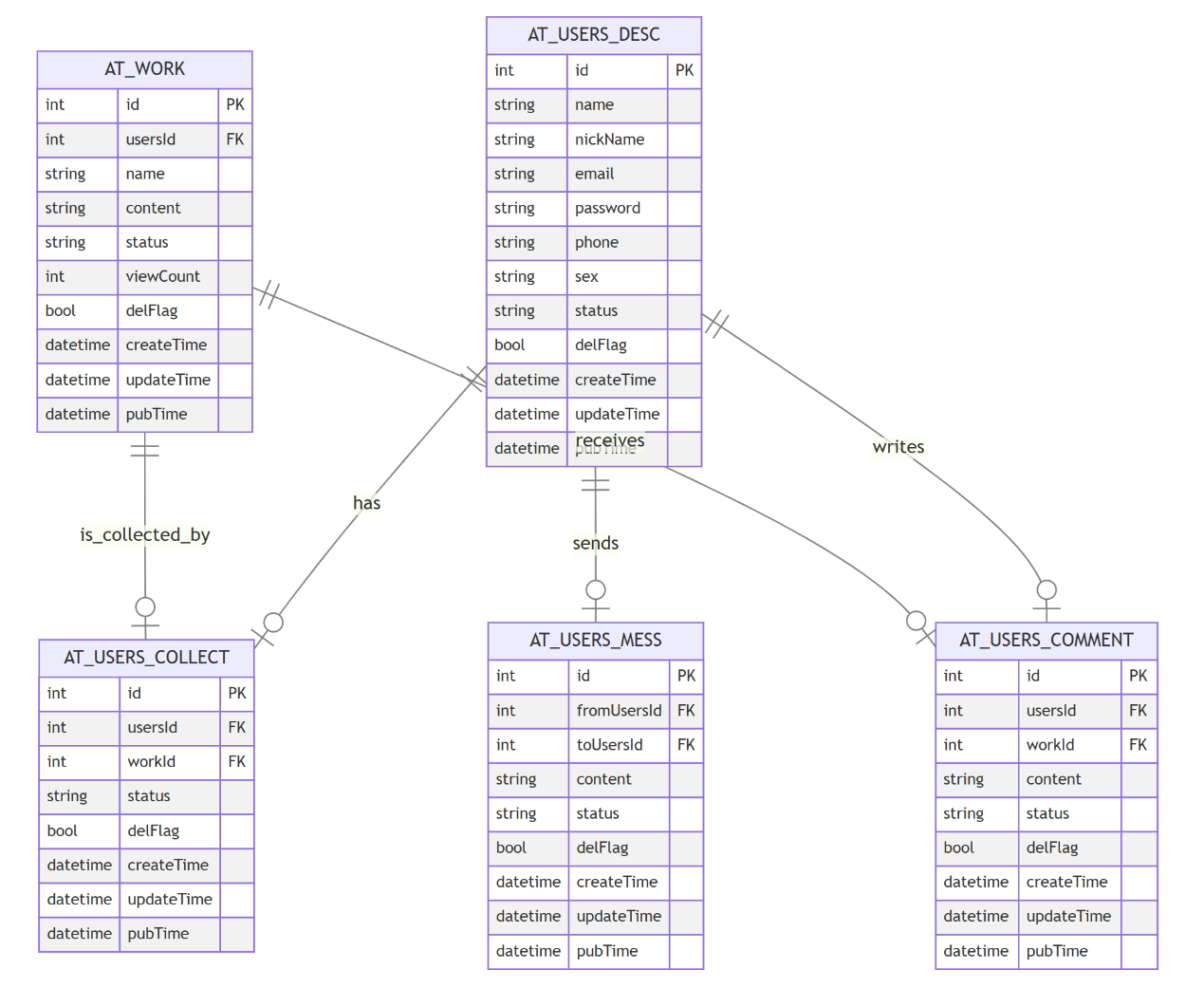
收藏表：收藏编号、用户编号、作品编号、收藏时间等

评论表：评论编号、作品编号、用户编号、评论内容、评论时间等

私信表：私信编号、发送者编号、接收者编号、消息内容、发送时间等

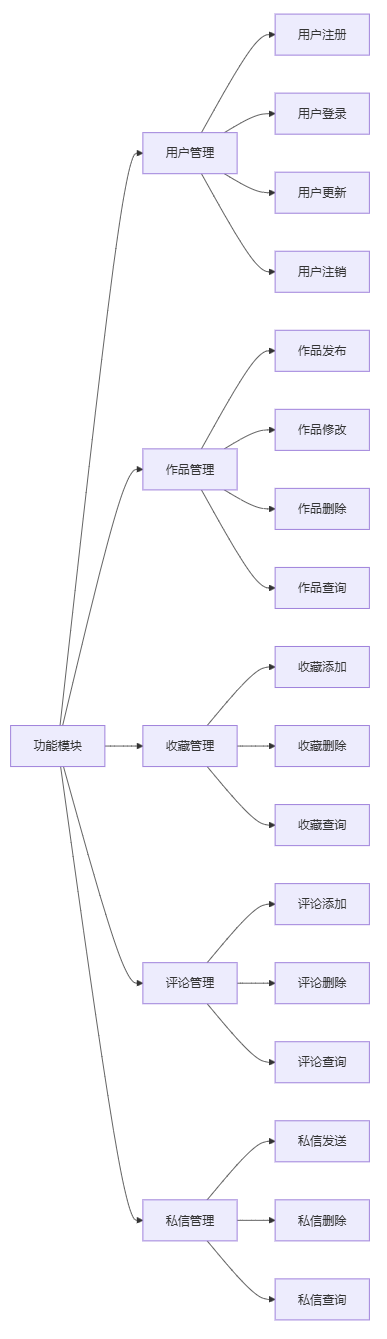
# 三、数据库和功能概念模型

## 1.数据库部分设计的概念模型如下图所示



用户部分设计的的概念模型（E-R模型）

## 2.功能部分设计的概念模型如下图所示：



# 四、技术栈与核心组件

（1）后端框架：Spring Boot

①DispatcherServlet：接收HTTP请求

②Interceptor：处理身份验证和日志记录

③Controller：处理业务请求

④Service：执行业务逻辑

⑤Mapper：数据库交互

（2）数据库：MySQL

①ORM框架：MyBatis Plus

②CRUD操作：简化数据库操作

③分页查询：优化数据展示

（3）缓存：Redis

缓存管理：提高应用性能

（4）前端技术：Thymeleaf

（5）安全策略

①JWT：身份认证和授权

②HTTPS：数据传输加密

③CSRF防护：防止跨站请求伪造

④SQL注入防护：防止SQL注入攻击

# 五、数据库代码实现

create database todayHead;

use todayHead;

SET NAMES utf8mb4;

SET FOREIGN\_KEY\_CHECKS = 0;

## 1.AT\_USERS\_DESC表(用户表)

DROP TABLE IF EXISTS `at\_users\_desc`;

CREATE TABLE `at\_users\_desc` (

`id` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NOT NULL COMMENT '主键',

`name` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NULL DEFAULT NULL COMMENT '姓名',

`nickName` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NULL DEFAULT NULL COMMENT '昵称',

`email` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NULL DEFAULT NULL COMMENT '邮箱',

`password` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NULL DEFAULT NULL COMMENT '密码',

`content` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NULL DEFAULT NULL COMMENT '内容',

`phone` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NULL DEFAULT NULL COMMENT '手机号',

`sex` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NULL DEFAULT NULL COMMENT '性别:0:未知,1:男,1:女',

`status` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NULL DEFAULT NULL COMMENT '状态:(0:禁用,1:启用)',

`delFlag` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NULL DEFAULT NULL COMMENT '逻辑删除(0:已删除,1:未删除)',

`createTime` datetime NULL DEFAULT NULL COMMENT '创建时间',

`updateTime` datetime NULL DEFAULT NULL COMMENT '更新时间',

`pubTime` datetime NULL DEFAULT NULL COMMENT '发布时间(排序)',

`lastLoginTime` datetime NULL DEFAULT NULL COMMENT '上次登录时间',

PRIMARY KEY (`id`) USING BTREE

) ENGINE = InnoDB CHARACTER SET = utf8mb4 COLLATE = utf8mb4\_0900\_ai\_ci COMMENT = '用户表' ROW\_FORMAT = DYNAMIC;

INSERT INTO `at\_users\_desc` VALUES ('1740737903736152066', '姓名\_02112', '昵称\_02112', '22@22.com', 'sha256$VCF$c14217abe8869c4da6bd5b3a14eee9ba374b000e50782aa7210c759ad582463d', NULL, '手机112', '1', '1', '0', '2023-12-29 22:13:52', '2024-11-15 12:55:46', '2023-12-29 22:13:52', '2024-11-15 12:51:05');

INSERT INTO `at\_users\_desc` VALUES ('1742550907872137217', '姓名\_0133', '昵称\_0122', '11@11.com', 'sha256$6ti$138cce9278a873e1ae57c7bdae22aa81662c8ca8ab77904522ce690794a50f94', '<p>3333333</p>', '手机11', '0', '1', '0', '2024-01-03 22:18:06', '2024-11-15 12:50:54', '2024-01-03 22:18:06', '2024-11-15 12:50:54');

INSERT INTO `at\_users\_desc` VALUES ('1783513651609661442', '姓名\_04', '昵称\_04', '44@44.com', 'sha256$VCF$c14217abe8869c4da6bd5b3a14eee9ba374b000e50782aa7210c759ad582463d', '<p>4444</p>', '44', '2', '1', '0', '2024-04-25 23:09:25', '2024-04-29 11:04:34', '2024-04-25 23:09:25', '2024-04-29 11:04:34');

INSERT INTO `at\_users\_desc` VALUES ('1783513808199806977', '姓名\_05', '昵称\_05', '55@55.com', 'sha256$Hjz$6ea78702dfd8f46bb6113cf08f3857d0fa35abf1d4101336d3887f80307965b1', '<p>555</p>', '55', '0', '1', '0', '2024-04-25 23:10:03', '2024-04-29 11:10:10', '2024-04-25 23:10:03', NULL);

## 2.AT\_USERS\_COLLECT表(用户收藏表)

DROP TABLE IF EXISTS `at\_users\_collect`;

CREATE TABLE `at\_users\_collect` (

`id` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NOT NULL COMMENT '主键',

`usersId` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NULL DEFAULT NULL COMMENT '用户Id',

`workId` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NULL DEFAULT NULL COMMENT '作品Id',

`status` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NULL DEFAULT NULL COMMENT '状态:(0:启用,1:禁用)',

`delFlag` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NULL DEFAULT NULL COMMENT '逻辑删除(0:未删除,1:已删除)',

`createTime` datetime NULL DEFAULT NULL COMMENT '创建时间',

`updateTime` datetime NULL DEFAULT NULL COMMENT '更新时间',

`pubTime` datetime NULL DEFAULT NULL COMMENT '发布时间(排序)',

PRIMARY KEY (`id`) USING BTREE

) ENGINE = InnoDB CHARACTER SET = utf8mb4 COLLATE = utf8mb4\_0900\_ai\_ci COMMENT = '用户\_收藏表' ROW\_FORMAT = DYNAMIC;

INSERT INTO `at\_users\_collect` VALUES ('1857080126156607490', '1740737903736152066', '1856719831294873602', '0', '0', '2024-11-14 23:16:20', '2024-11-14 23:16:20', '2024-11-14 23:16:20');

INSERT INTO `at\_users\_collect` VALUES ('1857285557500882946', '1742550907872137217', '1856719831294873602', '0', '0', '2024-11-15 12:52:38', '2024-11-15 12:52:38', '2024-11-15 12:52:38');

## 3.AT\_USERS\_COMMENT表(用户评论表)

DROP TABLE IF EXISTS `at\_users\_comment`;

CREATE TABLE `at\_users\_comment` (

`id` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NOT NULL COMMENT '主键',

`usersId` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NULL DEFAULT NULL COMMENT '用户Id',

`workId` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NULL DEFAULT NULL COMMENT '作品Id',

`name` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NULL DEFAULT NULL COMMENT '名称',

`content` text CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NULL COMMENT '内容',

`status` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NULL DEFAULT NULL COMMENT '状态:(0:禁用,1:启用)',

`delFlag` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NULL DEFAULT NULL COMMENT '逻辑删除(0:未删除,1:已删除)',

`createTime` datetime NULL DEFAULT NULL COMMENT '创建时间',

`updateTime` datetime NULL DEFAULT NULL COMMENT '更新时间',

`pubTime` datetime NULL DEFAULT NULL COMMENT '发布时间(排序)',

PRIMARY KEY (`id`) USING BTREE

) ENGINE = InnoDB CHARACTER SET = utf8mb4 COLLATE = utf8mb4\_0900\_ai\_ci COMMENT = '用户\_评论表' ROW\_FORMAT = DYNAMIC;

INSERT INTO `at\_users\_comment` VALUES ('1857082676083761154', '1740737903736152066', '1856719831294873602', '不错', '<p>确实 写的不错</p>', '0', '0', '2024-11-14 23:26:28', '2024-11-14 23:26:28', '2024-11-14 23:26:28');

INSERT INTO `at\_users\_comment` VALUES ('1857086341783822338', NULL, '1856719831294873602', 'dddd', '<p>dddd</p>', '0', '0', '2024-11-14 23:41:02', '2024-11-14 23:41:02', '2024-11-14 23:41:02');

INSERT INTO `at\_users\_comment` VALUES ('1857086375761879041', NULL, '1856719831294873602', 'dddd', '<p>ddddffff</p>', '0', '0', '2024-11-14 23:41:10', '2024-11-14 23:41:10', '2024-11-14 23:41:10');

INSERT INTO `at\_users\_comment` VALUES ('1857086876742131713', '1742550907872137217', '1856719831294873602', '我也评论', '<p>我也评论我也评论我也评论我也评论<br/></p>', '0', '0', '2024-11-14 23:43:09', '2024-11-14 23:43:09', '2024-11-14 23:43:09');

INSERT INTO `at\_users\_comment` VALUES ('1857285697691299842', '1740737903736152066', '1857285440379138049', '确实不错啊,写的不错', '<p>确实不错啊,写的不错</p>', '0', '0', '2024-11-15 12:53:12', '2024-11-15 12:53:12', '2024-11-15 12:53:12');

## 4.AT\_USERS\_MESS表(用户私信表)

DROP TABLE IF EXISTS `at\_users\_mess`;

CREATE TABLE `at\_users\_mess` (

`id` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NOT NULL COMMENT '主键',

`fromUsersId` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NULL DEFAULT NULL COMMENT '用户Id(发送者)',

`toUsersId` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NULL DEFAULT NULL COMMENT '用户Id(接收者)',

`name` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NULL DEFAULT NULL COMMENT '名称',

`content` text CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NULL COMMENT '内容',

`status` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NULL DEFAULT NULL COMMENT '状态:(0:已录入,1:发布,2:审核通过,3:审核不通过)',

`delFlag` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NULL DEFAULT NULL COMMENT '逻辑删除(0:未删除,1:已删除)',

`createTime` datetime NULL DEFAULT NULL COMMENT '创建时间',

`updateTime` datetime NULL DEFAULT NULL COMMENT '更新时间',

`pubTime` datetime NULL DEFAULT NULL COMMENT '发布时间(排序)',

PRIMARY KEY (`id`) USING BTREE

) ENGINE = InnoDB CHARACTER SET = utf8mb4 COLLATE = utf8mb4\_0900\_ai\_ci COMMENT = '用户\_私信表' ROW\_FORMAT = DYNAMIC;

INSERT INTO `at\_users\_mess` VALUES ('1857274378858643458', '1740737903736152066', '1783513808199806977', '1111', '<p>1111</p>', '0', '0', '2024-11-15 12:08:13', '2024-11-15 12:08:13', '2024-11-15 12:08:13');

INSERT INTO `at\_users\_mess` VALUES ('1857275968596328449', '1740737903736152066', '1742550907872137217', '测试一下', '<p>测试一下<br/></p>', '0', '0', '2024-11-15 12:14:32', '2024-11-15 12:14:32', '2024-11-15 12:14:32');

INSERT INTO `at\_users\_mess` VALUES ('1857285990990589953', '1742550907872137217', '1740737903736152066', '你好.', '<p>你好.你好.你好.你好.<br/></p>', '0', '0', '2024-11-15 12:54:22', '2024-11-15 12:54:22', '2024-11-15 12:54:22');

INSERT INTO `at\_users\_mess` VALUES ('1857286269790171137', '1740737903736152066', '1783513651609661442', '222', '<p>2222222222222222</p>', '0', '0', '2024-11-15 12:55:28', '2024-11-15 12:55:28', '2024-11-15 12:55:28');

## 5.AT\_WORK表(作品表)

DROP TABLE IF EXISTS `at\_work`;

CREATE TABLE `at\_work` (

`id` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NOT NULL COMMENT '主键',

`usersId` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NULL DEFAULT NULL COMMENT '员工Id',

`name` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NULL DEFAULT NULL COMMENT '名称',

`content` text CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NULL COMMENT '内容',

`status` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NULL DEFAULT NULL COMMENT '状态:(0:启用,1:禁用)',

`viewCount` int NULL DEFAULT NULL COMMENT '展现量',

`delFlag` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci NULL DEFAULT NULL COMMENT '逻辑删除(0:未删除,1:已删除)',

`createTime` datetime NULL DEFAULT NULL COMMENT '创建时间',

`updateTime` datetime NULL DEFAULT NULL COMMENT '更新时间',

`pubTime` datetime NULL DEFAULT NULL COMMENT '发布时间(排序)',

PRIMARY KEY (`id`) USING BTREE

) ENGINE = InnoDB CHARACTER SET = utf8mb4 COLLATE = utf8mb4\_0900\_ai\_ci COMMENT = '作品表' ROW\_FORMAT = DYNAMIC;

INSERT INTO `at\_work` VALUES ('1856719479854141442', '1742550907872137217', '测试一下', '<p>测试一下<br/></p>', '1', 1, '0', '2024-11-13 23:23:15', '2024-11-14 23:07:20', '2024-11-13 23:23:15');

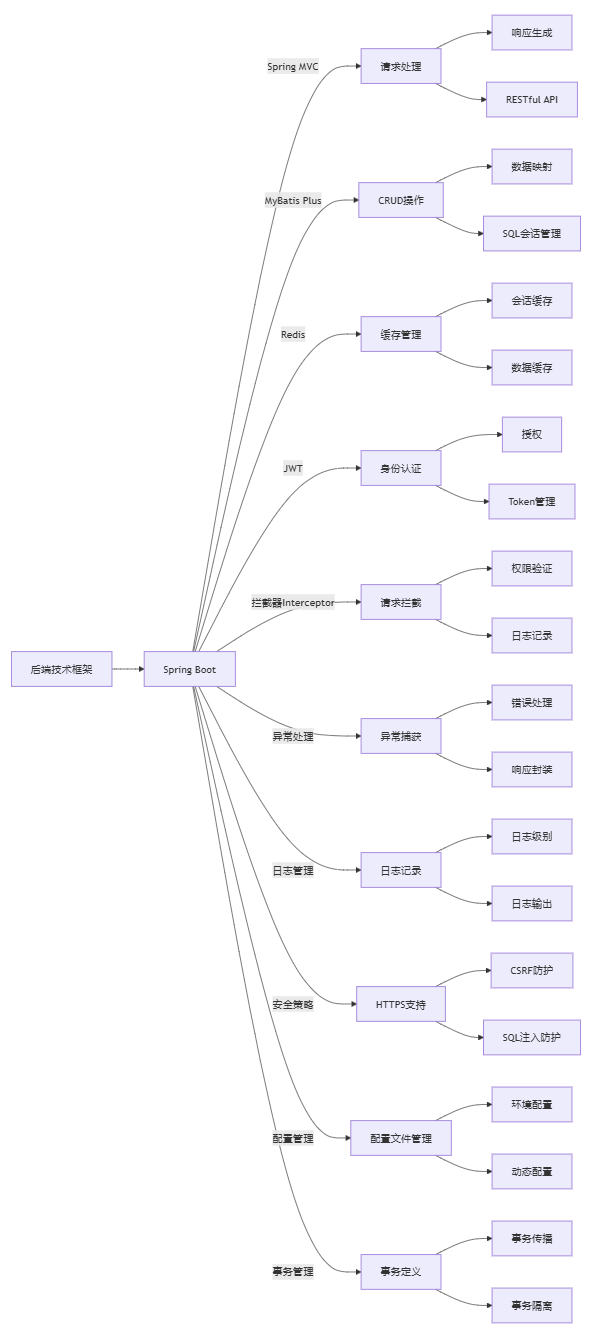
INSERT INTO `at\_work` VALUES ('1856719831294873602', '1740737903736152066', '测试03333111', '<p>测试0333311<br/></p>', '1', 63, '0', '2024-11-13 23:24:39', '2024-11-15 12:54:02', '2024-11-13 23:24:39');

INSERT INTO `at\_work` VALUES ('1857274202878230530', '1740737903736152066', 'aaa', '<p>aaa</p>', '1', 0, '0', '2024-11-15 12:07:31', '2024-11-15 12:07:31', '2024-11-15 12:07:31');

INSERT INTO `at\_work` VALUES ('1857285440379138049', '1742550907872137217', '测试——0002', '<p>测试——0002<br/></p>', '1', 4, '0', '2024-11-15 12:52:10', '2024-11-15 12:53:15', '2024-11-15 12:52:10');

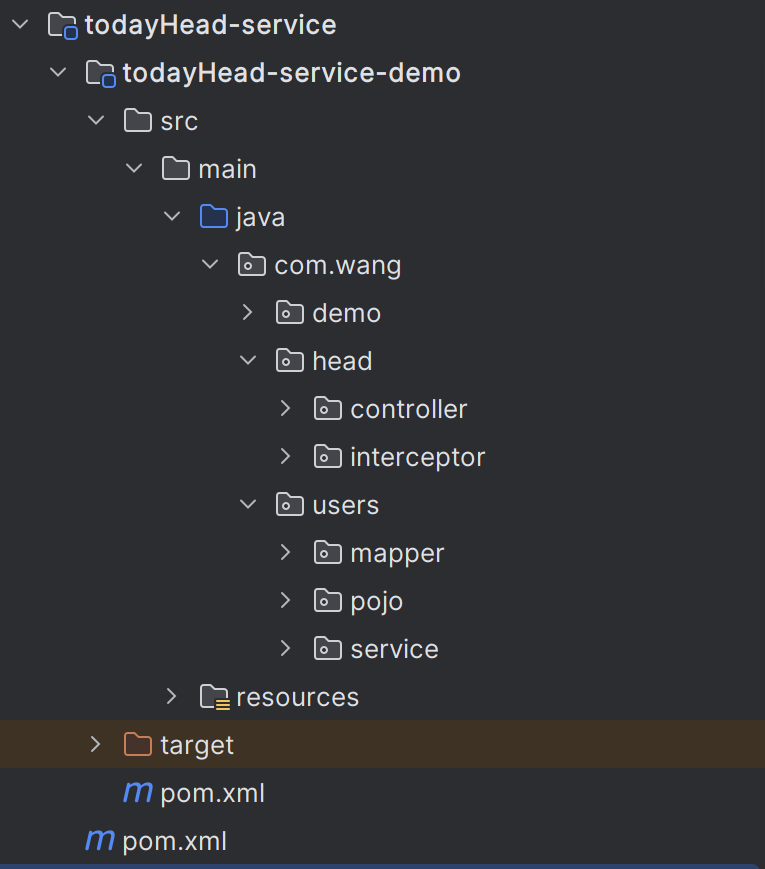
SET FOREIGN\_KEY\_CHECKS = 1;

# 六、后端整体框架图

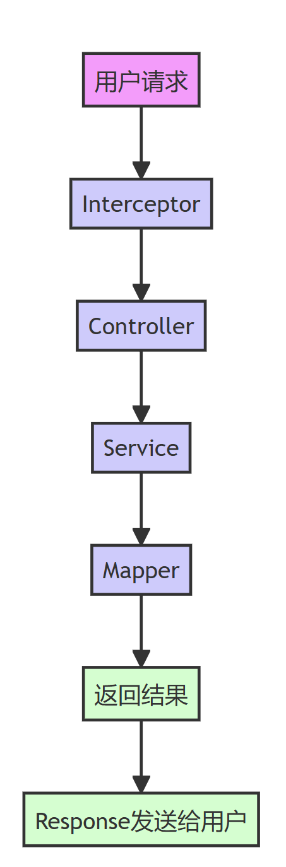


# 七、后端代码实现

## 1.后端整体目录



## 2.执行流程



### （1）用户请求发起

用户在浏览器或客户端发起HTTP请求，Spring Boot应用中的 DispatcherServlet接收请求。

### （2）拦截器Interceptor处理请求

在请求进入Controller之前，拦截器Interceptor会先进行处理。拦截器可以用来进行身份验证、日志记录等。

如果拦截器的preHandle返回false，请求会被中断，不会进入Controller层。

### （3）Controller处理请求

当请求通过拦截器时，Controller层接收请求并调用Service层进行业务处理。

Controller可以通过路径、参数等从前端接收信息，并传递到Service层。

### （4）Service进行业务处理

Service层根据业务需求处理数据（如逻辑判断、数据计算等）。

从数据库中获取数据，Service会调用Mapper层的方法来查询数据库。

### （5）Mapper与数据库交互

Mapper层通过SQL查询（MyBatis）来操作数据库。

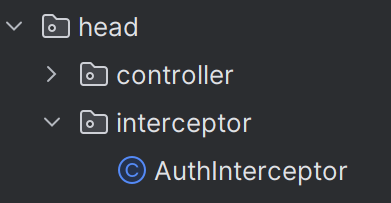
Mapper返回的数据会被传递到Service层，再由Service返回给Controller。

### （6）响应结果返回给客户端

当Controller接收到来自Service层的处理结果后，通常会将数据封装在 ResponseEntity或直接返回给前端。

## 3.Interceptor（拦截器）

### （1）文件位置



### （2）主要作用

拦截器的作用是在Spring MVC中拦截请求，在请求到达Controller之前进行处理。

这段代码实现了一个登录验证拦截器，主要功能是在用户访问受保护的资源时，检查用户是否已经登录，如果没有登录，则会被重定向到登录页面。

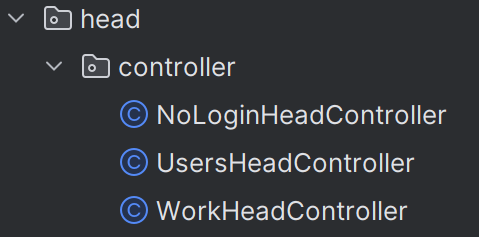
在这段代码中，AuthInterceptor实现了HandlerInterceptor接口，因此能够拦截所有进入请求的方法。preHandle方法是HandlerInterceptor接口中的一个方法，会在请求到达Controller之前执行，具体完成了用户登录验证的功能。

### （3）相关代码（AuthInterceptor.java）

@Log4j2  
@Component("authInterceptor")  
public class AuthInterceptor extends BaseController implements HandlerInterceptor {  
 @Autowired  
 private RedisUtil redisUtil;  
 @Override  
 public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {  
 log.info("--授权拦截器执行--");  
 JSONObject resultJson = new JSONObject();  
 HttpSession session = request.getSession();  
 /\* 获取用户对象 \*/  
 AtUsersDesc users = (AtUsersDesc) this.findObjFromSession(request, BasePojoEnum.SESS\_USERS);  
 if (users == null) {  
 String info = (String) this.redisUtil.hashGet(RedisUtil.REDIS\_KEY, ConstatFinalUtil.FINAL\_CONFIG\_INFO\_FAILED);  
 JSONObject infoJson = JSON.parseObject(info);  
 resultJson.put("code", "1");  
  
 resultJson.put("info", "非法访问,请先登录");  
  
 /\* 提示信息 \*/  
 session.setAttribute(ConstatFinalUtil.FINAL\_RESPONSE, resultJson);  
  
 response.sendRedirect(request.getContextPath() + "/login");  
 return false;  
 }  
 return true;  
 }  
}

## 4.Controller（控制器）

### （1）文件位置



### （2）主要作用

Controller负责接收请求并调用Service层处理业务，接受并处理来自前端的请求，调用相应的业务逻辑处理，然后返回结果。通过注解@RestController或@Controller标记。

使用@RequestMapping或其他映射注解（如 @GetMapping，@PostMapping）来定义方法与HTTP请求的映射。

在请求到达时，Controller接收请求并调用Service层处理具体的业务逻辑。

### （3）相关代码

#### ①NoLoginHeadController.java

这部分代码主要负责处理与用户登录、注册相关的请求。下面是各个方法的功能总结：

登录页面 (login方法)

@RequestMapping("/login")  
 public String login(HttpServletRequest request, HttpServletResponse response, HttpSession session) {  
 log.info("==login==");  
  
 AtUsersDesc users = (AtUsersDesc) this.findObjFromSession(request, BasePojoEnum.SESS\_USERS);  
// 获取会话中的 SESS\_USERS 对象，这个对象代表当前登录的用户。如果该对象不为空，说明用户已经登录。  
 if (users != null) {  
 if (StringUtils.isEmpty(users.getId())) {  
 return "/head/usersLogin";  
 }  
 /\* 说明用户肯定是登录了 \*/  
 return "redirect:/head/users/main";  
 }  
 return "/head/usersLogin";  
 }

当用户访问登录页面时，首先会检查会话中是否存在已经登录的用户（通过 findObjFromSession获取SESS\_USERS）。

如果已经登录，且用户的ID不为空，则会重定向到用户主页 (/head/users/main)。

如果没有登录，则跳转到登录页面 (/head/usersLogin)。

登录提交操作 (loginSubmit方法)

@RequestMapping("/loginSubmit")  
public String loginSubmit(HttpServletRequest request, String email, String password, String code, HttpSession session,  
 HttpServletResponse response, Model model) {  
 log.info("==loginSubmit==");  
 String info = "";  
 /\* 接收参数 \*/  
 String sessionCode = session.getAttribute("rand") + "";  
 log.info("==email==" + email);  
 log.info("==password==" + password);  
 log.info(code + "==password==" + password);  
 if (email == null || "".equalsIgnoreCase(email)) {  
 info = "请输入邮箱";  
 } else {  
 /\* 万能验证码 \*/  
 if (sessionCode.equalsIgnoreCase(code) || ConstatFinalUtil.FINAL\_CODE\_STR.equalsIgnoreCase(code)) {  
 /\* query对象 \*/  
 AtUsersDescQuery usersQuery = new AtUsersDescQuery();  
 AtUsersDesc usersParam = usersQuery.getEntity();  
 usersParam.setEmail(email);  
 AtUsersDesc users = this.usersDescService.findOne(usersQuery);  
 /\* 关联对象 \*/  
 if (Objects.nonNull(users)) {  
 /\* 根据邮箱查询到了===验证密码 \*/  
 if (this.encryptUtil.match(users.getPassword(), password)) {  
 /\* 密码正确==看状态 \*/  
 if (BasePojoEnum.STATUS\_ENABLE.getCode().equalsIgnoreCase(users.getStatus())) {  
 /\* 登录成功 \*/  
 session.setAttribute(BasePojoEnum.SESS\_USERS.getCode(), users);  
 session.setAttribute("lastLoginTime", users.getLastLoginTime());  
 /\* 更新上次登录时间 \*/  
 users.setLastLoginTime(new Date());  
 boolean dbFlag = this.usersDescService.updateById(users);  
 log.info("==用户操作数据库结果:=={}", dbFlag);  
 return "redirect:/head/users/main";  
 } else {  
 info = "账号状态不正确,请联系管理员";  
 }  
 } else {  
 /\* 密码不正确 \*/  
 info = "密码不正确";  
 }  
 model.addAttribute("email", email);  
 } else {  
 /\* 邮箱填写不正确 \*/  
 info = "邮箱填写不正确";  
 }  
 } else {  
 /\* 验证码不正确 \*/  
 info = "验证码不正确";  
 }  
 }  
 JSONObject resultJson = new JSONObject();  
 resultJson.put("info", info);  
 model.addAttribute(ConstatFinalUtil.FINAL\_RESPONSE, resultJson);  
 return this.login(request, response, session);  
}

用户在登录页面提交表单时，loginSubmit方法会处理用户输入的邮箱、密码和验证码。

首先检查邮箱是否为空。如果为空，则提示“请输入邮箱”。

然后通过会话中的验证码（sessionCode）与用户输入的验证码进行比对。

如果验证码正确，再验证用户输入的邮箱和密码。

如果邮箱和密码正确，且账户状态有效（通过users.getStatus()判断），则会将用户对象存入会话中，并更新用户的最后登录时间。

登录成功后会跳转到用户主页/head/users/main。

如果任何验证失败（如邮箱不正确、密码错误或账户状态无效），则返回错误信息，显示在登录页面。

注册页面 (register方法)

@RequestMapping("/register")  
public String register(HttpServletRequest request, HttpServletResponse response, HttpSession session) {  
 log.info("==register==");  
 return "/head/usersRegister";  
}

当用户访问注册页面时，跳转到/head/usersRegister页面。

注册提交操作 (registerSubmit方法)

@RequestMapping("/registerSubmit")  
public String registerSubmit(HttpServletRequest request, String email, String password, String code, HttpSession session,  
 HttpServletResponse response, Model model) {  
 log.info("==registerSubmit==");  
 String info = "";  
 /\* 接收参数 \*/  
 try {  
 /\* 获取到图片中的code \*/  
 String sessionCode = session.getAttribute("rand") + "";  
 if (email == null || "".equalsIgnoreCase(email)) {  
 info = "请输入邮箱";  
 } else {  
 /\* 万能验证码 \*/  
 if (sessionCode.equalsIgnoreCase(code) || ConstatFinalUtil.FINAL\_CODE\_STR.equalsIgnoreCase(code)) {  
 /\* 这叫query对象 \*/  
 AtUsersDesc users = new AtUsersDesc();  
 users.setEmail(email);  
 users.setPassword(this.encryptUtil.encrypt(password));  
 /\* 关联对象 \*/  
 boolean dbFlag = this.usersDescService.save(users);  
 log.info("==用户操作数据库结果:=={}", dbFlag);  
 info = "注册成功";  
 model.addAttribute("email", email);  
 } else {  
 /\* 验证码不正确 \*/  
 info = "验证码不正确";  
 }  
 }  
 } catch (Exception e) {  
 log.error("==注册用户报错==", e);  
 info = e.getMessage();  
 }  
 JSONObject resultJson = new JSONObject();  
 resultJson.put("info", info);  
 model.addAttribute(ConstatFinalUtil.FINAL\_RESPONSE, resultJson);  
 return this.register(request, response, session);  
}

用户在注册页面提交表单时，registerSubmit方法会处理用户输入的邮箱、密码和验证码。

首先检查邮箱是否为空。如果为空，则提示“请输入邮箱”。

验证验证码是否正确。如果验证码正确，则会创建一个新的用户对象，保存用户的邮箱和加密后的密码。

然后通过 usersDescService.save()方法将用户信息保存到数据库。

如果保存成功，提示“注册成功”，否则会捕获异常并提示错误信息。

#### ②UsersHeadController.java

这部分代码主要负责处理与用户首页相关的请求，包括用户信息展示、更新、注销等功能。以下是各个方法的功能总结：

首页 (main方法)

@RequestMapping("/main")  
public String main(HttpServletRequest request, Model model) {  
 log.info("--main--");  
 AtUsersDesc usersSess = (AtUsersDesc) this.findObjFromSession(request, BasePojoEnum.SESS\_USERS);  
 if (StringUtils.isEmpty(usersSess.getId())) {  
 return "redirect:/login";  
 }  
 AtUsersDescQuery paramQuery = new AtUsersDescQuery();  
 AtUsersDesc entity = paramQuery.getEntity();  
 entity.setId(usersSess.getId());  
 AtUsersDesc one = this.usersDescService.findOne(paramQuery);  
 model.addAttribute(ConstatFinalUtil.FINAL\_CODE\_ONE, one);  
 return "/head/users/main";  
}

处理用户首页的请求。首先检查当前会话中是否有已登录的用户。如果没有登录，则重定向到登录页面。

如果用户已经登录，查询该用户的详细信息并将其添加到模型中，返回用户首页视图。

欢迎页面 (welcome方法)

@RequestMapping("/welcome")  
public String welcome() {  
 log.info("--welcome--");  
 return "/head/users/welcome";  
}

打开一个简单的欢迎页面，通常用于用户登录后的欢迎信息展示。

打开用户更新页面 (usersUpdate方法)

@RequestMapping("/usersUpdate")  
 public String usersUpdate(HttpServletRequest request, AtUsersDescQuery paramQuery,Model model) {  
 log.info("--usersUpdate--");  
  
 /\* 从session中获取信息 \*/  
 AtUsersDesc one = (AtUsersDesc) this.findObjFromSession(request, BasePojoEnum.SESS\_USERS);  
 AtUsersDesc entity = paramQuery.getEntity();  
 entity.setId(one.getId());  
 one = this.usersDescService.findOne(paramQuery);  
 model.addAttribute(ConstatFinalUtil.FINAL\_CODE\_ONE, one);  
 model.addAttribute(ConstatFinalUtil.FINAL\_CODE\_PARAMQUERY, paramQuery);  
 if ("updateInfo".equalsIgnoreCase(paramQuery.getOperType())) {  
 /\* 根据不同的请求跳转不同的页面 \*/  
 return "/head/users/usersUpdateInfo"; // 更新信息  
 } else if ("updatePass".equalsIgnoreCase(paramQuery.getOperType())) {  
// 根据 paramQuery.getOperType() 的值判断请求的操作类型  
 /\* 根据不同的请求跳转不同的页面 \*/  
 return "/head/users/usersUpdatePass"; // 更新密码  
 }  
 return "/head/users/usersInfo";  
 }

根据请求参数（如operType）决定跳转到不同的用户更新页面。

如果操作类型是updateInfo，则跳转到更新个人信息页面；如果是updatePass，则跳转到更新密码页面。

获取当前用户信息，并将其添加到模型中。

提交用户更新操作 (usersUpdateSubmit方法)

@RequestMapping("/usersUpdateSubmit")  
 public String usersUpdateSubmit(HttpServletRequest request,AtUsersDescQuery paramQuery, Model model) {  
 log.info("--usersUpdateSubmit--");  
 AtUsersDesc users = (AtUsersDesc) this.findObjFromSession(request, BasePojoEnum.SESS\_USERS);  
 /\* 从session中获取信息 \*/  
 AtUsersDesc one = (AtUsersDesc) this.findObjFromSession(request, BasePojoEnum.SESS\_USERS);  
 AtUsersDesc entityCond = paramQuery.getEntity();  
 entityCond.setId(one.getId());  
 one = this.usersDescService.findOne(paramQuery);  
  
 if (BaseQuery.FINAL\_OPER\_UPDATE.equalsIgnoreCase(paramQuery.getOperType())) {  
 AtUsersDesc entity = paramQuery.getEntity();  
  
 boolean dbFlag = this.usersDescService.updateById(entity);  
 JSONObject resultJson = this.proccedResponseJson(dbFlag);  
// 根据更新操作的成功与否，返回相应的结果，使用proccedResponseJson()方法封装结果并返回给前端。  
 log.info("==dbFlag=={}", dbFlag);  
 model.addAttribute(ConstatFinalUtil.FINAL\_RESPONSE, resultJson);  
 return this.usersUpdate(request, paramQuery,model);  
 }  
 return "/head/users/usersInfo";  
 }

处理用户更新操作（如更新个人信息或密码）。

根据操作类型（operType）判断更新内容（个人信息或密码），然后更新数据库。

更新操作结果通过proccedResponseJson方法封装为JSON格式返回给前端。

退出登录 (logout方法)

@RequestMapping("/logout")  
public String logout(HttpServletRequest request, HttpSession session) {  
 log.info("--logout--");  
 /\* 删除 \*/  
 session.removeAttribute(BasePojoEnum.SESS\_USERS.getCode());  
 session.removeAttribute("lastLoginTime");  
 /\* 客户端跳转到登陆页面 \*/  
 return "redirect:/login";  
}

处理用户退出操作。删除当前会话中的用户信息并重定向到登录页面。

用户列表页面 (usersList方法)

@RequestMapping("/usersList")  
public String usersList(HttpServletRequest request, AtUsersDescQuery paramQuery,Model model) {  
 log.info("--usersList--");  
 /\* 分页对象 \*/  
 IPage<AtUsersDesc> paramPage = null;  
 try {  
 paramPage = new Page<>(paramQuery.getCurrent(), paramQuery.getSize());  
 } catch (Exception e) {  
 paramPage = new Page<>(1, 20);  
 }  
 /\* 时间处理 \*/  
 if (StringUtils.isNotEmpty(paramQuery.getStDate())) {  
 paramQuery.setPubTimeSt(this.dateUtil.strToDateTime(paramQuery.getStDate()));  
 }  
 paramQuery.addLike();  
  
 List<AtUsersDesc> dataList = this.usersDescService.findList(paramPage, paramQuery);  
 /\* 列表数据 \*/  
 model.addAttribute(ConstatFinalUtil.FINAL\_CODE\_DATALIST, dataList);  
 /\* 分页数据 \*/  
 model.addAttribute(ConstatFinalUtil.FINAL\_CODE\_PAGEINFOUTIL, paramPage);  
 model.addAttribute(ConstatFinalUtil.FINAL\_CODE\_PARAMQUERY, paramQuery);  
 model.addAttribute(ConstatFinalUtil.FINAL\_CODE\_ENUMSONE, new AtUsersCollect());  
 return "/head/users/usersList";  
}

显示用户列表，支持分页查询。

根据查询参数（如开始时间stDate）和分页信息，查询用户数据并将其添加到模型中，供前端显示。

支持对用户列表的分页显示，查询参数和分页信息都传递给前端页面。

#### ③WorkHeadController.java

这部分代码用于处理用户首页与作品、收藏、评论、私信等相关操作的控制器。它结合了多种功能，包括展示作品列表、添加和更新作品、评论管理、私信管理等。以下是该类中几个主要功能的总结：

@Log4j2  
@Controller  
@RequestMapping("/head/work/")  
public class WorkHeadController extends BaseController {  
 @Autowired  
 private AtWorkService workService;  
 @Autowired  
 private AtUsersCollectService collectService;  
 @Autowired  
 private AtUsersCommentService commentService;  
 @Autowired  
 private AtUsersMessService messService;

作品管理 (workList, workInsert, workUpdate等)

作品查询多条 (/workList): 提供了一个分页功能来显示作品列表，同时支持基于时间和其他条件进行筛选。

@RequestMapping("/workList")  
public String workList(AtWorkQuery paramQuery, Model model, HttpServletRequest request) {  
 log.info("==作品列表==");  
 AtUsersDesc usersDescSess = (AtUsersDesc) this.findObjFromSession(request, BasePojoEnum.SESS\_USERS);  
 /\* 分页对象 \*/  
 IPage<AtWork> paramPage = null;  
 try {  
 paramPage = new Page<>(paramQuery.getCurrent(), paramQuery.getSize());  
 } catch (Exception e) {  
 paramPage = new Page<>(1, 20);  
 }  
 /\* 时间处理 \*/  
 if (StringUtils.isNotEmpty(paramQuery.getStDate())) {  
 paramQuery.setPubTimeSt(this.dateUtil.strToDateTime(paramQuery.getStDate()));  
 }  
 paramQuery.addLike();  
  
 /\* 增加条件 \*/  
 AtWork entity = paramQuery.getEntity();  
 if (AtWorkQuery.FINAL\_SELF\_OBJ\_TRUE.equalsIgnoreCase(paramQuery.getSelfObj())) {  
 entity.setUsersId(usersDescSess.getId());  
 }  
  
 /\* 查询对象 \*/  
 List<AtWork> dataList = this.workService.findList(paramPage, paramQuery);  
 paramQuery.removeLike();  
 /\* 列表数据 \*/  
 model.addAttribute(ConstatFinalUtil.FINAL\_CODE\_DATALIST, dataList);  
 /\* 分页数据 \*/  
 model.addAttribute(ConstatFinalUtil.FINAL\_CODE\_PAGEINFOUTIL, paramPage);  
 model.addAttribute(ConstatFinalUtil.FINAL\_CODE\_PARAMQUERY, paramQuery);  
 model.addAttribute(ConstatFinalUtil.FINAL\_CODE\_ENUMSONE, new AtWork());  
 /\* 查询数据 \*/  
 return "/head/work/workList";  
}

作品添加 (/workInsert): 提供了一个表单页面用于添加新的作品。

@RequestMapping("/workInsert")  
public String workInsert(AtWorkQuery paramQuery, Model model, HttpServletRequest request) {  
 log.info("==作品添加==");  
 AtWork entity = paramQuery.getEntity();  
 model.addAttribute(ConstatFinalUtil.FINAL\_CODE\_ONE, entity);  
 /\* 查询数据 \*/  
 return "/head/work/workInsert";

作品插入提交 (/workInsertSubmit): 提交作品数据并保存。

@RequestMapping("/workInsertSubmit")  
public String workInsertSubmit(AtWorkQuery paramQuery, Model model, HttpServletRequest request) {  
 log.info("==作品添加提交==");  
 AtWork entity = paramQuery.getEntity();  
 /\* 从session中获取信息 \*/  
 AtUsersDesc one = (AtUsersDesc) this.findObjFromSession(request, BasePojoEnum.SESS\_USERS);  
 if (Objects.nonNull(one)) {  
 entity.setUsersId(one.getId());  
 }  
 boolean dbFlag = this.workService.save(entity);  
 JSONObject resultJson = this.proccedResponseJson(dbFlag);  
 model.addAttribute(ConstatFinalUtil.FINAL\_RESPONSE, resultJson);  
 /\* 查询数据 \*/  
 return this.workInsert(paramQuery, model, request);  
}

作品更新 (/workUpdate): 提供了一个页面来查看和更新已有作品的信息。

@RequestMapping("/workUpdate")  
public String workUpdate(AtWorkQuery paramQuery, Model model) {  
 log.info("==作品更新==");  
 /\* 根据id查询 \*/  
 AtWork one = this.workService.findOne(paramQuery);  
 model.addAttribute(ConstatFinalUtil.FINAL\_CODE\_ONE, one);  
 model.addAttribute(ConstatFinalUtil.FINAL\_CODE\_PARAMQUERY, paramQuery);  
 if (BaseQuery.FINAL\_OPER\_UPDATE.equalsIgnoreCase(paramQuery.getOperType())) {  
 /\* 查询数据 \*/  
 return "/head/work/workUpdate";  
 }  
 /\* 更新浏览量 \*/  
 one.setViewCount(one.getViewCount() + 1);  
 this.workService.updateById(one);  
 /\* 查询评论列表 \*/  
 AtUsersCommentQuery commentQuery = new AtUsersCommentQuery();  
 AtUsersComment commentEntity = commentQuery.getEntity();  
 commentEntity.setWorkId(one.getId());  
 List<AtUsersComment> commentList = this.commentService.findList(null, commentQuery);  
 model.addAttribute("commentList", commentList);  
 /\* 查询数据 \*/  
 return "/head/work/workInfo";  
}

作品更新提交 (/workUpdateSubmit): 提交作品数据并保存。

@RequestMapping("/workUpdateSubmit")  
public String workUpdateSubmit(AtWorkQuery paramQuery, Model model) {  
 log.info("==作品更新提交==");  
 AtWork entity = paramQuery.getEntity();  
 if (StringUtils.isNotEmpty(paramQuery.getPubTime())) {  
 entity.setPubTime(this.dateUtil.strToDateTime(paramQuery.getPubTime()));  
 }  
 /\* 根据id查询 \*/  
 boolean dbFlag = this.workService.updateById(entity);  
 JSONObject resultJson = this.proccedResponseJson(dbFlag);  
 model.addAttribute(ConstatFinalUtil.FINAL\_RESPONSE, resultJson);  
 /\* 查询数据 \*/  
 return this.workUpdate(paramQuery, model);  
}

收藏管理 (collectList, collectInsertSubmit, collectDelete等)

收藏查询 (/collectList): 提供了分页的收藏查询功能，列出用户收藏的所有作品。

@RequestMapping("/collectList")  
public String collectList(AtUsersCollectQuery paramQuery, Model model, HttpServletRequest request) {  
 log.info("==收藏列表==");  
 AtUsersDesc usersDescSess = (AtUsersDesc) this.findObjFromSession(request, BasePojoEnum.SESS\_USERS);  
 /\* 分页对象 \*/  
 IPage<AtUsersCollect> paramPage = null;  
 try {  
 paramPage = new Page<>(paramQuery.getCurrent(), paramQuery.getSize());  
 } catch (Exception e) {  
 paramPage = new Page<>(1, 20);  
 }  
 /\* 时间处理 \*/  
 if (StringUtils.isNotEmpty(paramQuery.getStDate())) {  
 paramQuery.setPubTimeSt(this.dateUtil.strToDateTime(paramQuery.getStDate()));  
 }  
 paramQuery.addLike();  
  
 /\* 增加条件 \*/  
 AtUsersCollect entity = paramQuery.getEntity();  
 entity.setUsersId(usersDescSess.getId());  
  
 /\* 查询对象 \*/  
 List<AtUsersCollect> dataList = this.collectService.findList(paramPage, paramQuery);  
 paramQuery.removeLike();  
 /\* 列表数据 \*/  
 model.addAttribute(ConstatFinalUtil.FINAL\_CODE\_DATALIST, dataList);  
 /\* 分页数据 \*/  
 model.addAttribute(ConstatFinalUtil.FINAL\_CODE\_PAGEINFOUTIL, paramPage);  
 model.addAttribute(ConstatFinalUtil.FINAL\_CODE\_PARAMQUERY, paramQuery);  
 model.addAttribute(ConstatFinalUtil.FINAL\_CODE\_ENUMSONE, new AtUsersCollect());  
 /\* 查询数据 \*/  
 return "/head/work/collectList";  
}

收藏添加 (/collectInsertSubmit): 支持用户将作品添加到收藏夹。

@RequestMapping(value = "/collectInsertSubmit", produces = "text/html;charset=UTF-8")  
@ResponseBody  
public String collectInsertSubmit(AtUsersCollectQuery paramQuery, Model model, HttpServletRequest request) {  
 log.info("==收藏添加提交==");  
 JSONObject resultJson = new JSONObject();  
 try {  
 AtUsersCollect entity = paramQuery.getEntity();  
 /\* 从session中获取信息 \*/  
 AtUsersDesc one = (AtUsersDesc) this.findObjFromSession(request, BasePojoEnum.SESS\_USERS);  
 if (Objects.nonNull(one)) {  
 entity.setUsersId(one.getId());  
 }  
 boolean dbFlag = this.collectService.save(entity);  
 resultJson = this.proccedResponseJson(dbFlag);  
 } catch (Exception e) {  
 log.error("操作失败", e);  
 resultJson.put("info", e.getMessage());  
 resultJson.put("code", "-1");  
 }  
 /\* 查询数据 \*/  
 return resultJson.toJSONString();  
}

收藏删除 (/collectDelete): 允许用户删除收藏的作品。

@RequestMapping(value = "/collectDelete")  
public String collectDelete(AtUsersCollectQuery paramQuery, Model model, HttpServletRequest request) {  
 AtUsersCollect entity = paramQuery.getEntity();  
 model.addAttribute(ConstatFinalUtil.FINAL\_CODE\_ONE, entity);  
 int dbRes = this.collectService.deleteBatch(paramQuery);  
 log.info("==收藏删除,id:{},删除条数:{}==", entity.getId(), dbRes);  
 JSONObject resultJson = this.proccedResponseJson(dbRes > 0);  
 /\* 查询数据 \*/  
 return this.collectList(paramQuery, model, request);  
}

评论管理 (commentList, commentInsert, commentUpdate等)

评论查询 (/commentList): 提供了分页功能来查询用户评论过的作品。

@RequestMapping("/commentList")  
public String commentList(AtUsersCommentQuery paramQuery, Model model, HttpServletRequest request) {  
 log.info("==评论列表==");  
 AtUsersDesc usersDescSess = (AtUsersDesc) this.findObjFromSession(request, BasePojoEnum.SESS\_USERS);  
 /\* 分页对象 \*/  
 IPage<AtUsersComment> paramPage = null;  
 try {  
 paramPage = new Page<>(paramQuery.getCurrent(), paramQuery.getSize());  
 } catch (Exception e) {  
 paramPage = new Page<>(1, 20);  
 }  
 /\* 时间处理 \*/  
 if (StringUtils.isNotEmpty(paramQuery.getStDate())) {  
 paramQuery.setPubTimeSt(this.dateUtil.strToDateTime(paramQuery.getStDate()));  
 }  
 paramQuery.addLike();  
  
 /\* 增加条件 \*/  
 AtUsersComment entity = paramQuery.getEntity();  
 entity.setUsersId(usersDescSess.getId());  
  
 /\* 查询对象 \*/  
 List<AtUsersComment> dataList = this.commentService.findList(paramPage, paramQuery);  
 paramQuery.removeLike();  
 /\* 列表数据 \*/  
 model.addAttribute(ConstatFinalUtil.FINAL\_CODE\_DATALIST, dataList);  
 /\* 分页数据 \*/  
 model.addAttribute(ConstatFinalUtil.FINAL\_CODE\_PAGEINFOUTIL, paramPage);  
 model.addAttribute(ConstatFinalUtil.FINAL\_CODE\_PARAMQUERY, paramQuery);  
 model.addAttribute(ConstatFinalUtil.FINAL\_CODE\_ENUMSONE, new AtUsersComment());  
 /\* 查询数据 \*/  
 return "/head/work/commentList";  
}

评论添加 (/commentInsert): 提供了一个表单来添加新的评论。

@RequestMapping("/commentInsert")  
public String commentInsert(AtUsersCommentQuery paramQuery, Model model, HttpServletRequest request) {  
 log.info("==评论添加==");  
 AtUsersComment entity = paramQuery.getEntity();  
 model.addAttribute(ConstatFinalUtil.FINAL\_CODE\_ONE, entity);  
 /\* 查询数据 \*/  
 return "/head/work/commentInsert";  
}

评论添加提交 (/commentInsertSubmit): 提交作品数据并保存。

@RequestMapping("/commentInsertSubmit")  
public String commentInsertSubmit(AtUsersCommentQuery paramQuery, Model model, HttpServletRequest request) {  
 log.info("==评论添加提交==");  
 AtUsersComment entity = paramQuery.getEntity();  
 /\* 从session中获取信息 \*/  
 AtUsersDesc one = (AtUsersDesc) this.findObjFromSession(request, BasePojoEnum.SESS\_USERS);  
 if (Objects.nonNull(one)) {  
 entity.setUsersId(one.getId());  
 }  
 boolean dbFlag = this.commentService.save(entity);  
 JSONObject resultJson = this.proccedResponseJson(dbFlag);  
 model.addAttribute(ConstatFinalUtil.FINAL\_RESPONSE, resultJson);  
 /\* 查询数据 \*/  
 return this.commentInsert(paramQuery, model, request);  
}

评论更新 (/commentUpdate): 支持编辑和更新已发布的评论。

@RequestMapping("/commentUpdate")  
public String commentUpdate(AtUsersCommentQuery paramQuery, Model model) {  
 log.info("==评论更新==");  
 /\* 根据id查询 \*/  
 AtUsersComment one = this.commentService.findOne(paramQuery);  
 model.addAttribute(ConstatFinalUtil.FINAL\_CODE\_ONE, one);  
 model.addAttribute(ConstatFinalUtil.FINAL\_CODE\_PARAMQUERY, paramQuery);  
 if (BaseQuery.FINAL\_OPER\_UPDATE.equalsIgnoreCase(paramQuery.getOperType())) {  
 /\* 查询数据 \*/  
 return "/head/work/commentUpdate";  
 }  
 /\* 查询数据 \*/  
 return "/head/work/commentInfo";  
}

评论更新提交 (/commentUpdateSubmit): 提交作品数据并保存。

@RequestMapping("/commentUpdateSubmit")  
public String commentUpdateSubmit(AtUsersCommentQuery paramQuery, Model model) {  
 log.info("==评论更新提交==");  
 AtUsersComment entity = paramQuery.getEntity();  
 if (StringUtils.isNotEmpty(paramQuery.getPubTime())) {  
 entity.setPubTime(this.dateUtil.strToDateTime(paramQuery.getPubTime()));  
 }  
 /\* 根据id查询 \*/  
 boolean dbFlag = this.commentService.updateById(entity);  
 JSONObject resultJson = this.proccedResponseJson(dbFlag);  
 model.addAttribute(ConstatFinalUtil.FINAL\_RESPONSE, resultJson);  
 /\* 查询数据 \*/  
 return this.commentUpdate(paramQuery, model);  
}

评论删除 (/commentUpdateSubmit): 提交作品数据并保存。

@RequestMapping(value = "/commentDelete", produces = "text/html;charset=UTF-8")

@ResponseBody

public String commentDelete(AtUsersCommentQuery paramQuery, Model model) {

AtUsersComment entity = paramQuery.getEntity();

model.addAttribute(ConstatFinalUtil.FINAL\_CODE\_ONE, entity);

boolean dbFlag = this.collectService.removeById(entity.getId());

log.info("==评论删除,id:{},结果:{}==", entity.getId(), dbFlag);

JSONObject resultJson = this.proccedResponseJson(dbFlag);

/\* 查询数据 \*/

return resultJson.toJSONString();

}

私信管理 (messList)

私信查询 (/messList): 提供了分页功能来查看用户的私信，包括发送和接收的私信。

@RequestMapping("/messList")  
public String messList(AtUsersMessQuery paramQuery, Model model, HttpServletRequest request) {  
 log.info("==私信列表==");  
 AtUsersDesc usersDescSess = (AtUsersDesc) this.findObjFromSession(request, BasePojoEnum.SESS\_USERS);  
 /\* 分页对象 \*/  
 IPage<AtUsersMess> paramPage = null;  
 try {  
 paramPage = new Page<>(paramQuery.getCurrent(), paramQuery.getSize());  
 } catch (Exception e) {  
 paramPage = new Page<>(1, 20);  
 }  
 /\* 时间处理 \*/  
 if (StringUtils.isNotEmpty(paramQuery.getStDate())) {  
 paramQuery.setPubTimeSt(this.dateUtil.strToDateTime(paramQuery.getStDate()));  
 }  
 paramQuery.addLike();  
  
 /\* 查询发送者是自己 \*/  
 AtUsersMess entity = paramQuery.getEntity();  
 if (AtUsersMessQuery.FINAL\_TRUE.equalsIgnoreCase(paramQuery.getFromUsersFlag())) {  
 entity.setFromUsersId(usersDescSess.getId());  
 }  
  
 /\* 查询接收者是自己 \*/  
 if (AtUsersMessQuery.FINAL\_TRUE.equalsIgnoreCase(paramQuery.getToUsersFlag())) {  
 entity.setToUsersId(usersDescSess.getId());  
 }  
  
 /\* 查询对象 \*/  
 List<AtUsersMess> dataList = this.messService.findList(paramPage, paramQuery);  
 paramQuery.removeLike();  
 /\* 列表数据 \*/  
 model.addAttribute(ConstatFinalUtil.FINAL\_CODE\_DATALIST, dataList);  
 /\* 分页数据 \*/  
 model.addAttribute(ConstatFinalUtil.FINAL\_CODE\_PAGEINFOUTIL, paramPage);  
 model.addAttribute(ConstatFinalUtil.FINAL\_CODE\_PARAMQUERY, paramQuery);  
 model.addAttribute(ConstatFinalUtil.FINAL\_CODE\_ENUMSONE, new AtUsersMess());  
 /\* 查询数据 \*/  
 return "/head/work/messList";  
}

私信添加 (/messInsert)：提供一个表单，允许用户发送新的私信。

@RequestMapping("/messInsert")  
public String messInsert(AtUsersMessQuery paramQuery, Model model, HttpServletRequest request) {  
 log.info("==私信添加==");  
 AtUsersMess entity = paramQuery.getEntity();  
 model.addAttribute(ConstatFinalUtil.FINAL\_CODE\_ONE, entity);  
 /\* 查询数据 \*/  
 return "/head/work/messInsert";  
}

私信添加提交 (/messInsertSubmit)：提交私信数据，并通过API处理相关逻辑。

@RequestMapping("/messInsertSubmit")  
public String messInsertSubmit(AtUsersMessQuery paramQuery, Model model, HttpServletRequest request) {  
 log.info("==私信添加提交==");  
 AtUsersMess entity = paramQuery.getEntity();  
 /\* 从session中获取信息 \*/  
 AtUsersDesc one = (AtUsersDesc) this.findObjFromSession(request, BasePojoEnum.SESS\_USERS);  
 if (Objects.nonNull(one)) {  
 entity.setFromUsersId(one.getId());  
 }  
 boolean dbFlag = this.messService.save(entity);  
 JSONObject resultJson = this.proccedResponseJson(dbFlag);  
 model.addAttribute(ConstatFinalUtil.FINAL\_RESPONSE, resultJson);  
 /\* 查询数据 \*/  
 return this.messInsert(paramQuery, model, request);  
}

私信更新 (/messUpdate)：更新已有私信内容。

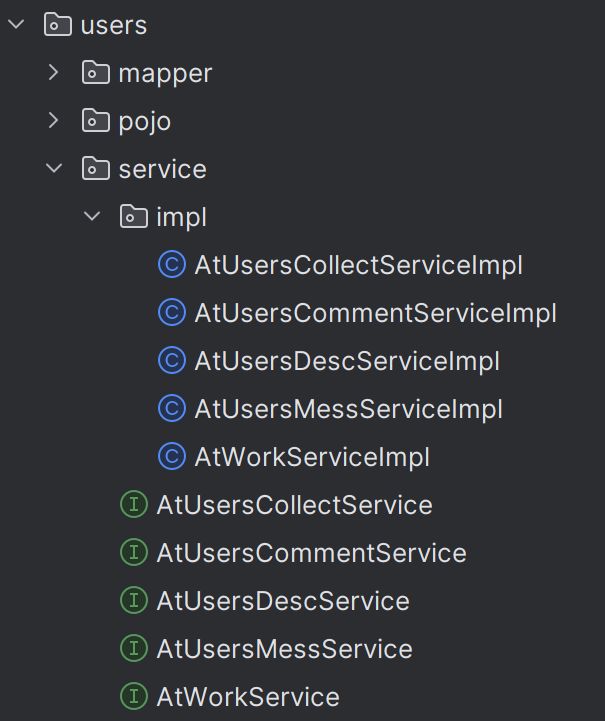
@RequestMapping("/messUpdate")  
public String messUpdate(AtUsersMessQuery paramQuery, Model model) {  
 log.info("==私信更新==");  
 /\* 根据id查询 \*/  
 AtUsersMess one = this.messService.findOne(paramQuery);  
 model.addAttribute(ConstatFinalUtil.FINAL\_CODE\_ONE, one);  
 model.addAttribute(ConstatFinalUtil.FINAL\_CODE\_PARAMQUERY, paramQuery);  
 if (BaseQuery.FINAL\_OPER\_UPDATE.equalsIgnoreCase(paramQuery.getOperType())) {  
 /\* 查询数据 \*/  
 return "/head/work/messUpdate";  
 }  
 /\* 查询数据 \*/  
 return "/head/work/messInfo";  
}

私信删除 (/messDelete)：删除用户的私信。

@RequestMapping("/messDelete")  
public String messDelete(AtUsersMessQuery paramQuery, Model model, HttpServletRequest request) {  
 log.info("==私信删除==");  
 AtUsersMess entity = paramQuery.getEntity();  
 model.addAttribute(ConstatFinalUtil.FINAL\_CODE\_ONE, entity);  
 int dbRes = this.messService.deleteBatch(paramQuery);  
 log.info("==私信删除,id:{},删除条数:{}==", entity.getId(), dbRes);  
 JSONObject resultJson = this.proccedResponseJson(dbRes > 0);  
 /\* 查询数据 \*/  
 return this.messList(paramQuery, model, request);  
}

## 5.Service（服务层）

### （1）文件位置



### （2）主要作用

Service层负责执行具体的业务逻辑，调用Mapper层进行数据持久化操作。

被Controller调用来执行业务逻辑；通过注解@Service来标记，表明这是一个业务层的组件；通过调用Mapper层与数据库进行交互。

### （3）相关代码（以作品（work）相关代码AtWorkServiceImpl.java和AtWorkService.java为例，其余功能代码与work类似）

#### ①AtWorkServiceImpl.java

这段代码是AtWorkServiceImpl类的实现，主要是实现了AtWorkService接口并通过MyBatis Plus提供的ServiceImpl类简化了数据库操作。该类实现了对AtWork实体类的增、删、改、查等基本业务逻辑。下面是对代码的详细解析：

类声明与注解

@Service  
public class AtWorkServiceImpl extends ServiceImpl<AtWorkMapper, AtWork>  
 implements AtWorkService {

@Service注解表示这是一个Spring服务类，Spring会自动将该类注册为 Bean进行管理。

该类继承自MyBatis Plus提供的ServiceImpl类，这个类提供了很多基础的数据库操作方法（如增、删、改、查），简化了数据库操作的实现。

AtWorkServiceImpl类实现了自定义的AtWorkService接口，接口中定义了业务逻辑相关的方法。

字段注入

@Autowired  
private BeanUtil beanUtil;

beanUtil是一个自定义的工具类，用来在不同对象之间进行属性复制，这里通过Spring的依赖注入将其注入到当前类中。

findList方法：查询收藏列表

@Override  
public List<AtWork> findList(IPage<AtWork> page, AtWorkQuery query) {  
 if (page != null) {  
 return this.baseMapper.findList(page, query);  
 }  
 return this.baseMapper.findList(query);  
}

这个方法用于查询符合条件的AtWork列表。

IPage<AtWork>是MyBatis Plus提供的分页接口。如果page不为空，则进行分页查询，否则进行非分页查询。

baseMapper.findList(page, query)调用底层的Mapper接口查询方法。

findOne方法：查询单条收藏记录

@Override

public AtWork findOne(AtWorkQuery query) {

AtWork one = this.baseMapper.findOne(query);

return one;

}

该方法根据查询条件query查询单条AtWork记录并返回。

updateBatch方法：批量更新

@Override  
public int updateBatch(AtWorkQuery query) {  
 return this.baseMapper.updateBatch(query);  
}

该方法用于批量更新符合query条件的AtWork记录，调用底层的Mapper批量更新方法。

deleteBatch方法：批量删除

@Override  
public int deleteBatch(AtWorkQuery query) {  
 return this.baseMapper.deleteBatch(query);  
}

该方法用于批量删除符合query条件的AtWork记录，调用底层的Mapper批量删除方法。

proccedUpdateData方法：数据额外处理

private void proccedUpdateData(AtWork entity){  
 if (Objects.isNull(entity.getCreateTime())) {  
 entity.setCreateTime(new Date());  
 }  
 entity.setUpdateTime(new Date());  
 /\* 判断对象是否为空 \*/  
 if (Objects.isNull(entity.getPubTime())) {  
 entity.setPubTime(new Date());  
 }  
}

该方法用于在保存或更新AtWork实体对象时，自动设置 createTime、updateTime和pubTime字段：

如果createTime为空，则设置为当前时间。

设置updateTime为当前时间。

如果pubTime为空，则设置为当前时间。

save方法：保存数据

@Override  
public boolean save(AtWork entity) {  
 this.proccedUpdateData(entity);  
 /\* 新增加的记录默认未删除 \*/  
 entity.setDelFlag(BasePojoEnum.DELFLAG\_YES.getCode());  
 entity.setStatus(AtWorkEnum.STATUS\_ENABLE.getCode());  
 return super.save(entity);  
}

该方法用于保存新的AtWork实体。

首先调用proccedUpdateData(entity)设置相关时间字段。

设置delFlag为 "未删除" 状态。

设置status为 "启用" 状态。

然后调用super.save(entity)调用MyBatis Plus提供的save方法将实体保存到数据库。

updateById方法：根据ID更新数据

@Override  
public boolean updateById(AtWork entity) {  
 /\* 先根据id查询数据库的内容; \*/  
 AtWorkQuery paramQuery = new AtWorkQuery();  
 AtWork entityParam = paramQuery.getEntity();  
 entityParam.setId(entity.getId());  
 /\* 数据库中查询的结果 \*/  
 AtWork oneDbResult = this.findOne(paramQuery);  
 entity.setUsers(null);  
 oneDbResult.setUsers(null);  
  
 oneDbResult = (AtWork) this.beanUtil.copy(entity, oneDbResult);  
 this.proccedUpdateData(oneDbResult);  
 return super.updateById(oneDbResult);  
}

#### ②AtWorkService.java

这段代码是AtWorkService接口，定义了对AtWork实体进行数据库操作的服务层接口。它继承自MyBatis Plus提供的IService<AtWork>接口，从而获得了MyBatis Plus基本的CRUD操作能力（增、删、改、查）。同时，还定义了一些扩展功能，满足具体业务需求。下面逐步讲解这个接口定义的功能：

接口概述

public interface AtWorkService extends IService<AtWork> {

这个接口继承了MyBatis Plus的IService<AtWork>接口，因此AtWorkService会自动拥有IService接口提供的常用数据库操作方法，如save、updateById、removeById、getById等。

AtWork是一个实体类，表示数据库中的一张表。该接口主要用于处理 AtWork表中的数据。

findList方法：查询多条记录（分页）

List<AtWork> findList(IPage<AtWork> page, AtWorkQuery query);

功能：查询符合条件的AtWork列表，并支持分页查询。

参数：

page：分页对象，MyBatis Plus的IPage接口提供分页功能。

query：查询对象，包含查询条件（例如通过字段过滤数据）。

返回：返回一个AtWork类型的列表，包含符合查询条件的数据。

用途：该方法用于查询符合条件的多条记录，返回分页后的数据集合。通常用于处理需要分页显示的查询结果。

findOne方法：查询一条记录

AtWork findOne(AtWorkQuery query);

功能：根据查询条件返回单条AtWork记录。

参数：query查询条件，通常是通过封装在AtWorkQuery对象中的字段来指定查询条件。

返回：返回符合查询条件的单条AtWork记录，如果没有符合的记录则返回 null。

用途：该方法用于查询符合条件的单条数据记录。常用于获取特定条件下的单个结果。

updateBatch方法：批量更新操作

int updateBatch(AtWorkQuery query);

功能：根据查询条件批量更新AtWork数据表中的记录。通常是批量恢复或更新一些状态信息。

参数：query查询条件，封装了需要更新的记录的标识（如id等）。

返回：返回影响的记录数，即更新成功的条目数。

用途：该方法用于批量更新符合条件的记录。在业务中可以用来实现批量恢复某些数据的功能，通常配合状态字段（如“是否删除”）使用。

deleteBatch方法：批量删除操作

int deleteBatch(AtWorkQuery query);

功能：根据查询条件批量删除AtWork数据表中的记录，进行物理删除。

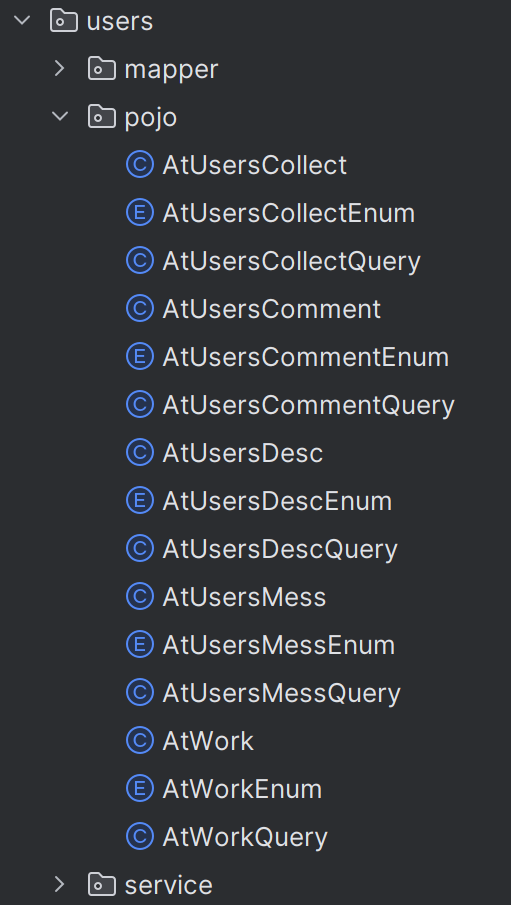
参数：query查询条件，封装了需要删除的记录的标识（如id等）。

返回：返回影响的记录数，即成功删除的条目数。

用途：该方法用于批量删除符合条件的记录，通常用于需要物理删除数据的场景。

## 6.Pojo（实体类）

### （1）文件位置



### （2）主要作用

Pojo用于在Mapper层和Service层之间传递数据，用于表示数据库中的表结构。通常一个Pojo类对应一个数据库表。

包含数据库表中的字段以及对应的getter和setter方法。

被Mapper层用来进行数据库的映射。

### （3）相关代码（以作品（work）相关代码AtWork.java、AtWorkEnum.java和AtWorkQuery.java为例，其余功能代码与work类似）

#### ①AtWork.java

上面的代码定义了AtWork类，它是一个与数据库表at\_work对应的实体类。这个类使用了MyBatis Plus提供的注解，便于与数据库的交互。同时，代码中还包括了一些与枚举和状态处理相关的功能。下面是对代码的详细讲解：

类声明和注解

@TableName(value = "at\_work")  
@Data  
public class AtWork extends BasePojo implements Serializable {  
 private static final long serialVersionUID = -4983153209668388992L;

@TableName(value = "at\_work")：MyBatis Plus提供的注解，用来指定该类对应的数据库表名为at\_work。这意味着AtWork类的实例将与at\_work数据表中的记录进行映射。

@Data：Lombok提供的注解，自动生成类的getter和setter方法，以及 toString、equals、hashCode方法，简化代码。

成员变量

@TableField(value = "viewCount")  
private int viewCount;  
@TableField(value = "usersId")  
private String usersId;

@TableField：MyBatis Plus提供的注解，指定类中的字段与数据库表中的列对应。viewCount对应数据库中的viewCount列，usersId对应usersId列。

非数据库字段：状态字段statusStr和users

@TableField(exist = false)  
private String statusStr;

@TableField(exist = false)：这个注解表示该字段不与数据库表的列对应。statusStr用于存储状态的描述信息，表示AtWork的状态（例如启用、禁用等），并不是从数据库中读取的字段，而是通过代码中的枚举解析动态生成的。

private AtUsersDesc users;

users是一个关联对象引用，表示AtWork与AtUsersDesc之间的关系，假设AtUsersDesc是一个包含用户信息的对象。它不是数据库表中的字段，而是用于表示与其他表的关联关系。

getEnumsJson方法

@Override  
public JSONObject getEnumsJson() {  
 JSONObject enumsJson = super.getEnumsJson();  
 AtWorkEnum[] enumsArr = AtWorkEnum.values();  
 for (AtWorkEnum enumsTemp : enumsArr) {  
 String varName = enumsTemp.toString();  
 String[] varNameArr = varName.split(ConstatFinalUtil.FINAL\_JAVA\_ENUMS\_SPLIT);  
 if (varNameArr.length == 2) {  
 /\* 条目 \*/  
 JSONObject jsonEntryTemp = new JSONObject();  
 jsonEntryTemp.put(ConstatFinalUtil.FINAL\_ENUMSVAL, varName);  
 jsonEntryTemp.put(ConstatFinalUtil.FINAL\_CODE, enumsTemp.getCode());  
 jsonEntryTemp.put(ConstatFinalUtil.FINAL\_INFO, enumsTemp.getInfo());  
  
 String jsonKey = varNameArr[0];  
 /\* 源数组 \*/  
 JSONArray jsonArrVal = enumsJson.getJSONArray(jsonKey);  
 if (Objects.isNull(jsonArrVal)) {  
 jsonArrVal = new JSONArray();  
 }  
 jsonArrVal.add(jsonEntryTemp);  
  
 /\* 存储最终结果 \*/  
 enumsJson.put(jsonKey, jsonArrVal);  
 }  
 }  
 return enumsJson;  
}

该方法重写了BasePojo中的getEnumsJson()方法。它会遍历AtWorkEnum枚举类中的所有枚举项，并将每个枚举项的code和info以JSON格式返回。

AtWorkEnum是与AtWork状态相关的枚举类，code表示状态码，info表示状态描述。

该方法会返回一个JSON对象，包含枚举项的值及其描述，用于前端展示或其它业务处理。

getStatusStr方法

public String getStatusStr() {  
 AtWorkEnum[] enumsArr = AtWorkEnum.values();  
 for (AtWorkEnum enumsTemp : enumsArr) {  
 String varName = enumsTemp.toString();  
 if (varName.startsWith("STATUS\_") && enumsTemp.getCode().equalsIgnoreCase(this.getStatus()) ) {  
 this.statusStr = enumsTemp.getInfo();  
 break;  
 }  
 }  
 return statusStr;  
}

getStatusStr方法根据AtWork实例的status字段（表示状态码）从 AtWorkEnum枚举类中查找对应的info（状态描述）。

它会遍历AtWorkEnum中的所有枚举项，找到与当前status匹配的枚举项，并返回相应的状态描述。

如果找到了对应的状态，就将其赋值给statusStr字段，并返回该状态描述。

#### ②AtWorkEnum.java

这部分代码是一个定义在Java中的枚举类 (Enum)。以下是对代码的详细讲解：

类的定义

public enum AtWorkEnum {

定义了一个枚举类AtWorkEnum，用于表示一组固定的常量。

枚举类型在Java中是一种特殊的类，用于定义一组有限且不可变的值（如状态、枚举项等）。

枚举常量的定义

STATUS\_ENABLE("1", "启用"),  
STATUS\_DISABLE("0", "禁用"),

枚举中定义的两个常量，分别表示 "启用" 和 "禁用" 两种状态。

每个枚举常量都对应两个参数：

code：状态码（如 "1" 和 "0"）。

info：状态描述（如 "启用" 和 "禁用"）。

枚举常量在定义时调用了枚举类的私有构造方法 AtWorkEnum(String code, String info) 来初始化每个枚举实例。

枚举的私有属性

private String code;  
private String info;

code：用于存储状态对应的编码（如 "1" 表示启用，"0" 表示禁用）。

Info：用于存储状态的文本描述（如 "启用" 和 "禁用"）。

这些属性与构造方法配合使用，用来初始化枚举常量。

枚举的构造方法

private AtWorkEnum(String code, String info) {  
 this.code = code;  
 this.info = info;  
}

构造方法是私有的，保证只能在枚举内部定义常量时调用，不能通过外部代码创建新的枚举实例。

this.code = code 和 this.info = info：初始化枚举实例时，将参数赋值给对应的属性。

Getter和Setter方法

public String getCode() {  
 return code;  
}  
  
public void setCode(String code) {  
 this.code = code;  
}  
  
public String getInfo() {  
 return info;  
}  
  
public void setInfo(String info) {  
 this.info = info;  
}

Getter方法：getCode()和getInfo()用于获取枚举实例的code和info值。

Setter方法：setCode()和setInfo()用于修改枚举实例的code和info值。

#### ③AtWorkQuery.java

这段代码定义了一个名为AtWorkQuery的类，主要用于封装查询条件，通常在业务逻辑层中，用来接收前端传来的查询参数或用于构建查询条件。

字段定义

private AtWork entity = new AtWork();

定义了一个AtWork类型的entity字段，并初始化为new AtWork()。

通常用于存储查询条件中的实体信息，例如具体的字段值（状态、时间等）。

private String selfObj;  
private String empTime;

selfObj：用于标识是否查询自己的数据（如登录用户）。

empTime：存储查询时的时间信息。

常量定义

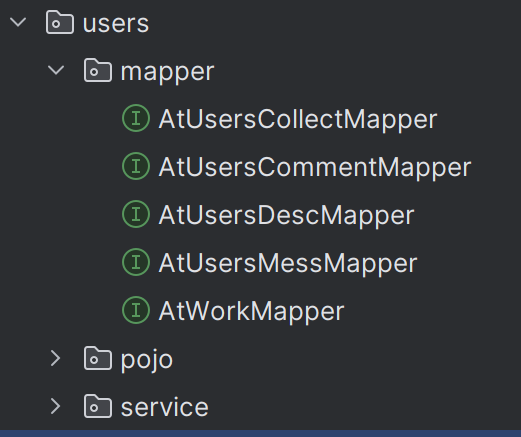
public static final String FINAL\_SELF\_OBJ\_TRUE = "true";

定义了一个静态常量 FINAL\_SELF\_OBJ\_TRUE，值为"true"。

通常用来表示是否查询当前用户自己的标识，避免使用硬编码。

## 7.Mapper（数据访问层）

### （1）文件位置



### （2）主要作用

Mapper层与数据库交互，执行具体的CRUD操作，通常包含基本的CRUD操作，如selectById、insert、update、delete。

用于执行数据库的操作，使用MyBatis来执行SQL语句，被Service层调用来执行数据库操作。

### （3）相关代码（以作品（work）相关代码AtWorkMapper.java为例，其余功能代码与work类似）

这段代码是一个MyBatis Plus的数据访问接口（Mapper），用于操作数据库中的AtWork表。扩展了BaseMapper接口，并定义了一些自定义的方法来实现更复杂的查询和操作逻辑。以下是代码的逐步解析：

查询多条记录（分页）

List<AtWork> findList(IPage<AtWork> page, @Param("param") AtWorkQuery query);

用途：根据AtWorkQuery条件查询多条记录，并支持分页。

参数：

IPage<AtWork> page：分页参数，MyBatis Plus提供的分页对象。

@Param("param") AtWorkQuery query：查询条件对象，通过@Param注解绑定到SQL中的参数param。

返回值：返回符合条件的AtWork对象列表。

查询多条记录（不分页）

List<AtWork> findList(@Param("param") AtWorkQuery query);

用途：根据AtWorkQuery条件查询所有符合条件的记录，不分页。

参数：仅有一个查询条件对象query。

返回值：返回符合条件的AtWork对象列表。

查询单条记录

AtWork findOne(@Param("param") AtWorkQuery query);

用途：根据AtWorkQuery条件查询单条记录。

参数：query表示查询条件。

返回值：返回符合条件的单条AtWork对象（期望查询结果只有一个）。

批量删除

int deleteBatch(@Param("param") AtWorkQuery query);

用途：根据AtWorkQuery条件批量删除记录。

参数：query表示删除的条件。

返回值：返回删除的记录条数（int）。

批量更新

int updateBatch(@Param("param") AtWorkQuery query);

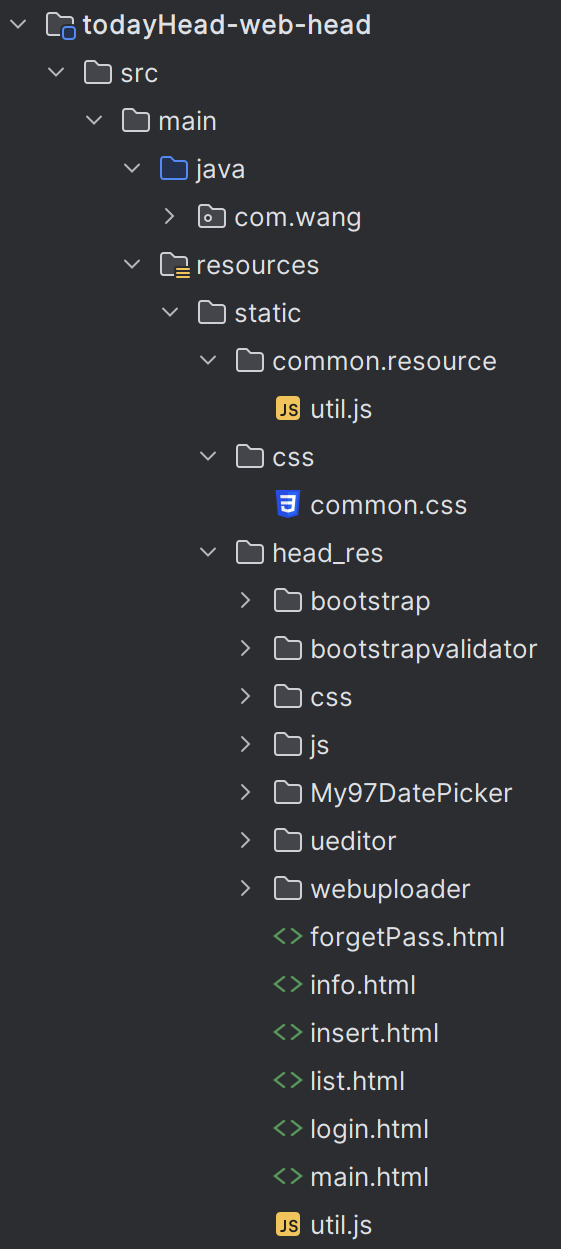
用途：根据AtWorkQuery条件批量更新记录。

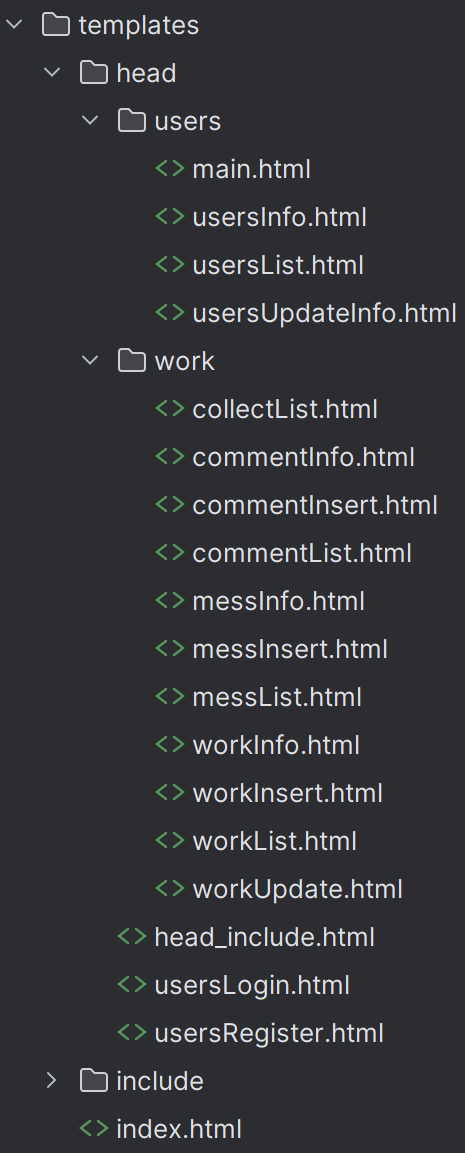
参数：query表示更新的条件和内容。

返回值：返回被更新的记录条数（int）。

# 八、前端代码实现和页面展示

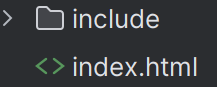
## 1.前端整体目录





## 2.跳转页面（index.html）

### （1）文件位置



### （2）相关代码

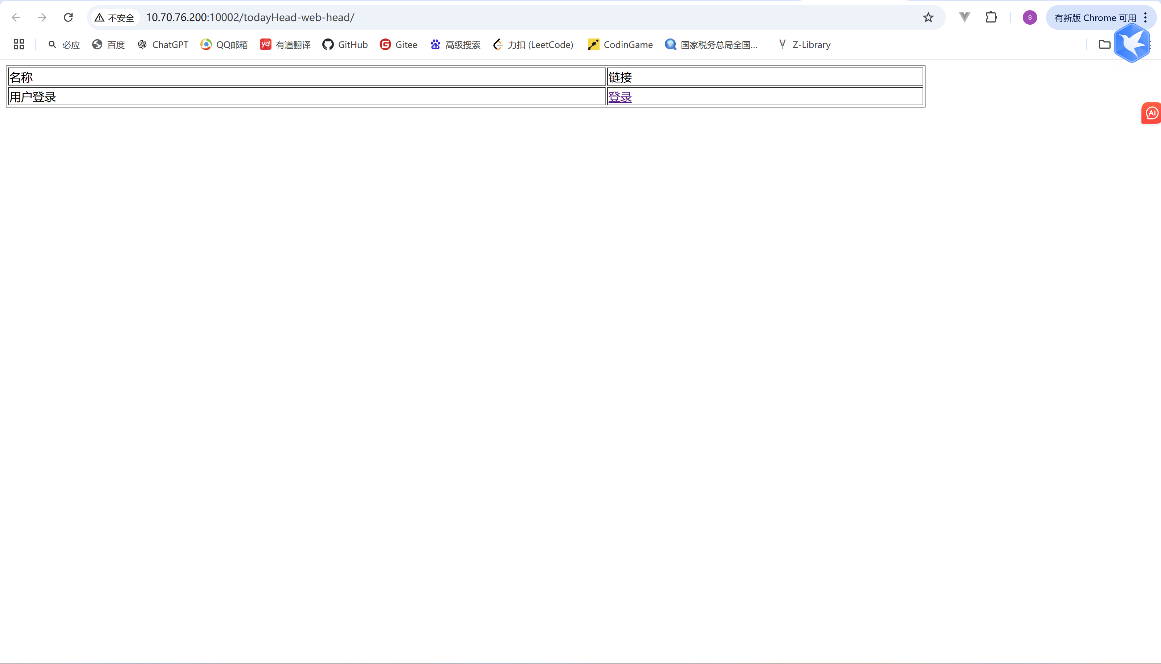
<html lang="en" xmlns:th="http://www.thymeleaf.org">  
<head>  
 <meta charset="UTF-8">  
 <title>今日头条(用户)-菜单-首页</title>  
</head>  
<body>  
<table border="1" width="80%">  
 <tr>  
 <td>名称</td>  
 <td>链接</td>  
 </tr>  
 <tr>  
 <td>用户登录</td>  
 <td><a th:href="@{/login}" target="\_blank">登录</a></td>  
 </tr>  
</table>  
</body>  
</html>

用途：根据AtWorkQuery条件批量更新记录。

参数：query表示更新的条件和内容。

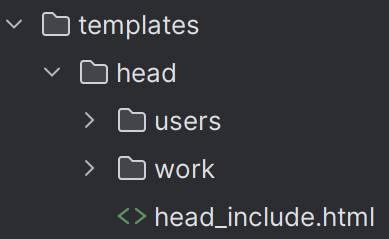
返回值：返回被更新的记录条数（int）。

### （3）页面展示



## 3.首页抬头（head\_include.html）

### （1）文件位置



### （2）相关代码

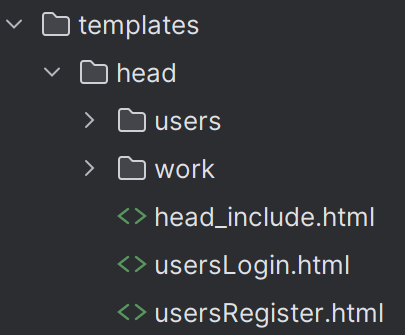
<html lang="en" xmlns:th="http:/www.thymeleaf.org">  
<head>  
 <meta charset="UTF-8">  
 <title>今日头条(用户)-前端首页</title>  
 <th:block th:fragment="title">  
 <meta charset="utf-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">  
 <link rel="stylesheet" type="text/css" th:href="@{/head\_res/bootstrap/css/bootstrap.min.css}" />  
 <link rel="stylesheet" type="text/css" th:href="@{/head\_res/css/main.css}" />  
 <link rel="stylesheet" type="text/css" th:href="@{/css/common.css}" />  
  
 <script type="text/javascript" th:src="@{/head\_res/js/jquery-3.3.1.min.js}"></script>  
 <script type="text/javascript" th:src="@{/head\_res/js/popper.min.js}"></script>  
 <script type="text/javascript" th:src="@{/head\_res/bootstrap/js/bootstrap.min.js}"></script>  
  
 <script type="text/javascript" th:src="@{/head\_res/My97DatePicker/WdatePicker.js}"></script>  
 <script type="text/javascript" th:src="@{/head\_res/util.js}"></script>  
 </th:block>  
</head>  
<body>  
<th:block th:fragment="header">  
 <!-- 网页头部开始 -->  
 <nav class="navbar navbar-expand-md navbar-light bg-light fixed-top">  
 <a class="navbar-brand" href="#">Logo</a>  
 <button class="navbar-toggler" type="button" data-toggle="collapse"  
 data-target="#navbarSupportedContent"  
 aria-controls="navbarSupportedContent" aria-expanded="false"  
 aria-label="Toggle navigation">  
 <span class="navbar-toggler-icon"></span>  
 </button>  
 <div class="collapse navbar-collapse" id="navbarSupportedContent">  
 <ul class="navbar-nav mr-auto">  
 <li class="nav-item active">  
 <a class="nav-link" th:href="@{/login}">首页  
 <span class="sr-only">(current)</span>  
 </a>  
 </li>  
 <th:block th:if="${session.usersSess != null}">  
 <li class="nav-item"><a class="nav-link" th:href="@{/head/users/usersList}">用户列表</a></li>  
 <li class="nav-item"><a class="nav-link" th:href="@{/head/work/workList}">作品列表(全部)</a></li>  
 <li class="nav-item"><a class="nav-link" th:href="@{/head/work/workList?selfObj=true}">作品列表(自己)</a></li>  
 <li class="nav-item"><a class="nav-link" th:href="@{/head/work/collectList}">收藏列表</a></li>  
 <li class="nav-item"><a class="nav-link" th:href="@{/head/work/commentList}">评论列表</a></li>  
 <li class="nav-item"><a class="nav-link" th:href="@{/head/work/messList?fromUsersFlag=true}">私信列表(发送)</a></li>  
 <li class="nav-item"><a class="nav-link" th:href="@{/head/work/messList?toUsersFlag=true}">私信列表(接收)</a></li>  
 </th:block>  
 <th:block th:if="${session.usersSess == null}">  
 <li class="nav-item"><a class="nav-link" th:href="@{/login}">登陆</a></li>  
 </th:block>  
 </ul>  
  
 <th:block th:if="${session.usersSess != null}">  
 <ul class="navbar-nav">  
 <li class="nav-item dropdown">  
 <a class="nav-link dropdown-toggle" href="#" id="navbarDropdownRight"  
 role="button" data-toggle="dropdown" aria-haspopup="true"  
 aria-expanded="false"> 个人中心  
 <span class="sr-only">(current)</span>  
 </a>  
 <div class="dropdown-menu" aria-labelledby="navbarDropdownRight">  
 <a class="dropdown-item" th:href="@{/head/users/usersUpdate}">个人信息</a>  
 <a class="dropdown-item" th:href="@{/head/users/usersUpdate?operType=updateInfo}">修改个人信息</a>  
 <a class="dropdown-item" th:href="@{/head/users/logout}" onclick="return confirm('确认退出吗?')">退出</a>  
 </div>  
 </li>  
 </ul>  
 </th:block>  
 </div>  
 </nav>  
</th:block>  
<th:block th:fragment="footer">  
 <th:block th:if="${response != null && response.info != null}">  
 <!-- 提示信息 -->  
 <div class="container">  
 <div class="col-sm-8 col-sm-offset-1">  
 <div class="alert alert-warning alert-dismissible fade show" role="alert">  
 <button type="button" class="close" data-dismiss="alert"  
 aria-label="Close">  
 <span aria-hidden="true">×</span>  
 </button>  
 <span th:text="${response?.info}"></span>  
 </div>  
 </div>  
 </div>  
 </th:block>  
  
 <footer class="footer">  
 <div class="container">  
 <div class="row">  
 <div class="col-sm-11 col-lg-6">  
 <h4>  
 </h4>  
 <p>  
 本网站所列开源项目的中文版文档全部由<a href="#">今日头条(用户)</a>顶力开发 <a href="#"  
 target="\_blank">CC BY 3.0</a>协议发布。  
 </p>  
 </div>  
 <div class="col-sm-11 col-lg-6">  
 <div class="row">  
 <div class="col-sm-3">  
 <ul class="list-unstyled">  
 <li class="mb-1">关于</li>  
 <li class=""><a href="#" class="text-muted">关于我们</a></li>  
 <li class=""><a href="#" class="text-muted">广告合作</a></li>  
 <li class=""><a href="#" class="text-muted">友情链接</a></li>  
 <li class=""><a href="#" class="text-muted">招聘</a></li>  
 </ul>  
 </div>  
 <div class="col-sm-3">  
 <ul class="list-unstyled">  
 <li class="mb-1">联系方式</li>  
 <li class=""><a href="#" class="text-muted">新浪微博</a></li>  
 <li class=""><a href="#" class="text-muted">电子邮件</a></li>  
 </ul>  
 </div>  
 <div class="col-sm-3">  
 <ul class="list-unstyled">  
 <li class="mb-1">旗下网站</li>  
 <li class=""><a href="#" class="text-muted">Laravel中文网</a></li>  
 <li class=""><a href="#" class="text-muted">Ghost中国</a></li>  
 </ul>  
 </div>  
 <div class="col-sm-3">  
 <ul class="list-unstyled">  
 <li class="mb-1">赞助商</li>  
 <li class=""><a href="#" class="text-muted">UCloud</a></li>  
 <li class=""><a href="#" class="text-muted">又拍云</a></li>  
 </ul>  
 </div>  
 </div>  
 </div>  
 </div>  
 <hr>  
 </div>  
 </footer>  
</th:block>  
</body>  
</html>

### （3）页面展示



## 4.登录和注册（userLogin.html和userRegister.html）

### （1）文件位置



### （2）相关代码

#### ①userLogin.html

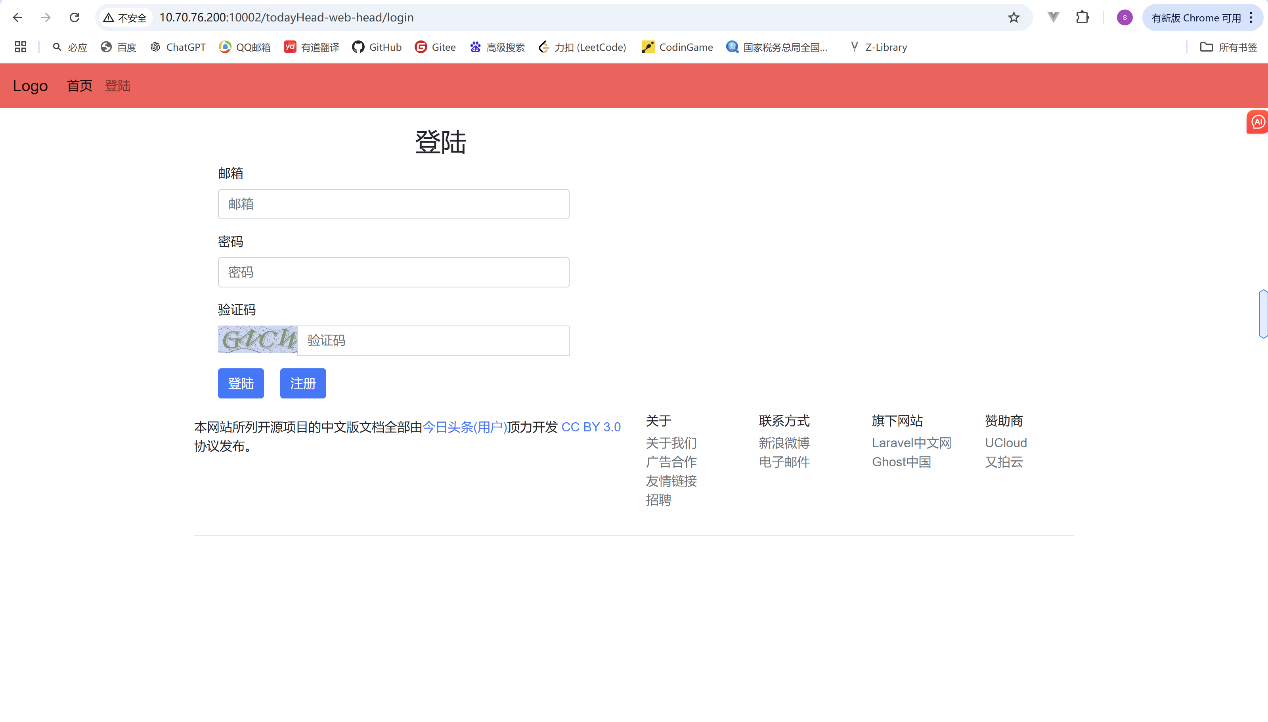
<!DOCTYPE html>  
<html lang="zh-CN" xmlns:th="http://www.thymeleaf.org">  
<head>  
 <title>登陆 - 今日头条(用户)</title>  
 <meta th:replace="~{/head/head\_include :: title}"/>  
 <link rel="stylesheet" th:href="@{/head\_res/bootstrapvalidator/css/form-validation.css}">  
 <script type="text/javascript" th:src="@{/head\_res/bootstrapvalidator/js/form-validation.js}"></script>  
 <script type="text/javascript">  
 $(function()  
 {  
 $("#form").FormValidation({  
 fields:  
 {  
 /\* 邮箱 \*/  
 email:  
 {  
 validators: {  
 notEmpty: {  
 message: '邮箱不能为空'  
 },  
 stringLength: {  
 min: 4,  
 max:40,  
 message: '邮箱的长度为在4位与40位之间'  
 },  
 regexp: {  
 regexp: /^[\w-\_]+@[\w-\_]+\.[\w-\_]+$/i,  
 message: '邮箱不符合规则'  
 }  
 }  
 },  
 /\* 密码 \*/  
 password:  
 {  
 validators: {  
 notEmpty: {  
 message: '密码不能为空'  
 },  
 stringLength: {  
 min: 4,  
 max:40,  
 message: '密码的长度为在4位与40位之间'  
 }  
 }  
 },  
 /\* 验证码 \*/  
 code:  
 {  
 validators: {  
 notEmpty: {  
 message: '验证码不能为空'  
 },  
 stringLength: {  
 min: 4,  
 max:4,  
 message: '验证码的长度为在4位字符'  
 }  
 }  
 }  
 }  
 });  
 });  
 </script>  
</head>  
<body>  
<meta th:replace="~{/head/head\_include :: header}"/>  
<!-- 网页的主体 -->  
<div class="container">  
 <div class="col-md-8 col-sm-10">  
 <form id="form" class="form-horizontal" method="post"  
 th:action="@{/loginSubmit}">  
 <input type="hidden" name="returnUrl" value="">  
  
 <h2 class="col-sm-10 col-sm-offset-1 form-signin-heading text-center">登陆</h2>  
 <div class="form-group">  
 <label for="email" class="col-sm-3 control-label">邮箱</label>  
 <div class="col-sm-8">  
 <input type="text" id="email" name="email" class="form-control"  
 value="" placeholder="邮箱">  
 </div>  
 </div>  
 <div class="form-group">  
 <label for="inputPassword" class="col-sm-3 control-label">密码</label>  
 <div class="col-sm-8">  
 <input type="password" name="password" id="inputPassword"  
 class="form-control" placeholder="密码"/>  
 </div>  
 </div>  
 <div class="form-group">  
 <label for="email" class="col-sm-3 control-label">验证码</label>  
 <div class="col-sm-8">  
 <div class="input-group">  
 <span class="input-group-addon"> <a href="#" tabindex="-1"  
 onclick="return refreshCode('imgCode');"> <img  
 id="imgCode" width="100" height="35"  
 th:src="@{/common/imgCode}" title="点击刷新验证码" />  
 </a>  
 </span>  
 <input type="text" id="code" name="code" class="form-control"  
 placeholder="验证码">  
 </div>  
 </div>  
 </div>  
 <div class="form-group">  
 <div class="col-sm-8 col-sm-offset-2">  
 <button class="btn btn-md btn-primary" type="submit">登陆</button>  
 <a th:href="@{/register}" class="btn btn-md btn-primary ml-3">注册</a>  
 </div>  
 </div>  
 </form>  
 </div>  
</div>  
<meta th:replace="~{/head/head\_include :: footer}"/>  
</body>  
</html>

#### ②userRegister.html

<!DOCTYPE html>  
<html lang="zh-CN" xmlns:th="http://www.thymeleaf.org">  
<head>  
 <title>注册 - 今日头条(用户)</title>  
 <meta th:replace="~{/head/head\_include :: title}"/>  
 <link rel="stylesheet" th:href="@{/head\_res/bootstrapvalidator/css/form-validation.css}">  
 <script type="text/javascript" th:src="@{/head\_res/bootstrapvalidator/js/form-validation.js}"></script>  
 <script type="text/javascript">  
 $(function()  
 {  
 $("#form").FormValidation({  
 fields:  
 {  
 /\* 邮箱 \*/  
 email:  
 {  
 validators: {  
 notEmpty: {  
 message: '邮箱不能为空'  
 },  
 stringLength: {  
 min: 4,  
 max:40,  
 message: '邮箱的长度为在4位与40位之间'  
 },  
 regexp: {  
 regexp: /^[\w-\_]+@[\w-\_]+\.[\w-\_]+$/i,  
 message: '邮箱不符合规则'  
 }  
 }  
 },  
 /\* 密码 \*/  
 password:  
 {  
 validators: {  
 notEmpty: {  
 message: '密码不能为空'  
 },  
 stringLength: {  
 min: 4,  
 max:40,  
 message: '密码的长度为在4位与40位之间'  
 }  
 }  
 },  
 /\* 确认密码 \*/  
 confirmPass:  
 {  
 validators: {  
 notEmpty: {  
 message: '确认密码不能为空'  
 },  
 stringLength: {  
 min: 4,  
 max:40,  
 message: '确认密码的长度为在4位与40位之间'  
 },  
 equalsTo :  
 {  
 /\* 比较密码的id \*/  
 name:"password",  
 message: '两次密码输入不一样'  
 }  
 }  
 },  
 /\* 验证码 \*/  
 code:  
 {  
 validators: {  
 notEmpty: {  
 message: '验证码不能为空'  
 },  
 stringLength: {  
 min: 4,  
 max:4,  
 message: '验证码的长度为在4位字符'  
 }  
 }  
 }  
 }  
 });  
 });  
 </script>  
</head>  
<body>  
<meta th:replace="~{/head/head\_include :: header}"/>  
<!-- 网页的主体 -->  
<div class="container">  
 <div class="col-md-8 col-sm-10">  
 <form id="form" class="form-horizontal" method="post"  
 th:action="@{/registerSubmit}">  
 <input type="hidden" name="returnUrl" value="">  
  
 <h2 class="col-sm-10 col-sm-offset-1 form-signin-heading text-center">注册</h2>  
 <div class="form-group">  
 <label for="email" class="col-sm-3 control-label">邮箱</label>  
 <div class="col-sm-8">  
 <input type="text" id="email" name="email" class="form-control"  
 value="" placeholder="邮箱">  
 </div>  
 </div>  
 <div class="form-group">  
 <label for="inputPassword" class="col-sm-3 control-label">密码</label>  
 <div class="col-sm-8">  
 <input type="password" name="password" id="inputPassword"  
 class="form-control" placeholder="密码"/>  
 </div>  
 </div>  
 <div class="form-group">  
 <label for="inputPassword" class="col-sm-3 control-label">确认密码</label>  
 <div class="col-sm-8">  
 <input type="password" name="confirmPass" id="confirmPass"  
 class="form-control" placeholder="确认密码"/>  
 </div>  
 </div>  
 <div class="form-group">  
 <label for="email" class="col-sm-3 control-label">验证码</label>  
 <div class="col-sm-8">  
 <div class="input-group">  
 <span class="input-group-addon"><a href="#" tabindex="-1"  
 onclick="return refreshCode('imgCode');"> <img  
 id="imgCode" width="100" height="35"  
 th:src="@{/common/imgCode}" title="点击刷新验证码" />  
 </a>  
 </span>  
 <input type="text" id="code" name="code" class="form-control"  
 placeholder="验证码">  
 </div>  
 </div>  
 </div>  
 <div class="form-group">  
 <div class="col-sm-8 col-sm-offset-2">  
 <button class="btn btn-md btn-primary" type="submit">注册</button>  
 <a th:href="@{/login}" class="btn btn-md btn-primary ml-3">登录</a>  
 </div>  
 </div>  
 </form>  
 </div>  
</div>  
<meta th:replace="~{/head/head\_include :: footer}"/>  
</body>  
</html>

### （3）页面展示

#### ①登录界面

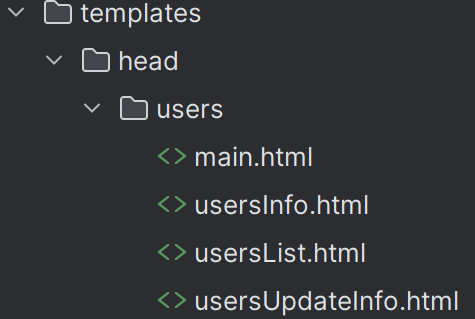


#### ②注册界面



## 5.用户部分

### （1）文件位置



### （2）相关代码

#### ①main.html

<!DOCTYPE html>  
<html lang="zh-CN" xmlns:th="http://www.thymeleaf.org">  
<head>  
 <title>登录后首页 - 今日头条(用户)</title>  
 <meta th:replace="~{/head/head\_include :: title}"/>  
 <link rel="stylesheet" th:href="@{/head\_res/bootstrapvalidator/css/form-validation.css}">  
 <script type="text/javascript" th:src="@{/head\_res/bootstrapvalidator/js/form-validation.js}"></script>  
</head>  
<body>  
<meta th:replace="~{/head/head\_include :: header}"/>  
<div class="container">  
 <div class="row">  
 <div class="col-sm-8 col-sm-offset-1 main">  
 <h2 class="col-sm-10 col-sm-offset-1 form-signin-heading text-center">查看个人信息</h2>  
 <input type="hidden" name="operType" value="update">  
 <div class="form-group row">  
 <label for="email" class="col-sm-2 col-form-label">邮箱:</label>  
 <div class="col-sm-8">  
 <span th:text="${one.email}"></span>  
 </div>  
 </div>  
 <div class="form-group row">  
 <label for="email" class="col-sm-2 col-form-label">名称:</label>  
 <div class="col-sm-8">  
 <span th:text="${one.name}"></span>  
 </div>  
 </div>  
 <div class="form-group row">  
 <label for="email" class="col-sm-2 col-form-label">昵称:</label>  
 <div class="col-sm-8">  
 <span th:text="${one.nickName}"></span>  
 </div>  
 </div>  
 <div class="form-group row">  
 <label for="email" class="col-sm-2 col-form-label">手机号:</label>  
 <div class="col-sm-8">  
 <span th:text="${one.phone}"></span>  
 </div>  
 </div>  
 <div class="form-group row">  
 <label for="email" class="col-sm-2 col-form-label">性别:</label>  
 <div class="col-sm-8">  
 <span th:text="${one.sexStr}"></span>  
 </div>  
 </div>  
 </div>  
 </div>  
</div>  
<meta th:replace="~{/head/head\_include :: footer}"/>  
<!--Thymeleaf 会从 head/head\_include 文件中提取名为 footer 的模板片段，并将该片段的内容替换掉当前 meta 标签的内容。-->  
</body>  
</html>

#### ②usersInfo.html

<!DOCTYPE html>  
<html lang="zh-CN" xmlns:th="http://www.thymeleaf.org">  
<head>  
 <title>登录后首页 - 今日头条(用户)</title>  
 <meta th:replace="~{/head/head\_include :: title}"/>  
 <link rel="stylesheet" th:href="@{/head\_res/bootstrapvalidator/css/form-validation.css}">  
 <script type="text/javascript" th:src="@{/head\_res/bootstrapvalidator/js/form-validation.js}"></script>  
</head>  
<body>  
<meta th:replace="~{/head/head\_include :: header}"/>  
<div class="container">  
 <div class="row">  
 <div class="col-sm-8 col-sm-offset-1 main">  
 <h2 class="col-sm-10 col-sm-offset-1 form-signin-heading text-center">查看个人信息</h2>  
 <input type="hidden" name="operType" value="update">  
 <div class="form-group row">  
 <label for="email" class="col-sm-2 col-form-label">邮箱:</label>  
 <div class="col-sm-8">  
 <span th:text="${one.email}"></span>  
 </div>  
 </div>  
 <div class="form-group row">  
 <label for="email" class="col-sm-2 col-form-label">名称:</label>  
 <div class="col-sm-8">  
 <span th:text="${one.name}"></span>  
 </div>  
 </div>  
 <div class="form-group row">  
 <label for="email" class="col-sm-2 col-form-label">昵称:</label>  
 <div class="col-sm-8">  
 <span th:text="${one.nickName}"></span>  
 </div>  
 </div>  
 <div class="form-group row">  
 <label for="email" class="col-sm-2 col-form-label">手机号:</label>  
 <div class="col-sm-8">  
 <span th:text="${one.phone}"></span>  
 </div>  
 </div>  
 <div class="form-group row">  
 <label for="email" class="col-sm-2 col-form-label">性别:</label>  
 <div class="col-sm-8">  
 <span th:text="${one.sexStr}"></span>  
 </div>  
 </div>  
 </div>  
 </div>  
</div>  
<meta th:replace="~{/head/head\_include :: footer}"/>  
</body>  
</html>

#### ③usersList.html

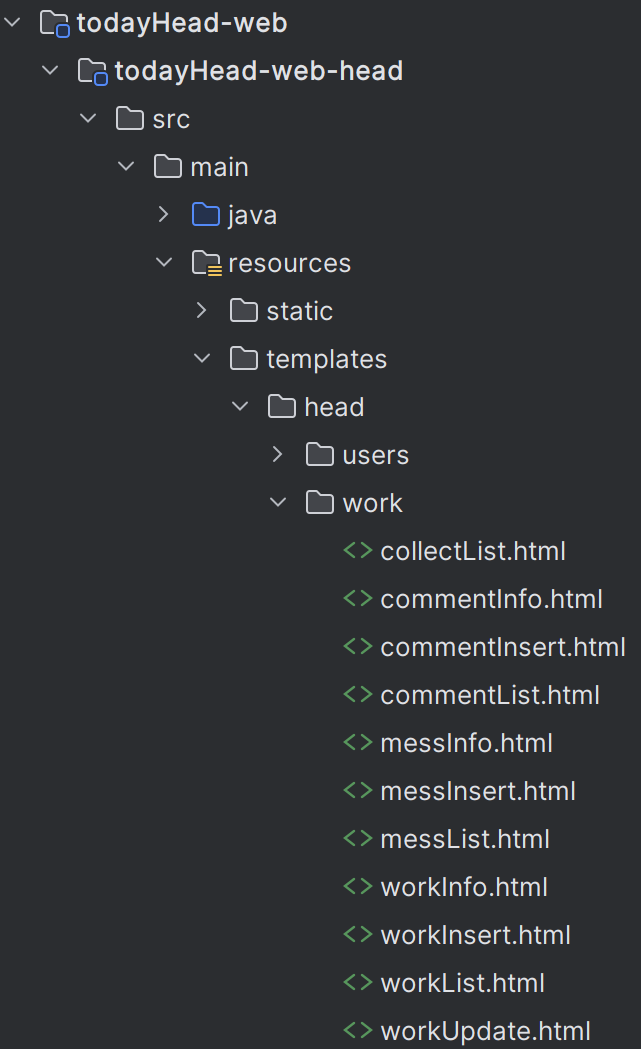
<!DOCTYPE html>  
<html lang="zh-CN" xmlns:th="http://www.thymeleaf.org">  
<head>  
 <title>用户列表 - 今日头条(用户)</title>  
 <meta th:replace="~{/head/head\_include :: title}"/>  
</head>  
<body>  
<meta th:replace="~{/head/head\_include :: header}"/>  
<!-- 网页的主体部分 -->  
<div class="container">  
 <div class="row">  
 <div class="col-md-11 col-sm-offset-1">  
 <h2 class="sub-header">用户列表</h2>  
 <form th:action="@{/head/users/usersList}" method="post" class="form-inline">  
 <input type="hidden" name="dynastyId" value="${requestScope.dynastyId }">  
 </form>  
 </div>  
 <div class="col-md-11 col-sm-offset-1">  
 <div class="table-responsive mt-2">  
 <table class="table table-striped table-hover">  
 <thead>  
 <tr>  
 <th>主键</th>  
 <th>名称</th>  
 <th>昵称</th>  
 <th>邮箱</th>  
 <th>手机</th>  
 <th>创建时间</th>  
 <th>更新时间</th>  
 <th>操作</th>  
 </tr>  
 </thead>  
 <tbody>  
 <th:block th:each="oneTemp : ${dataList }">  
 <tr>  
 <td th:title="${oneTemp.id}" th:text="${oneTemp.id.length() > 5 ? oneTemp.id.substring(0,5) : oneTemp.id}"></td>  
 <td class="text-l" th:title="${oneTemp.name}">  
 <a title="用户查看"  
 th:href="@{/head/users/usersUpdate('entity.id'=${oneTemp.id })}"  
 th:text="${oneTemp.name.length() > 5 ? oneTemp.name.substring(0,5) : oneTemp.name}" target="\_blank">  
 </a>  
 </td>  
 <td class="text-l" th:title="${oneTemp.nickName}">  
 <span  
 th:text="${oneTemp.nickName.length() > 5 ? oneTemp.nickName.substring(0,5) : oneTemp.nickName}">  
 </span>  
 </td>  
 <td class="text-l" th:title="${oneTemp.email}">  
 <span  
 th:text="${oneTemp.email.length() > 5 ? oneTemp.email.substring(0,5) : oneTemp.email}">  
 </span>  
 </td>  
 <td class="text-l" th:title="${oneTemp.phone}">  
 <span  
 th:text="${oneTemp.phone.length() > 5 ? oneTemp.phone.substring(0,5) : oneTemp.phone}">  
 </span>  
 </td>  
 <td th:text="${#dates.format(oneTemp.createTime,'yyyy-MM-dd HH:mm:ss')}"></td>  
 <td th:text="${#dates.format(oneTemp.updateTime,'yyyy-MM-dd HH:mm:ss')}"></td>  
 <td>  
 <a data-title="发私信" title="发私信" target="\_blank"  
 th:href="@{/head/work/messInsert('entity.toUsersId'=${oneTemp.id })}"  
 >发私信</a>  
 </td>  
 </tr>  
 </c:forEach>  
 </tbody>  
 </table>  
 </div>  
 </div>  
 <div class="col-md-11 col-sm-offset-1">  
 <form id="pageForm" th:action="@{/head/users/usersList}"  
 method="post" class="form-inline col-md-11 col-sm-offset-1">  
 <!-- 设置一些参数 -->  
 <input type="hidden" name="st" value="${requestScope.st }">  
 <input type="hidden" name="ed" value="${requestScope.ed }">  
  
 <nav aria-label="Page navigation example">  
 <ul class="pagination justify-content-end">  
 <li class="page-item">  
 <a href="#" class="page-link"  
 th:onclick="return pageFormSubmit('pageForm','current','1','size','[[${pageInfoUtil?.size}]]')"  
 >  
 <span>首页</span>  
 </a>  
 </li>  
 <li class="page-item">  
 <a href="#" class="page-link"  
 th:onclick="return pageFormSubmit('pageForm','current','[[${pageInfoUtil?.current - 1}]]','size','[[${pageInfoUtil?.size}]]')"  
 >  
 <span>上一页</span>  
 </a>  
 </li>  
 <li class="page-item">  
 <a href="#" class="page-link"  
 th:onclick="return pageFormSubmit('pageForm','current','[[${pageInfoUtil?.total}]]','size','[[${pageInfoUtil?.size}]]')"  
 >  
 <span>下一页</span>  
 </a>  
 </li>  
 <li class="page-item">  
 <a href="#" class="page-link"  
 onclick="return pageFormSubmit('pageForm', 'currentPage', '${pageInfoUtil.totalPage}', 'pageSize','${pageInfoUtil.pageSize}')"  
 >  
 <span>尾页</span>  
 </a>  
 </li>  
 <li class="page-item">  
 共<span th:text="${pageInfoUtil?.pages}"></span>页，  
 共<span th:text="${pageInfoUtil?.total }"></span>条  
 </li>  
 <li class="page-item">  
 第<input type="text" id="current" name="current"  
 th:value="${pageInfoUtil?.current}" class="form-control"  
 size="5" maxlength="5"/>页  
 </li>  
 <li class="page-item">  
 <input type="text" id="size" name="size"  
 th:value="${pageInfoUtil?.size}" class="form-control" size="5" maxlength="5"/>  
 </li>  
 <li class="page-item"><input type="submit" value="GO" class="form-control btn btn-primary"/></li>  
 </ul>  
 </nav>  
 </form>  
 </div>  
 </div>  
</div>  
<meta th:replace="~{/head/head\_include :: footer}"/>  
</body>  
</html>

#### ④usersUpdateInfo.html

<!DOCTYPE html>  
<html lang="zh-CN" xmlns:th="http://www.thymeleaf.org">  
<head>  
 <title>修改个人信息 - 今日头条(用户)</title>  
 <meta th:replace="~{/head/head\_include :: title}"/>  
 <link rel="stylesheet" th:href="@{/head\_res/bootstrapvalidator/css/form-validation.css}">  
 <script type="text/javascript" th:src="@{/head\_res/bootstrapvalidator/js/form-validation.js}"></script>  
 <script type="text/javascript" charset="utf-8" th:src="@{/head\_res/ueditor/ueditor.config.js}"></script>  
 <script type="text/javascript" charset="utf-8" th:src="@{/head\_res/ueditor/ueditor.all.min.js}"> </script>  
 <script type="text/javascript" charset="utf-8" th:src="@{/head\_res/ueditor/lang/zh-cn/zh-cn.js}"></script>  
 <script type="text/javascript" th:inline="javascript">  
 $(function()  
 {  
 $("#form").FormValidation({  
 fields:  
 {  
 /\* 邮箱 \*/  
 name:  
 {  
 validators: {  
 notEmpty: {  
 message: '名称不能为空'  
 },  
 stringLength: {  
 min: 4,  
 max:40,  
 message: '名称的长度为在4位与40位之间'  
 }  
 }  
 }  
 }  
 });  
  
 var ue = *UE*.getEditor('editor');  
 var content = [[${one?.content}]];  
 /\* 百度编辑器提供的方法 \*/  
 ue.addListener("ready", function () {  
 /\* 为百度编辑器赋值 \*/  
 ue.setContent(content);  
 });  
 });  
 </script>  
</head>  
<body>  
<meta th:replace="~{/head/head\_include :: header}"/>  
<!-- 网页的主体 -->  
<div class="container">  
 <div class="col-md-8 col-sm-10">  
 <form id="form" class="form-horizontal" method="post"  
 th:action="@{/head/users/usersUpdateSubmit}">  
 <input type="hidden" name="entity.id" th:value="${one.id}">  
 <input type="hidden" name="operType" th:value="${paramQuery.operType}">  
  
 <h2 class="col-sm-10 col-sm-offset-1 form-signin-heading text-center">更新用户信息</h2>  
 <div class="form-group">  
 <label for="email" class="col-sm-3 control-label">邮箱</label>  
 <div class="col-sm-8">  
 <input type="text" id="email" name="entity.email" class="form-control"  
 th:value="${one.email}" placeholder="邮箱">  
 </div>  
 </div>  
 <div class="form-group">  
 <label for="name" class="col-sm-3 control-label">名称</label>  
 <div class="col-sm-8">  
 <input type="text" id="name" name="entity.name" class="form-control"  
 th:value="${one.name}" placeholder="名称">  
 </div>  
 </div>  
 <div class="form-group">  
 <label for="nickName" class="col-sm-3 control-label">昵称</label>  
 <div class="col-sm-8">  
 <input type="text" id="nickName" name="entity.nickName" class="form-control"  
 th:value="${one.nickName}" placeholder="名称">  
 </div>  
 </div>  
 <div class="form-group">  
 <label for="phone" class="col-sm-3 control-label">手机</label>  
 <div class="col-sm-8">  
 <input type="text" id="phone" name="entity.phone" class="form-control"  
 th:value="${one.phone}" placeholder="手机">  
 </div>  
 </div>  
 <div class="form-group">  
 <label for="sex" class="col-sm-2 control-label">性别:</label>  
 </div>  
 <div class="form-group">  
 <div class="col-sm-8 form-check form-check-inline">  
 &nbsp;&nbsp;&nbsp;&nbsp;  
 <!-- 拆分,按照-拆分,取最后一个 -->  
 <input type="radio" id="sex0" name="entity.sex" class="form-check-input" value="0" th:checked="${one.sex == '0'}">  
 <label for="sex0" class="form-check-label">未知</label>  
 &nbsp;&nbsp;  
 <!-- 拆分,按照-拆分,取最后一个 -->  
 <input type="radio" id="sex1" name="entity.sex" class="form-check-input" value="1" th:checked="${one.sex == '1'}">  
 <label for="sex1" class="form-check-label">男</label>  
 &nbsp;&nbsp;  
 <input type="radio" id="sex2" name="entity.sex" class="form-check-input" value="2" th:checked="${one.sex == '2'}">  
 <label for="sex1" class="form-check-label">女</label>  
 </div>  
 </div>  
 <div class="form-group">  
 <div class="col-sm-8 col-sm-offset-2">  
 <button class="btn btn-md btn-primary" type="submit">提交</button>  
 </div>  
 </div>  
 </form>  
 </div>  
</div>  
<meta th:replace="~{/head/head\_include :: footer}"/>  
</body>  
</html>

## 6.功能部分（以作品（work）相关代码为例，其余功能前端代码与work类似）

### （1）文件位置



### （2）作品（work）相关代码

#### ①workList.html

<!DOCTYPE html>  
<html lang="zh-CN" xmlns:th="http://www.thymeleaf.org">  
<head>  
 <title>作品列表 - 今日头条(用户)</title>  
 <meta th:replace="~{/head/head\_include :: title}"/>  
</head>  
<body>  
<meta th:replace="~{/head/head\_include :: header}"/>  
<!-- 网页的主体部分 -->  
<div class="container">  
 <div class="row">  
 <div class="col-md-11 col-sm-offset-1">  
 <h2 class="sub-header">作品列表<span th:if="${paramQuery.selfObj == 'true'}">(查询自己)</span></h2>  
 <form th:action="@{/head/work/workList}" method="post" class="form-inline">  
 <input type="hidden" name="dynastyId" value="${requestScope.dynastyId }">  
 <a th:if="${paramQuery.selfObj == 'true'}" th:href="@{/head/work/workInsert}" target="\_blank" class="btn btn-primary ml-2">添加</a>  
 </form>  
 </div>  
 <div class="col-md-11 col-sm-offset-1">  
 <div class="table-responsive mt-2">  
 <table class="table table-striped table-hover">  
 <thead>  
 <tr>  
 <th>主键</th>  
 <th>名称</th>  
 <th>浏览量</th>  
 <th>作者</th>  
 <th>创建时间</th>  
 <th>更新时间</th>  
 <th>操作</th>  
 </tr>  
 </thead>  
 <tbody>  
 <th:block th:each="oneTemp : ${dataList }">  
 <tr>  
 <td th:title="${oneTemp.id}" th:text="${oneTemp.id.length() > 5 ? oneTemp.id.substring(0,5) : oneTemp.id}"></td>  
 <td class="text-l" th:title="${oneTemp.name}">  
 <a title="作品查看"  
 th:href="@{/head/work/workUpdate('entity.id'=${oneTemp.id })}"  
 th:text="${oneTemp.name.length() > 5 ? oneTemp.name.substring(0,5) : oneTemp.name}" target="\_blank">  
 </a>  
 </td>  
 <td class="td-status">  
 <span class="label label-success radius">  
 <span class="label label-success radius" th:text="${oneTemp?.viewCount}">  
 </span>  
 </span>  
 </td>  
 <td class="text-l" th:title="${oneTemp.users?.nickName}">  
 <a title="作者查看"  
 th:href="@{/head/users/usersUpdate('entity.id'=${oneTemp.usersId })}"  
 th:text="${oneTemp.users?.nickName.length() > 5 ? oneTemp.users?.nickName.substring(0,5) : oneTemp.users?.nickName}" target="\_blank">  
 </a>  
 </td>  
 <td th:text="${#dates.format(oneTemp.createTime,'yyyy-MM-dd HH:mm:ss')}"></td>  
 <td th:text="${#dates.format(oneTemp.updateTime,'yyyy-MM-dd HH:mm:ss')}"></td>  
 <td>  
 <a data-title="作品更新" title="作品更新" target="\_blank"  
 th:href="@{/head/work/workUpdate(operType='updateInfo','entity.id'=${oneTemp.id })}"  
 >更新</a>  
 </td>  
 </tr>  
 </c:forEach>  
 </tbody>  
 </table>  
 </div>  
 </div>  
 <div class="col-md-11 col-sm-offset-1">  
 <form id="pageForm" th:action="@{/head/work/workList}"  
 method="post" class="form-inline col-md-11 col-sm-offset-1">  
 <!-- 设置一些参数 -->  
 <input type="hidden" name="st" value="${requestScope.st }">  
 <input type="hidden" name="ed" value="${requestScope.ed }">  
  
 <nav aria-label="Page navigation example">  
 <ul class="pagination justify-content-end">  
 <li class="page-item">  
 <a href="#" class="page-link"  
 th:onclick="return pageFormSubmit('pageForm','current','1','size','[[${pageInfoUtil?.size}]]')"  
 >  
 <span>首页</span>  
 </a>  
 </li>  
 <li class="page-item">  
 <a href="#" class="page-link"  
 th:onclick="return pageFormSubmit('pageForm','current','[[${pageInfoUtil?.current - 1}]]','size','[[${pageInfoUtil?.size}]]')"  
 >  
 <span>上一页</span>  
 </a>  
 </li>  
 <li class="page-item">  
 <a href="#" class="page-link"  
 th:onclick="return pageFormSubmit('pageForm','current','[[${pageInfoUtil?.total}]]','size','[[${pageInfoUtil?.size}]]')"  
 >  
 <span>下一页</span>  
 </a>  
 </li>  
 <li class="page-item">  
 <a href="#" class="page-link"  
 onclick="return pageFormSubmit('pageForm', 'currentPage', '${pageInfoUtil.totalPage}', 'pageSize','${pageInfoUtil.pageSize}')"  
 >  
 <span>尾页</span>  
 </a>  
 </li>  
 <li class="page-item">  
 共<span th:text="${pageInfoUtil?.pages}"></span>页，  
 共<span th:text="${pageInfoUtil?.total }"></span>条  
 </li>  
 <li class="page-item">  
 第<input type="text" id="current" name="current"  
 th:value="${pageInfoUtil?.current}" class="form-control"  
 size="5" maxlength="5"/>页  
 </li>  
 <li class="page-item">  
 <input type="text" id="size" name="size"  
 th:value="${pageInfoUtil?.size}" class="form-control" size="5" maxlength="5"/>  
 </li>  
 <li class="page-item"><input type="submit" value="GO" class="form-control btn btn-primary"/></li>  
 </ul>  
 </nav>  
 </form>  
 </div>  
 </div>  
</div>  
<meta th:replace="~{/head/head\_include :: footer}"/>  
</body>  
</html>

#### ②workInfo.html

<!DOCTYPE html>  
<html lang="zh-CN" xmlns:th="http://www.thymeleaf.org">  
<head>  
 <title>作品列表 - 今日头条(用户)</title>  
 <meta th:replace="~{/head/head\_include :: title}"/>  
 <script type="text/javascript">  
 /\* 作品收藏 要发送ajax请求 \*/  
 function collectInsert(){  
 *console*.log("collectInsert,添加");  
 if(!*window*.confirm('确认要操作吗?')) {  
 return false;  
 }  
 var workId = $("#workIdSpan").html();  
 $.post(*rootPath* + "/head/work/collectInsertSubmit", {  
 "entity.workId": workId  
 }, function(result){  
 *console*.info("返回结果:" , result);  
 alert(result.info)  
 }, "json");  
 return false  
 }  
 </script>  
</head>  
<body>  
<meta th:replace="~{/head/head\_include :: header}"/>  
<div class="container">  
 <div class="row">  
 <div class="col-sm-8 col-sm-offset-1 main">  
 <h2 class="col-sm-10 col-sm-offset-1 form-signin-heading text-center">查看作品</h2>  
 <a class="btn btn-primary ml-2" href="#" onclick="return collectInsert()">收藏</a>  
 <a th:href="@{/head/work/commentInsert(entity.workId=${one.id})}" target="\_blank" class="btn btn-primary ml-2">评论</a>  
 <div class="form-group row">  
 <label for="email" class="col-sm-2 control-label">Id</label>  
 <div class="col-sm-8">  
 <span id="workIdSpan" th:text="${one?.id }"></span>  
 </div>  
 </div>  
 <div class="form-group row">  
 <label for="email" class="col-sm-2 col-form-label">名称:</label>  
 <div class="col-sm-8">  
 <span th:text="${one.name}"></span>  
 </div>  
 </div>  
 <div class="form-group row">  
 <label for="email" class="col-sm-2 col-form-label">浏览量:</label>  
 <div class="col-sm-8">  
 <span th:text="${one.viewCount}"></span>  
 </div>  
 </div>  
 <div class="form-group row">  
 <label for="email" class="col-sm-2 col-form-label">创建时间:</label>  
 <div class="col-sm-8">  
 <span th:text="${#dates.format(one.createTime,'yyyy-MM-dd HH:mm:ss')}"></span>  
 </div>  
 </div>  
 <div class="form-group row">  
 <label for="email" class="col-sm-2 col-form-label">更新时间:</label>  
 <div class="col-sm-8">  
 <span th:text="${#dates.format(one.updateTime,'yyyy-MM-dd HH:mm:ss')}"></span>  
 </div>  
 </div>  
 <div class="form-group row">  
 <label for="email" class="col-sm-2 col-form-label">内容:</label>  
 <div class="col-sm-8">  
 [(${one?.content})]  
 </div>  
 </div>  
 </div>  
 </div>  
  
 <div class="row show" id="ansSelList">  
 <div class="col-md-11 col-sm-offset-1">  
 <h4 class="sub-header">评论列表</h4>  
 <div class="table-responsive mt-2">  
 <table class="table table-striped table-hover">  
 <thead>  
 <tr>  
 <th>标题</th>  
 <th width="400">内容</th>  
 <th>作者</th>  
 <th>创建时间</th>  
 </tr>  
 </thead>  
 <tbody>  
 <th:block th:each="oneTemp : ${commentList }">  
 <tr>  
 <td th:text="${oneTemp.name }"></td>  
 <td>[(${oneTemp?.content})]</td>  
 <td th:text="${oneTemp.users?.nickName }"></td>  
 <td th:text="${#dates.format(oneTemp.createTime,'yyyy-MM-dd HH:mm:ss')}"></td>  
 </tr>  
 </th:block>  
 </tbody>  
 </table>  
 </div>  
 </div>  
 </div>  
</div>  
<meta th:replace="~{/head/head\_include :: footer}"/>  
</body>  
</html>

#### ③workInsert.html

<!DOCTYPE html>  
<html lang="zh-CN" xmlns:th="http://www.thymeleaf.org">  
<head>  
 <title>作品列表 - 今日头条(用户)</title>  
 <meta th:replace="~{/head/head\_include :: title}"/>  
 <link rel="stylesheet" th:href="@{/head\_res/bootstrapvalidator/css/form-validation.css}">  
 <script type="text/javascript" th:src="@{/head\_res/bootstrapvalidator/js/form-validation.js}"></script>  
 <script type="text/javascript" charset="utf-8" th:src="@{/head\_res/ueditor/ueditor.config.js}"></script>  
 <script type="text/javascript" charset="utf-8" th:src="@{/head\_res/ueditor/ueditor.all.min.js}"> </script>  
 <script type="text/javascript" charset="utf-8" th:src="@{/head\_res/ueditor/lang/zh-cn/zh-cn.js}"></script>  
 <script type="text/javascript" th:inline="javascript">  
 $(function()  
 {  
 $("#form").FormValidation({  
 fields:  
 {  
 /\* 邮箱 \*/  
 name:  
 {  
 validators: {  
 notEmpty: {  
 message: '名称不能为空'  
 },  
 stringLength: {  
 min: 4,  
 max:40,  
 message: '名称的长度为在4位与40位之间'  
 }  
 }  
 }  
 }  
 });  
  
 var ue = UE.getEditor('editor');  
 var content = [[${one?.content}]];  
 /\* 百度编辑器提供的方法 \*/  
 ue.addListener("ready", function () {  
 /\* 为百度编辑器赋值 \*/  
 ue.setContent(content);  
 });  
 });  
 </script>  
</head>  
<body>  
<meta th:replace="~{/head/head\_include :: header}"/>  
<!-- 网页的主体 -->  
<div class="container">  
 <div class="col-md-8 col-sm-10">  
 <form id="form" class="form-horizontal" method="post"  
 th:action="@{/head/work/workInsertSubmit}">  
  
 <h2 class="col-sm-10 col-sm-offset-1 form-signin-heading text-center">添加作品</h2>  
 <div class="form-group">  
 <label for="email" class="col-sm-3 control-label">名称</label>  
 <div class="col-sm-8">  
 <input type="text" id="name" name="entity.name" class="form-control"  
 th:value="${one.name}" placeholder="名称">  
 </div>  
 </div>  
 <div class="form-group">  
 <label for="content" class="col-sm-2 control-label">内容:</label>  
 <div class="col-sm-8">  
 <script id="editor" name="entity.content" type="text/plain" style="width:800px;height:300px;" th:text="${one.content}"></script>  
 </div>  
 </div>  
 <div class="form-group">  
 <div class="col-sm-8 col-sm-offset-2">  
 <button class="btn btn-md btn-primary" type="submit">提交</button>  
 </div>  
 </div>  
 </form>  
 </div>  
</div>  
<meta th:replace="~{/head/head\_include :: footer}"/>  
</body>  
</html>

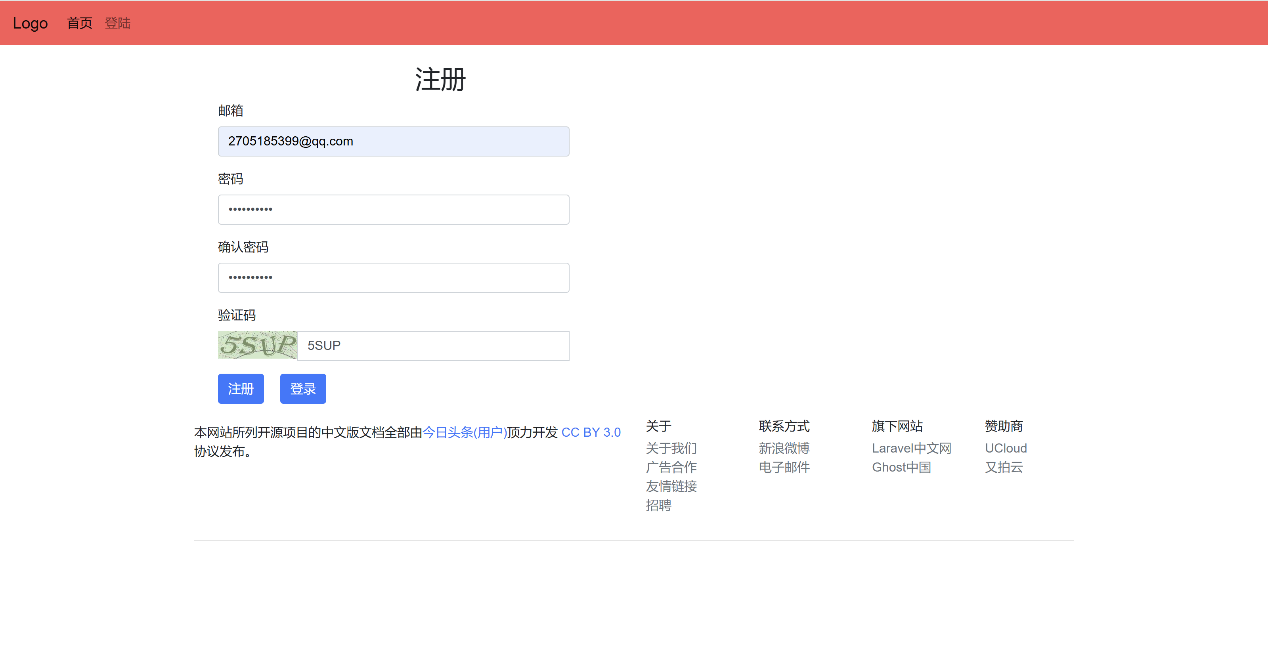
#### ①workUpdate.html

<!DOCTYPE html>  
<html lang="zh-CN" xmlns:th="http://www.thymeleaf.org">  
<head>  
 <title>工作列表 - 今日头条(用户)</title>  
 <meta th:replace="~{/head/head\_include :: title}"/>  
 <link rel="stylesheet" th:href="@{/head\_res/bootstrapvalidator/css/form-validation.css}">  
 <script type="text/javascript" th:src="@{/head\_res/bootstrapvalidator/js/form-validation.js}"></script>  
 <script type="text/javascript" charset="utf-8" th:src="@{/head\_res/ueditor/ueditor.config.js}"></script>  
 <script type="text/javascript" charset="utf-8" th:src="@{/head\_res/ueditor/ueditor.all.min.js}"> </script>  
 <script type="text/javascript" charset="utf-8" th:src="@{/head\_res/ueditor/lang/zh-cn/zh-cn.js}"></script>  
 <script type="text/javascript" th:inline="javascript">  
  
 // 表单验证和编辑器初始化  
 $(function()  
 {  
 $("#form").FormValidation({  
 // 这里使用了 FormValidation 插件对表单进行验证  
 fields:  
 {  
 /\* 邮箱 \*/  
 name:  
 {  
 validators: {  
 notEmpty: {  
 message: '名称不能为空'  
 },  
 stringLength: {  
 min: 4,  
 max:40,  
 message: '名称的长度为在4位与40位之间'  
 }  
 }  
 }  
 }  
 });  
  
 var ue = UE.getEditor('editor');  
 // 初始化百度的富文本编辑器  
 var content = [[${one?.content}]];  
 // 百度编辑器提供的方法  
 ue.addListener("ready", function () {  
 // 为百度编辑器赋值  
 ue.setContent(content);  
 });  
 });  
 </script>  
</head>  
<body>  
<meta th:replace="~{/head/head\_include :: header}"/>  
  
<!-- 网页的主体 -->  
<div class="container">  
 <div class="col-md-8 col-sm-10">  
 <form id="form" class="form-horizontal" method="post"  
 th:action="@{/head/work/workUpdateSubmit}">  
 <input type="hidden" name="entity.id" th:value="${one.id}">  
<!-- one 变量是从后台传递过来的数据对象，其中包含了当前作品的 ID 和其他相关信息-->  
 <input type="hidden" name="operType" th:value="${paramQuery.operType}">  
<!-- paramQuery.operType 是 URL 或请求参数中传递的操作类型（例如，“编辑”或“更新”）-->  
 <h2 class="col-sm-10 col-sm-offset-1 form-signin-heading text-center">更新作品</h2>  
 <div class="form-group">  
 <label for="email" class="col-sm-3 control-label">名称</label>  
 <div class="col-sm-8">  
 <input type="text" id="name" name="entity.name" class="form-control"  
 th:value="${one.name}" placeholder="名称">  
 </div>  
 </div>  
 <div class="form-group">  
 <label for="content" class="col-sm-2 control-label">内容:</label>  
 <div class="col-sm-8">  
 <script id="editor" name="entity.content" type="text/plain" style="width:800px;height:300px;" th:text="${one.content}"></script>  
 </div>  
 </div>  
<!-- 这个部分使用了百度的富文本编辑器，one.content会将当前作品的内容传递给编辑器，让用户能够查看并编辑它-->  
 <div class="form-group">  
 <div class="col-sm-8 col-sm-offset-2">  
 <button class="btn btn-md btn-primary" type="submit">提交</button>  
 </div>  
 </div>  
 </form>  
 </div>  
</div>  
<meta th:replace="~{/head/head\_include :: footer}"/>  
</body>  
</html>

# 九、相关功能展示

## 1.注册









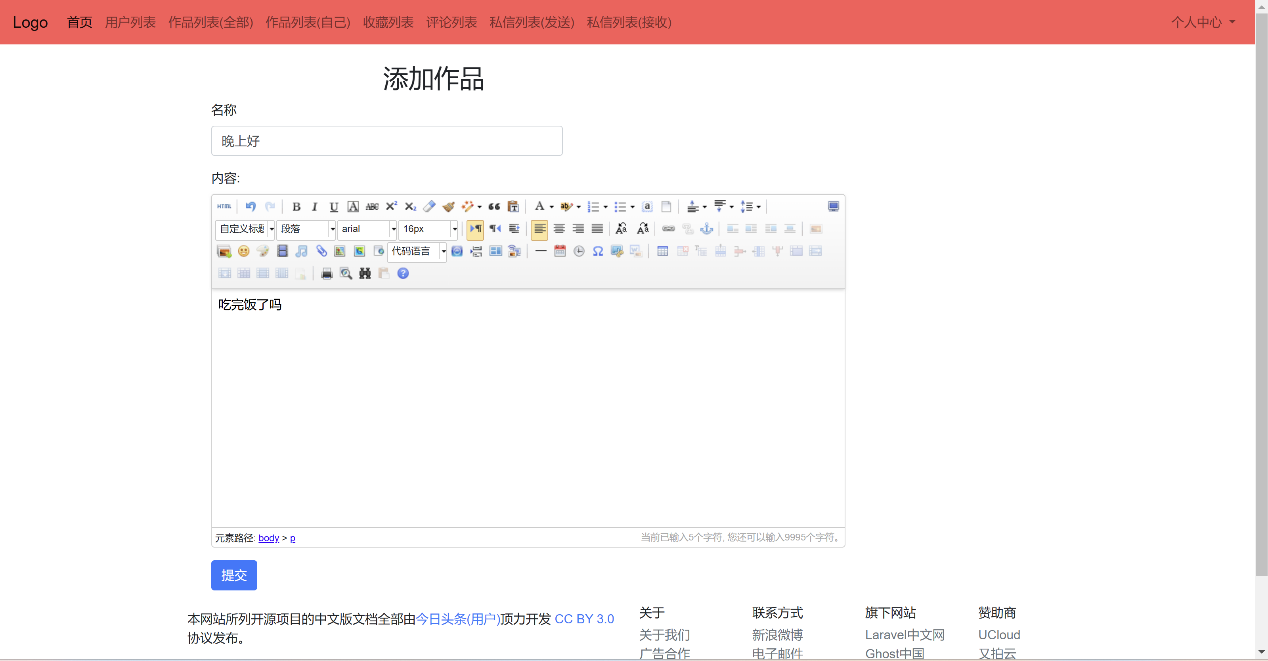


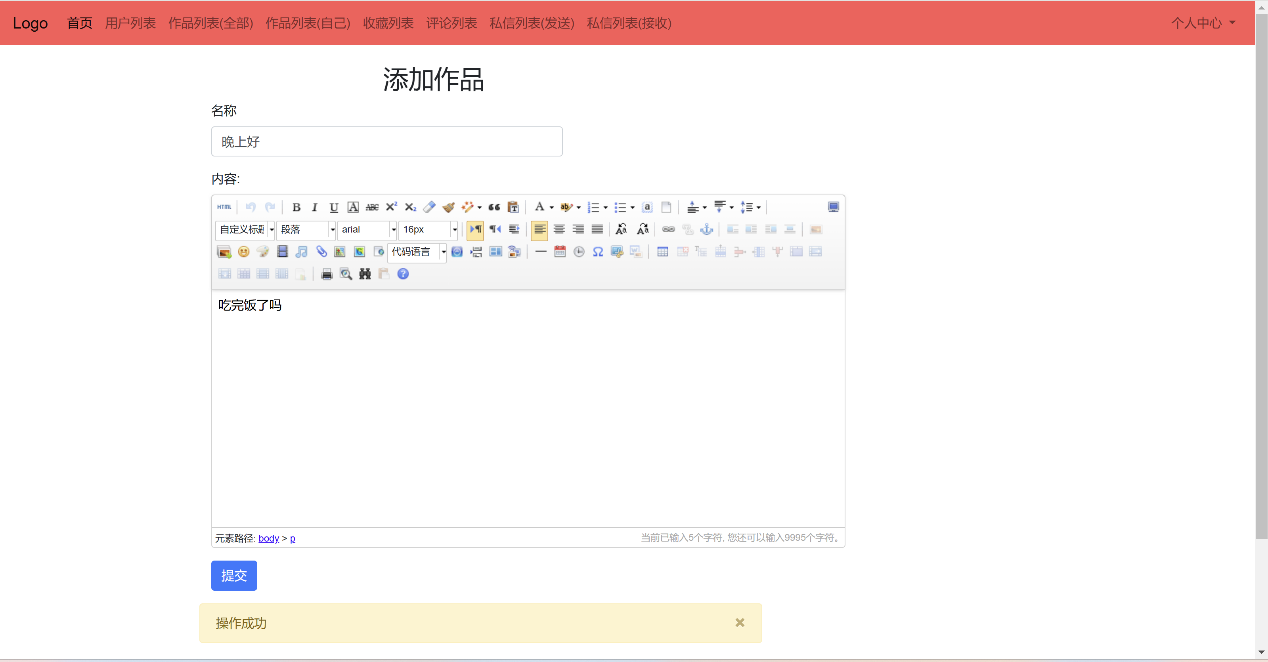
注册一个新的账号，并且注册后登录。

## 2.作品（还是以作品为例，剩下的已经验收过了）



### （1）发表作品

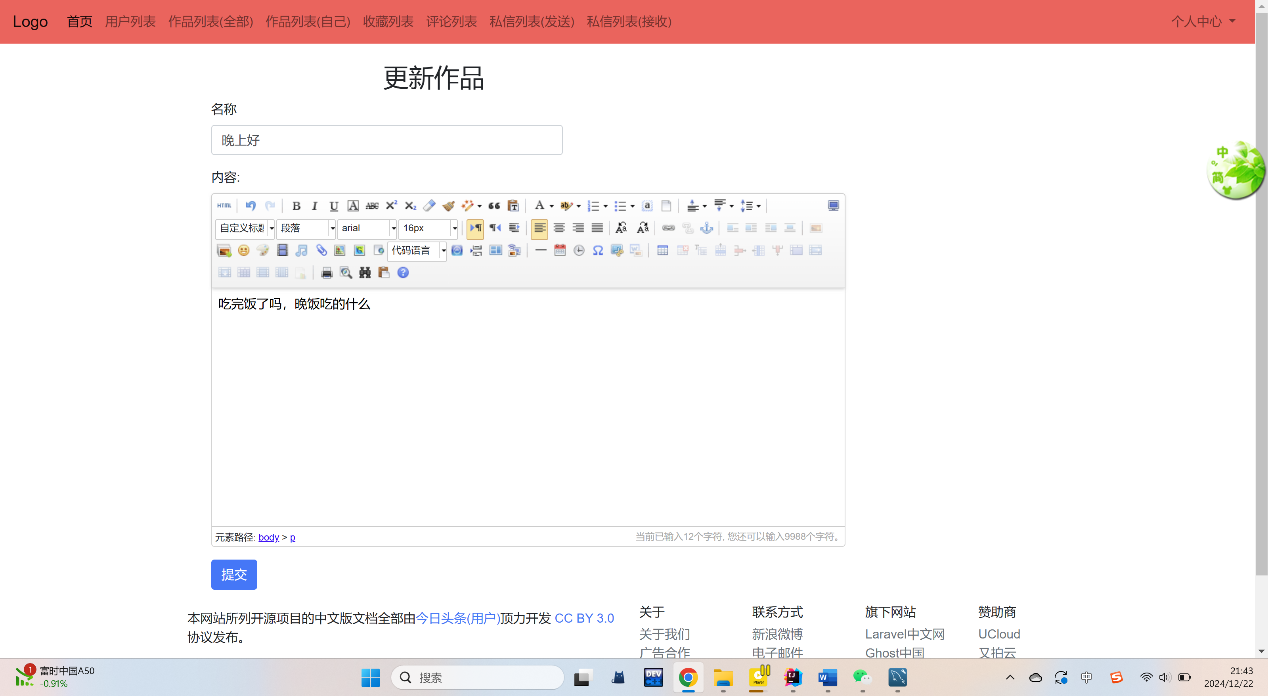


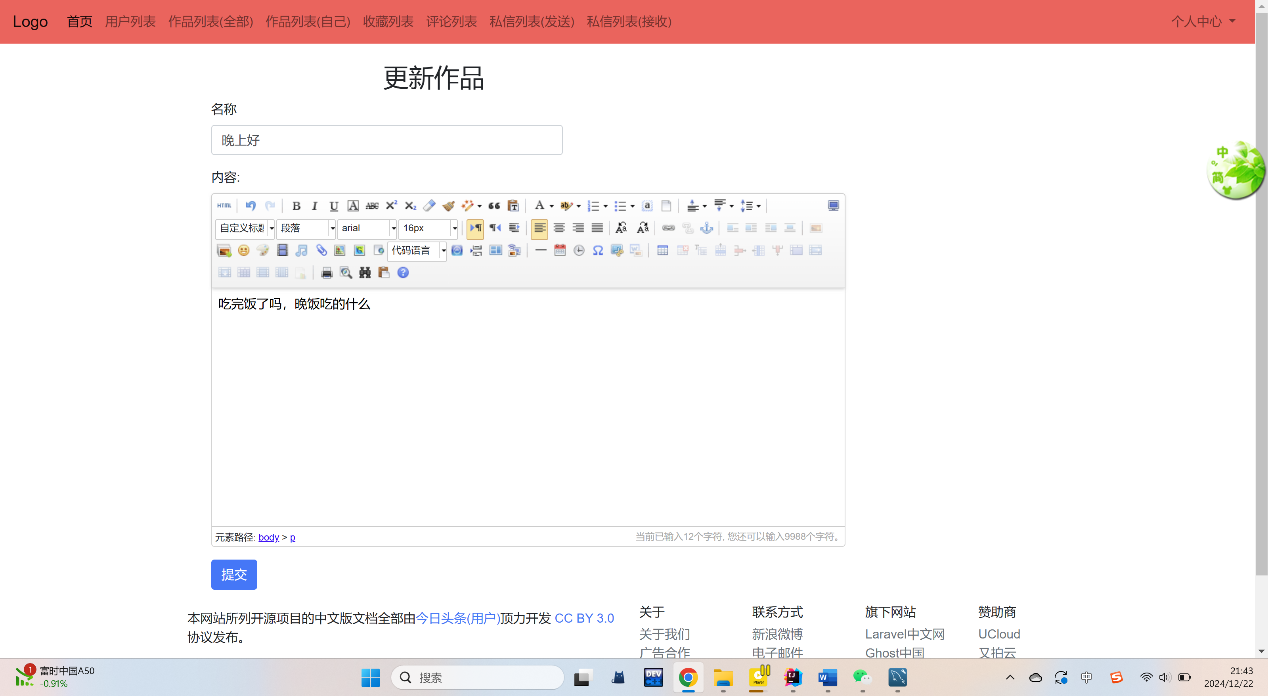


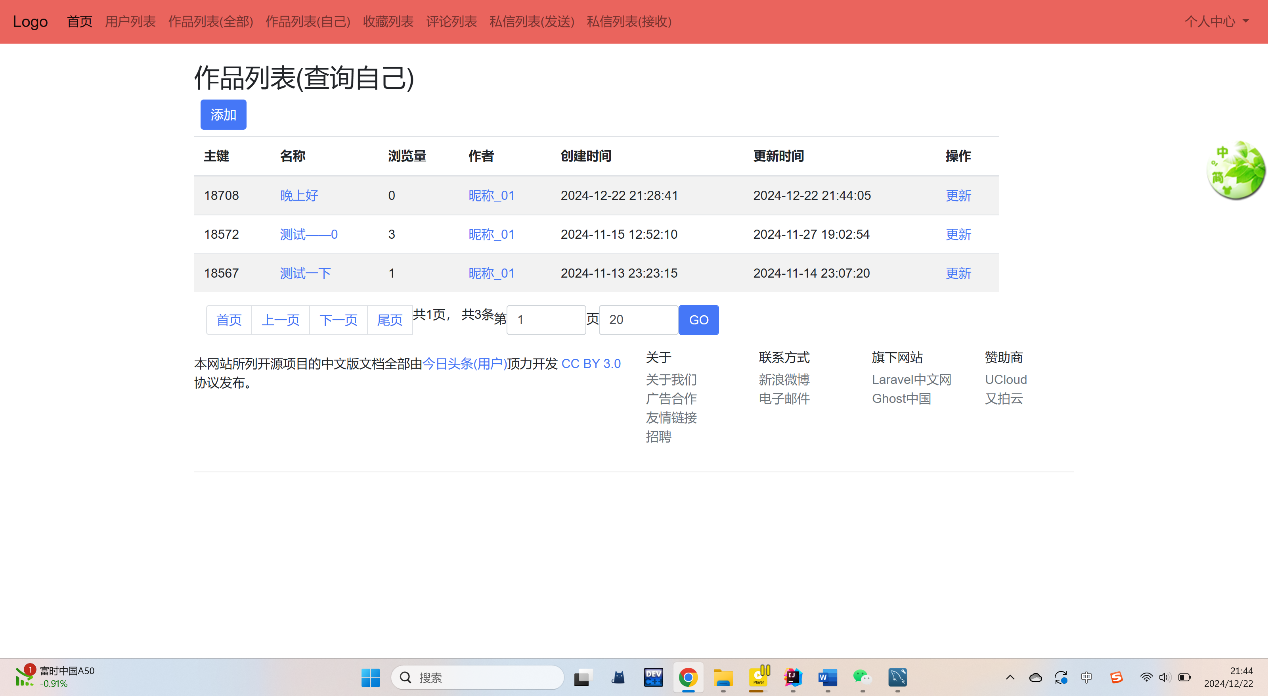


创建了一个新的作品，并且记录了创建的时间。

### （2）修改作品







作品内容发生了修改，并且记录了修改时间。

# 十、结论

通过本次数据库原理课程设计，我成功模拟实现了今日头条用户部分的核心功能，包括用户注册、作品发布与管理、收藏、评论以及私信交流等。在设计过程中，我深入理解了数据库的需求分析、概念设计、逻辑设计以及物理设计等关键环节，掌握了使用MySQL数据库进行数据存储与管理的方法。同时，通过Spring Boot框架的应用，实现了后端业务逻辑的高效处理，以及与前端页面的无缝对接。

本设计不仅加深了我对数据库原理的理解，还提升了我在实际项目中应用数据库技术的能力。在功能实现方面，我们确保了用户操作的便捷性、数据存储的安全性以及信息交互的实时性。通过分页展示、数据筛选等功能的实现，优化了用户体验，提高了信息检索的效率。此外，我还对系统进行了初步的性能测试，确保在高并发情况下系统的稳定运行。

综上所述，本次课程设计达到了预期目标，为我今后在软件开发领域，尤其是数据库相关项目的开发积累了宝贵的经验。随着技术的不断进步，数据库系统将面临更多的挑战和机遇，我也将持续学习，不断提升自己的技术水平，以适应未来的发展需求。