****

**应用基础实践（一）网络与JAVA**

**课程设计实验报告**

|  |  |
| --- | --- |
| 学生姓名 | 邱泓铭 |
| 学 号 | 8208241214 |
| 专业班级 | 计科2407 |
| 指导教师 | 刘嫔 |
| 学 院 | 计算机学院 |
| 完成时间 | 2025/12/12 |

计算机学院

2025年12月

* + 1. **系统描述：**

本次实训搭建的云图书借阅系统是中南大学大学计科 java 一班班的实训项目，旨在为图书馆提供高效的图书管理与借阅服务。该系统涵盖图书管理、用户管理、借阅管理、评价管理等多个功能模块，通过线上线平台实现图书资源的数字化管理与便捷访问。

**基本要求：**

1. **功能性**：系统应具备完整的图书借阅管理功能，包括但不限于图书借阅、图书归还、图书管理、图书统计等。
2. **安全性**：确保系统的安全性和数据隐私保护，防止未经授权的访问和数据泄露。实现各个功能的封装，提高系统的安全性和鲁棒性。
3. **可扩展性**：系统设计应具备良好的扩展性，能够方便地添加新功能或集成第三方服务。
4. **性能**：系统应具备高并发处理能力，能够在大量用户同时使用时保持稳定运行。

**基本内容：**

本人在此次实验中主要负责**用户认证与管理模块**的开发，主要负责实现基于 JWT 的用户登录鉴权、新用户注册入库（含密码加密处理）以及当前登录用户信息获取功能。需要完成从 Controller 层到 Service 层再到 DAO 层的完整设计，基于 MyBatis-Plus 的 BaseMapper 实现数据交互；同时负责封装通用的 Token 解析与线程上下文（ThreadLocal）工具，与团队其他成员对接，为整个系统的其他业务模块提供统一的身份校验基础，确保接口访问的安全性和无状态性。

**二．功能模块结构**

**1. 用户认证模块（登录功能）**

该模块实现用户的安全登录认证，并颁发 JWT 令牌。

**1）用户登录：：**

**结构：**

**1.表现层：** 包含 UserController（处理 /user/login 请求），接收 UserLoginDTO 参数，返回统一的 ApiResponse<LoginVO>。

**2.业务层：** 由 IUserService 接口和 UserServiceImpl 实现类组成，负责执行核心的身份验证逻辑（账号查询、密码比对）。

**3.数据访问层：** UserMapper 继承 MyBatis-Plus 的 BaseMapper，提供对 user 表的 CRUD 能力。

**4.实体与视图对象：** User（映射数据库表）、UserLoginDTO（封装登录请求参数）、LoginVO（封装登录成功后的用户信息及 Token）。

**5.工具组件：** JwtUtils（用于生成 JWT 令牌）、DigestUtils（用于 MD5 密码加密）。

**流程：**

 1.前端页面发起登录请求，传入用户名和密码，封装为 JSON 格式。

 2.UserController 接收请求，将参数映射为 UserLoginDTO，调用业务层接口 userService.login()。

 3.业务层（UserServiceImpl）使用 LambdaQueryWrapper 根据用户名查询数据库。

 4.若用户不存在，抛出“用户名或密码错误”异常。

 5.若用户存在，获取前端输入的密码，使用 DigestUtils.md5DigestAsHex 进行 MD5 加密。

 6.将加密后的密码与数据库中存储的密文密码进行比对。若不匹配，抛出异常。

 7.验证通过后，Controller 层构建 Claims（包含 userId 和 role），调用 JwtUtils.generateJwt 生成 Token 字符串。

8.将用户信息和 Token 封装进 LoginVO，通过 ApiResponse 返回给前端。

**2. 用户管理模块（注册与查询）**

**（1）用户注册：**

实现新用户注册入库及用户基础信息的获取。

**流程：**

1. 前端提交注册信息（用户名、密码、邮箱）。
2. UserController 接收 UserRegisterDTO。
3. 业务层首先查询数据库，检查用户名是否已存在（selectCount），若存在则抛出异常。
4. 对用户输入的明文密码进行 **MD5 加密**处理，确保数据库不存储明文。
5. 设置默认角色（如 "user"），调用 save() 方法将用户信息写入数据库。
6. 注册成功后返回包含用户 ID 和注册时间的 RegisterVO。

（2）用户信息获取：

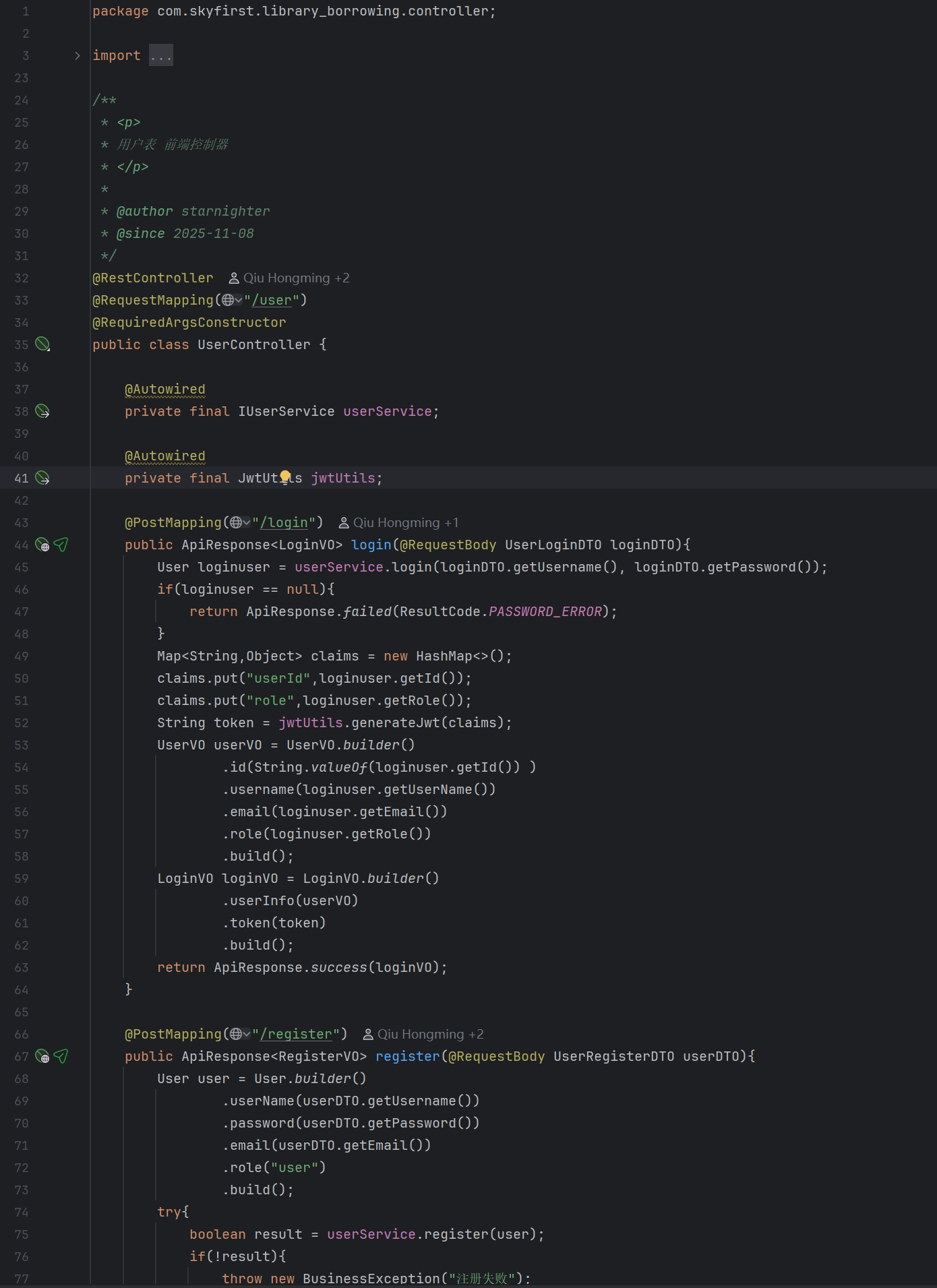
**流程：**

1. 前端请求 /user/info 接口，在 Header 中携带 Token。
2. 拦截器（预处理）解析 Token，将解析出的 userId 存入 BaseContext 线程局部变量中。
3. UserController 通过 BaseContext.getCurrentId() 获取当前登录用户的 ID。
4. 调用 userService.getUserInfo 根据 ID 查询用户详情。
5. 将 User 实体转换为 UserVO（脱敏，去除密码字段）返回给前端。

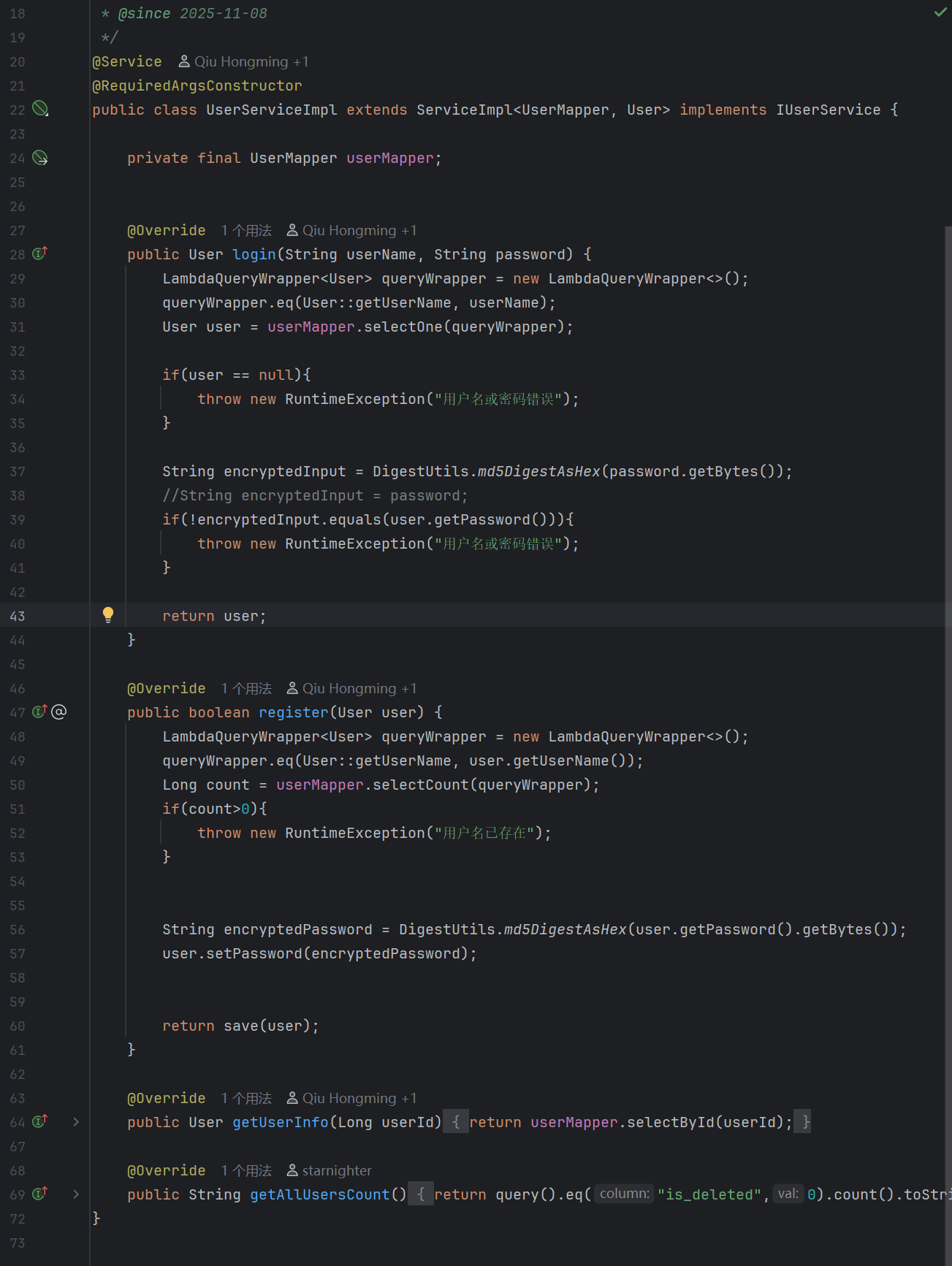
**三、主要模块的算法说明**

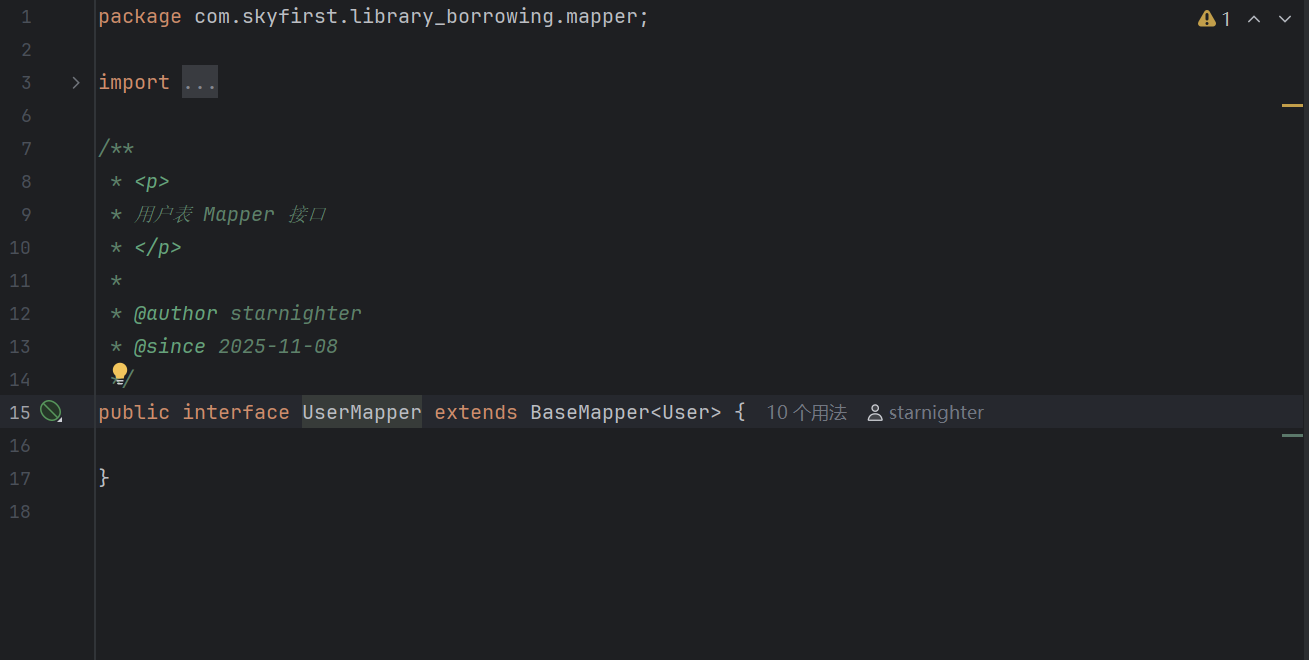
**1. 分层架构设计**

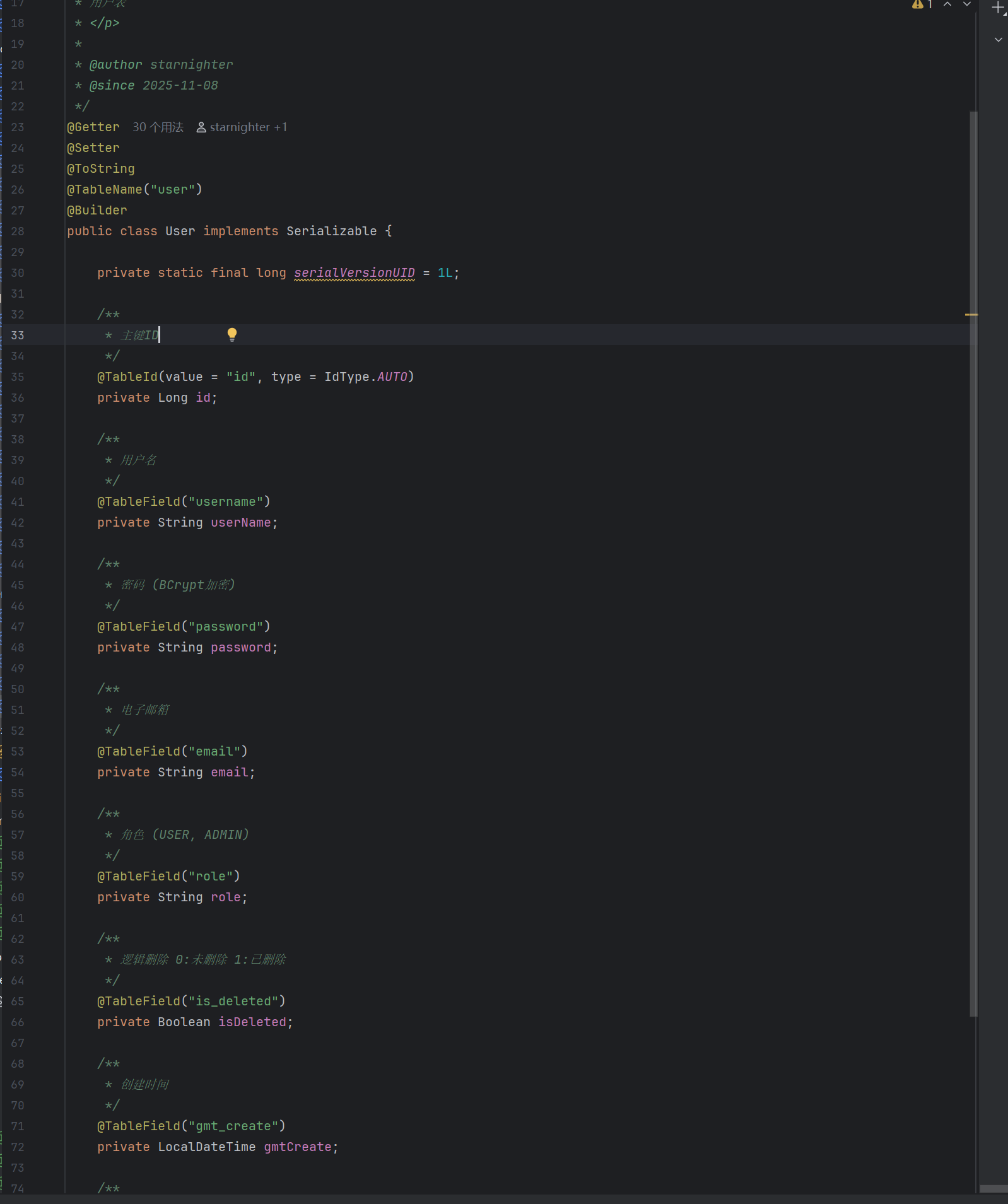
* **Controller 层：**系统认证入口。例如 UserController 使用 @RestController 标识，通过 @PostMapping("/login") 暴露登录接口。利用 ApiResponse 对结果进行统一封装，确保前后端交互格式一致。



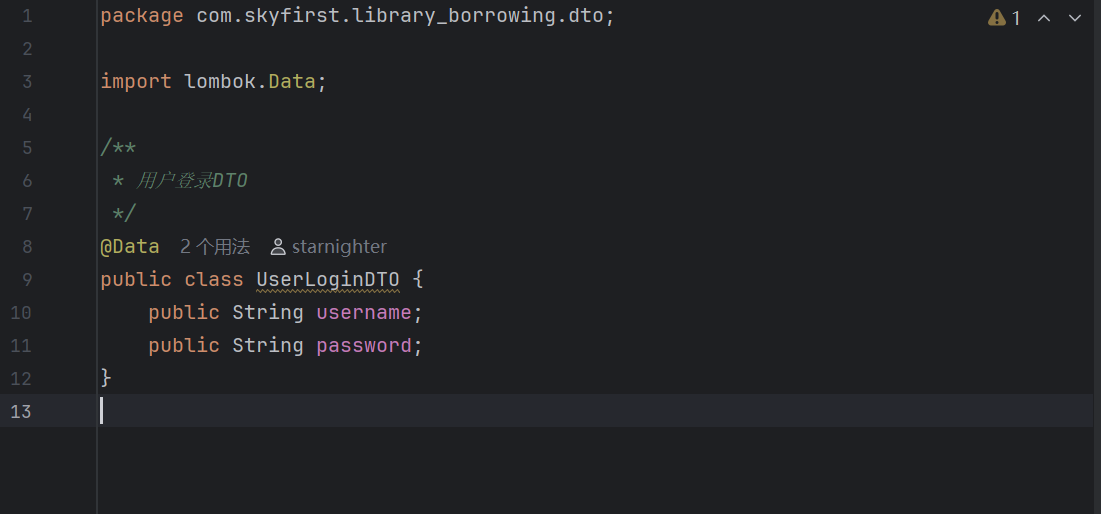
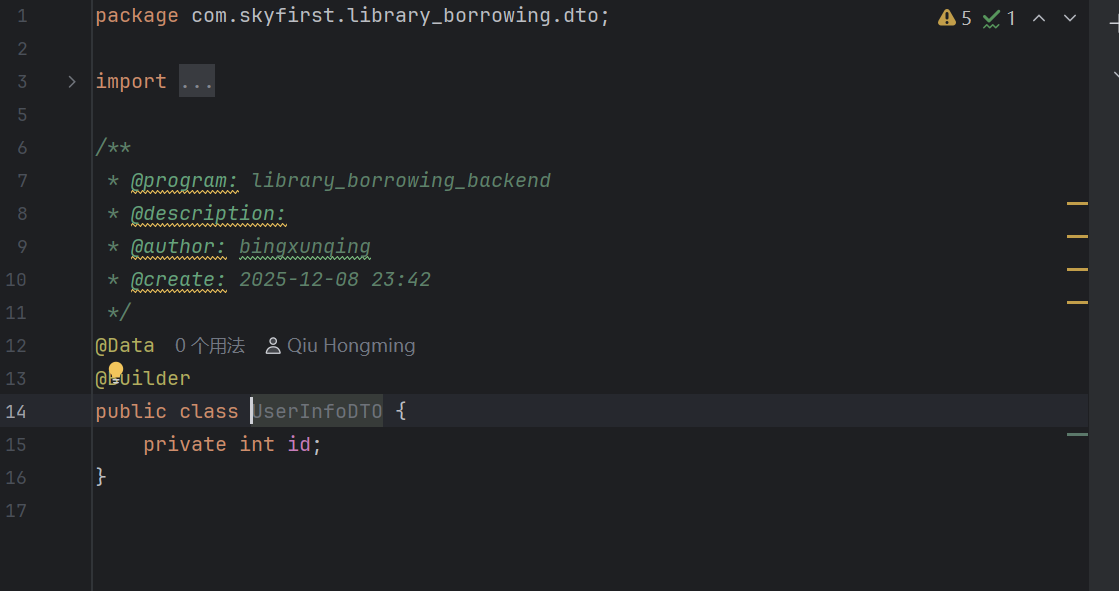
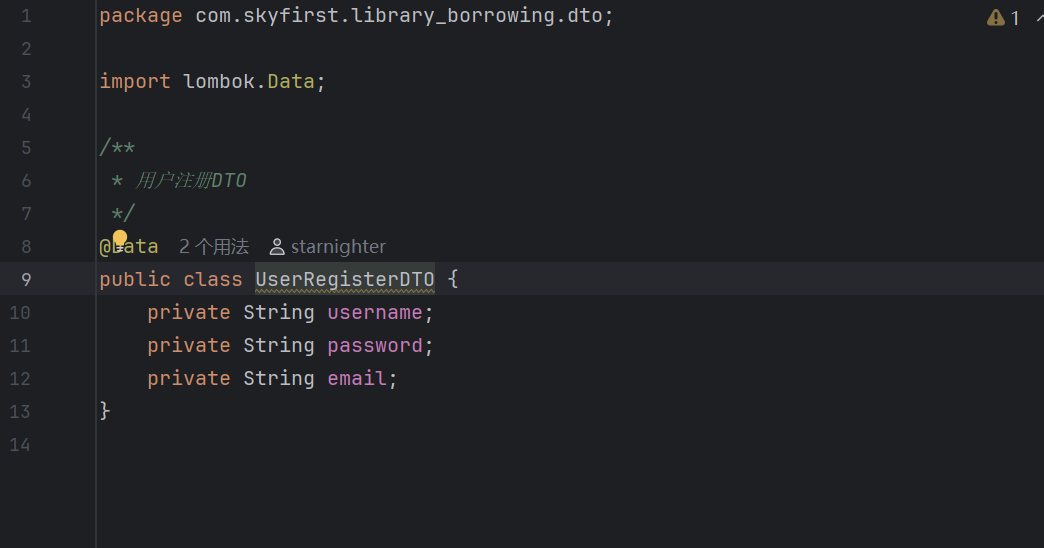
* **Service 层：**核心业务逻辑载体。UserServiceImpl 继承 ServiceImpl，实现了复杂的登录校验逻辑（如先查号再验密）和事务控制。



* **Mapper 层:** 基于 MyBatis-Plus 自动代理，UserMapper 仅需继承接口即可获得标准的 SQL。 
* **Entity 层:**映射数据库表结构，是数据在程序中的载体。



**DTO/VO层：**   
DTO ： 如 UserLoginDTO，仅包含 username 和 password，精确控制前端传递的参数。

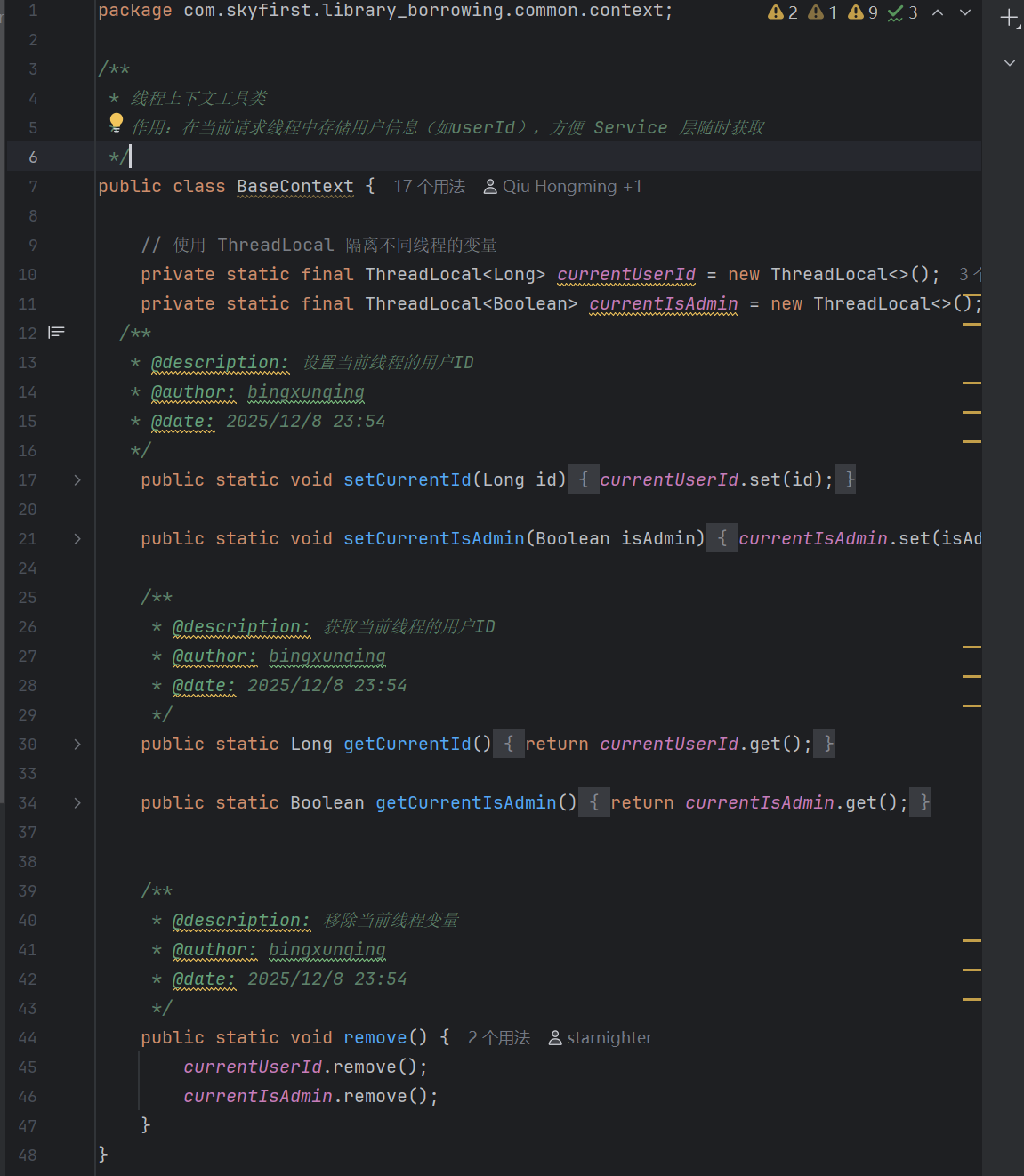
 VO： 如 UserVO，用于响应数据，特意将 ID 转为 String 类型防止前端精度丢失，并去掉了password这种不应该传回前端的敏感的字段，进一步切合前端需要的参数。  

1. **具体功能代码展示**

(1)BaseContext

在 Spring Boot 的并发模型中，每个 HTTP 请求由独立的线程处理。为了在 Controller、Service 或 Dao 层之间方便地共享“当前登录用户”的信息，而无需在每个方法参数中层层传递 userId，本系统设计了 BaseContext 工具类。

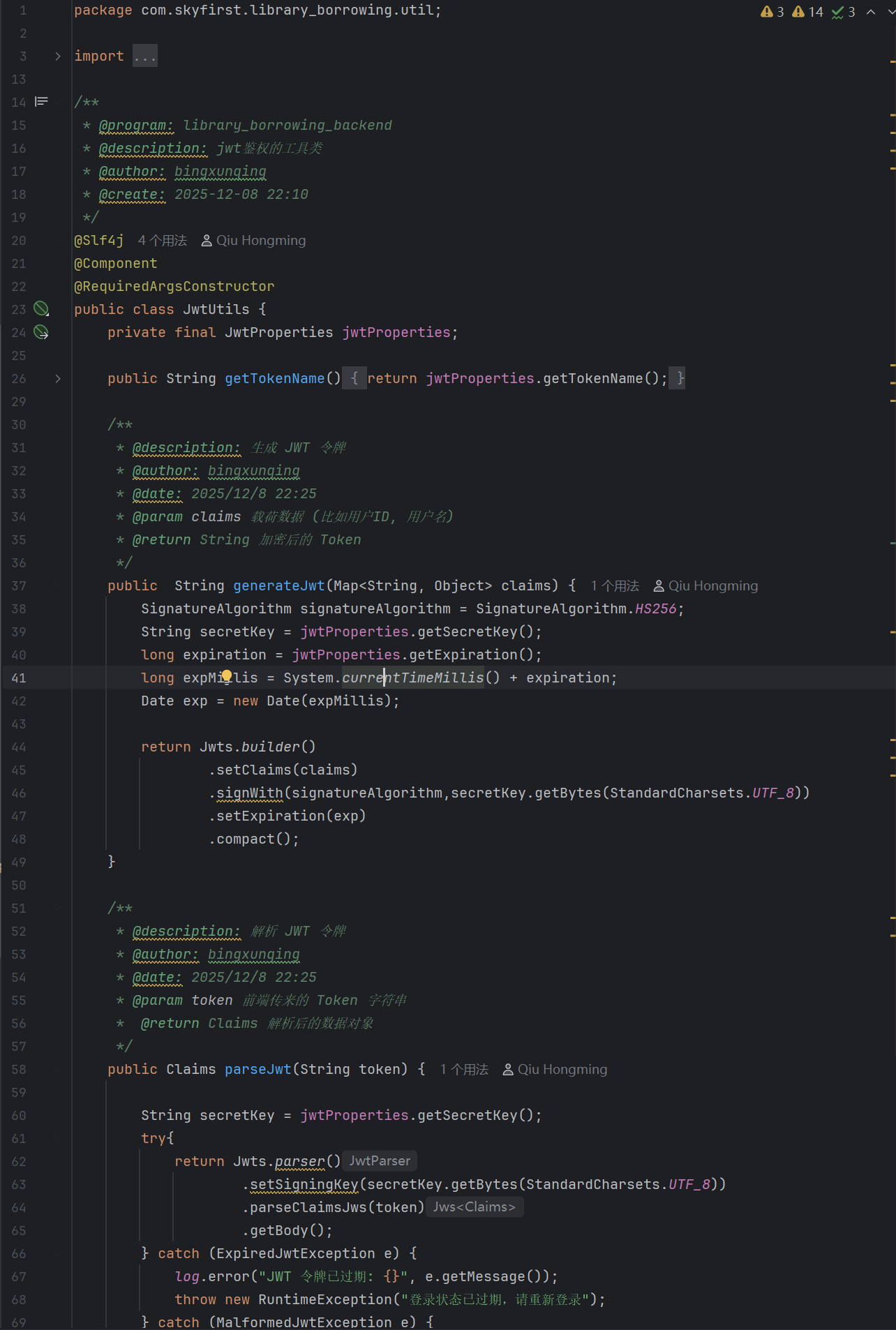
* **算法原理：** 利用 JDK 原生的 **ThreadLocal** 机制。ThreadLocal 为每个线程提供独立的变量副本，实现了**线程级别的数据隔离**。
  + **存储 (setCurrentId)：** 在请求拦截器（Interceptor）解析 Token 成功后，将 userId 和 isAdmin 存入当前线程的 ThreadLocal 容器。
  + **获取 (getCurrentId)：** 业务层（Service）在任意位置调用此方法，即可获取当前请求对应的用户信息，且不会与其他并发请求发生冲突。
  + **清理 (remove)：** 在请求结束（响应返回后），强制清空 ThreadLocal，防止内存泄漏和线程复用导致的数据污染。



**（2）Jwt令牌生成和解析工具类**

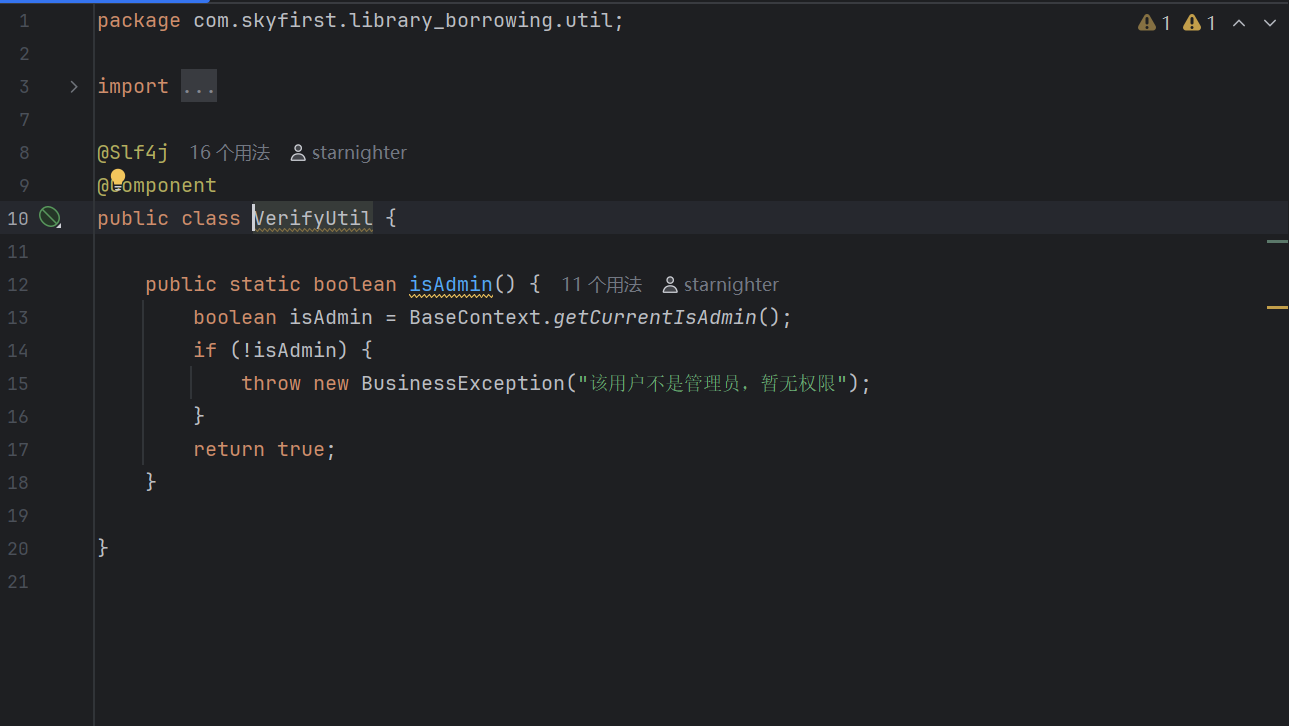
为了实现前后端分离架构下的无状态认证，系统引入了 JWT (JSON Web Token) 技术。

* 配置隔离 (JwtProperties)： 采用 Spring Boot 的 @ConfigurationProperties 注解，将配置文件（application.yml）中前缀为 skyfirst.jwt 的敏感参数（如密钥 secretKey、过期时间 expiration）映射为 Java Bean。这种做法实现了配置与代码的解耦，便于后续运维修改参数而无需重新编译代码。
* 令牌生成算法 (generateJwt)： 使用 HMAC-SHA256 (HS256) 对称加密算法。
  1. 载荷构建： 接收包含用户 ID 和角色的 Map 集合作为 Claims（载荷）。
  2. 签名过程： 使用配置中的 secretKey 对头部和载荷进行签名，防止令牌在传输过程中被篡改。
  3. 时效控制： 通过 System.currentTimeMillis() + expiration 计算绝对过期时间，确保令牌具有生命周期，降低被盗用后的风险。
* 令牌解析与异常处理 (parseJwt)： 在解析过程中，通过 Jwts.parser() 反向验证签名。算法内部封装了详细的异常处理机制：
  1. 捕获 ExpiredJwtException：处理令牌过期情况。
  2. 捕获 MalformedJwtException：处理令牌格式被篡改的情况。
  3. 最终返回解析后的 Claims 对象，供拦截器提取用户信息。

**（3）统一权限校验工具 (VerifyUtil)**

为了简化代码并增强安全性，系统封装了静态工具类 VerifyUtil 用于鉴权。

* 实现逻辑： 该工具类依赖于 BaseContext。当调用 VerifyUtil.isAdmin() 时，系统直接从当前线程上下文中获取用户的角色状态。如果用户非管理员，直接抛出自定义的 BusinessException。
* 作用： 将权限校验逻辑从业务代码中剥离，实现了关注点分离。只需一行代码即可完成权限拦截，显著降低了代码冗余度。

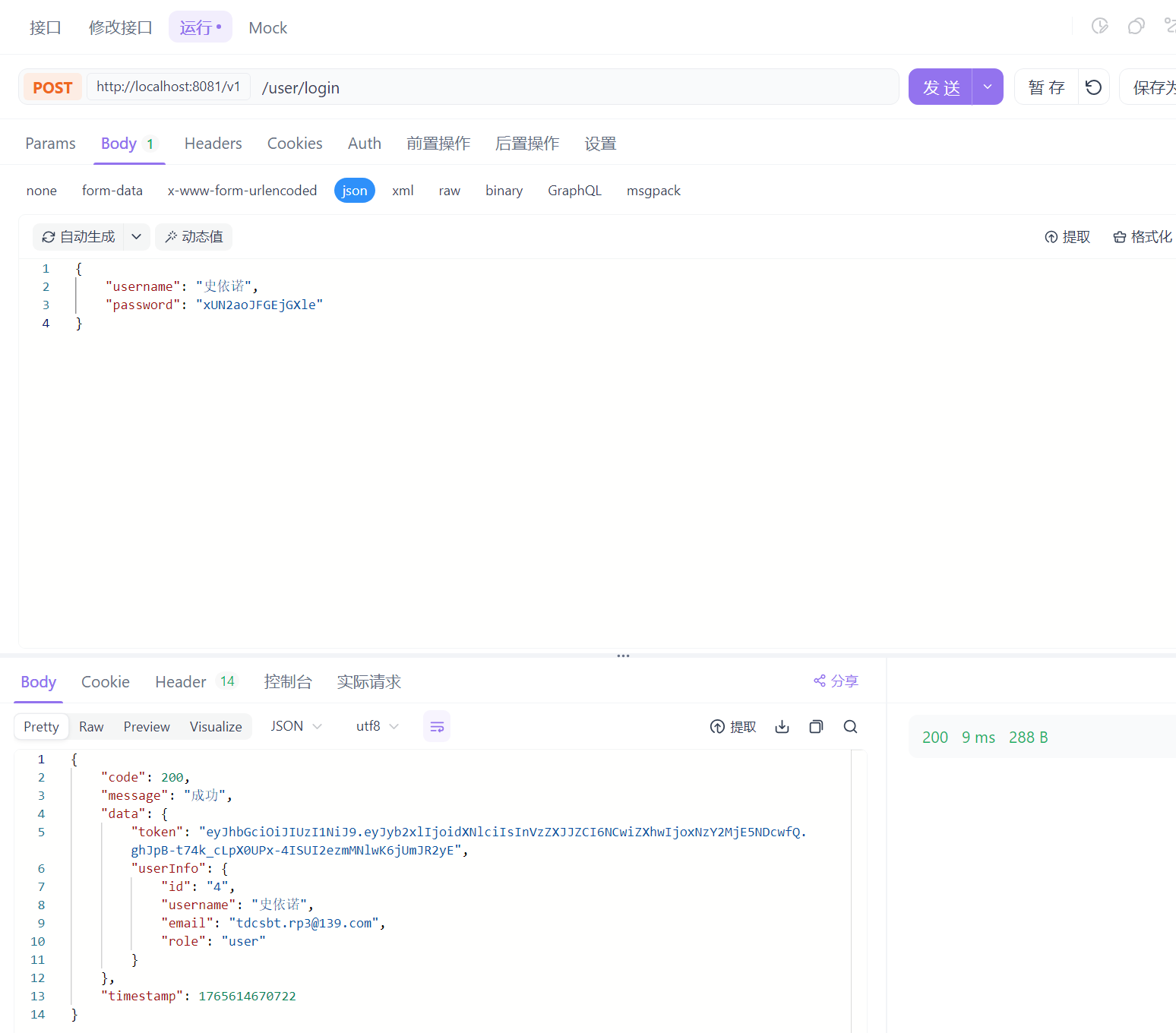


**四**．**运行结果：**

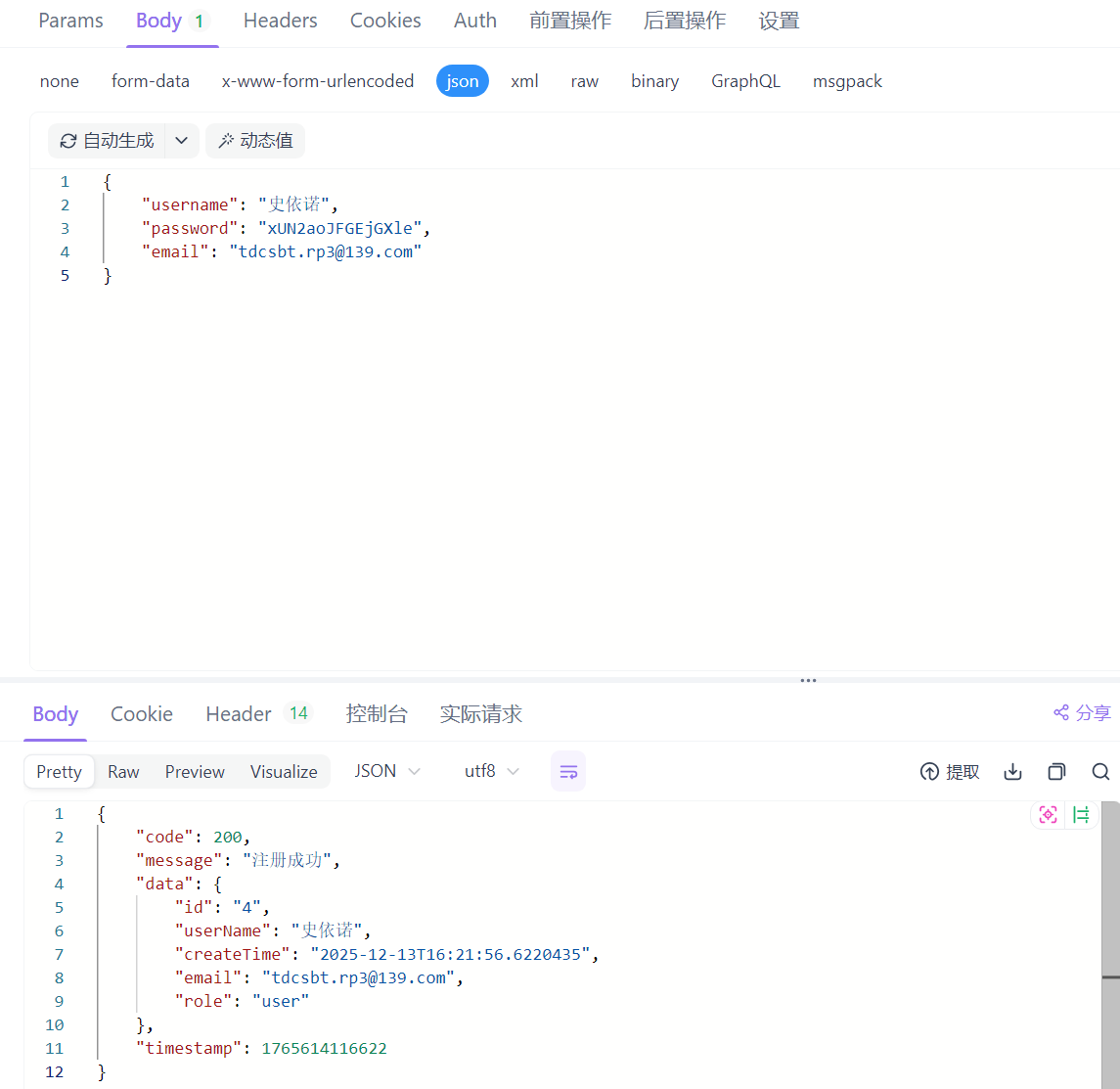
登录页面功能：

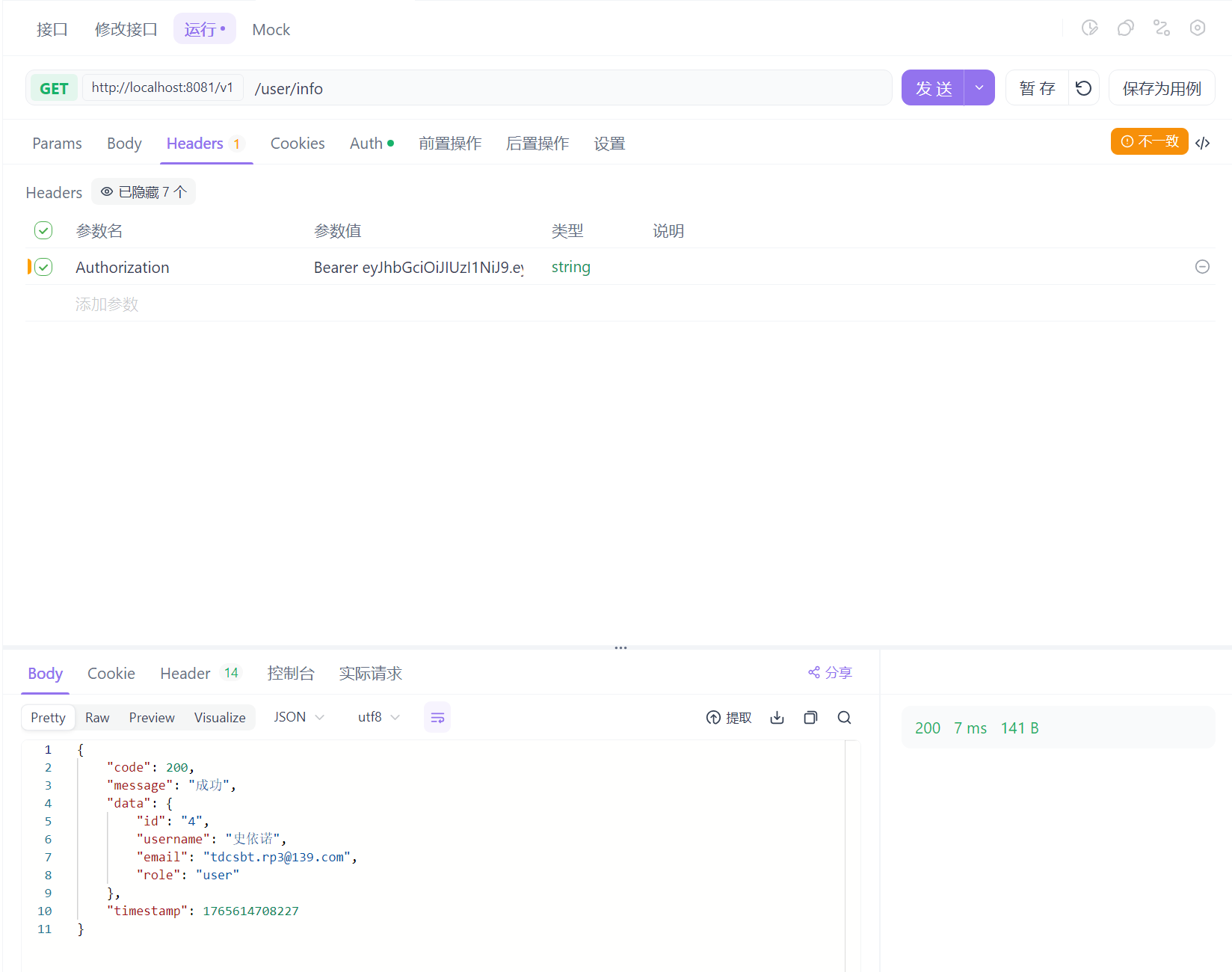






注册功能：

个人信息：



1. **课程设计总结：**

**遇到的问题及解决方案**

1. 前后端数据类型不匹配问题（精度丢失）： 在开发初期，发现数据库中的 id 为 Long 类型（19位），直接返回给前端 Apifox 测试时，JavaScript 会因为精度不足导致 ID 后几位变为 0。 解决方案： 修改 VO 层逻辑，在 UserVO 和 LoginVO 中将 Long 类型的 id 统一通过 String.valueOf() 转换为字符串格式返回，彻底解决了前端 ID 精度丢失的问题。
2. 用户信息在多层间的传递问题： 最初在获取用户个人信息时，试图在 Controller 和 Service 的每个方法参数中都传递 userId，导致代码极其冗余且难以维护。 解决方案： 学习并引入了 ThreadLocal 技术（封装为 BaseContext 工具类），在拦截器解析 Token 后将用户信息存入当前线程，实现了用户信息在 Service 层的随取随用，极大地净化了代码结构。
3. 密码安全性问题： 起初为了方便测试，数据库直接存储明文密码，意识到这存在极大的安全隐患。 解决方案： 引入 DigestUtils 工具类，在注册时对密码进行 MD5 加密存入数据库；在登录时，将输入的明文同样进行加密后再与数据库密文比对，确保了用户敏感数据的安全性。
4. JWT 令牌过期与异常处理： 在使用 JWT 时，如果令牌过期或被篡改，系统会直接抛出 500 错误，前端无法获得有效提示。 解决方案： 在 JwtUtils 中增加了对 ExpiredJwtException 等异常的捕获逻辑，并结合全局异常处理器，抛出自定义的 BusinessException，向前端返回清晰的“登录已过期”提示状态码。

**收获与体会**

通过本次云图书借阅系统登录模块的开发，我从单纯的“写代码”进阶到了“设计系统”的层面。我深入理解了 **HTTP 无状态协议** 与 **JWT 认证机制** 的配合原理，弄懂了 token 是如何在 Header 中传递并被后端解析的。

在技术细节上，我熟练掌握了 ThreadLocal 在 Spring Boot 并发模型中的应用，这让我对“线程安全”有了具体的实战认知。同时，通过编写 UserLoginDTO、UserVO 等对象，我深刻体会到了 **DTO/VO 分层领域模型** 的重要性——它不仅是为了规范，更是为了安全（如不小心把密码字段返回给前端）。

在团队协作中，由于登录模块是所有业务的“入口”，其他同学的模块都需要依赖我解析出的 userId。这迫使我必须编写清晰的文档和规范的工具类接口，让我切身体会到了“契约优先”和“高内聚低耦合”在团队开发中的价值。这次实训不仅锻炼了我的编码能力，更培养了我的安全意识和全局架构观。

**六、参考文献**

**七、附录**

**(1).entity**

package com.skyfirst.library\_borrowing.entity;  
  
import com.baomidou.mybatisplus.annotation.IdType;  
import com.baomidou.mybatisplus.annotation.TableField;  
import com.baomidou.mybatisplus.annotation.TableId;  
import com.baomidou.mybatisplus.annotation.TableName;  
import lombok.Builder;  
import lombok.Getter;  
import lombok.Setter;  
import lombok.ToString;  
  
import java.io.Serializable;  
import java.time.LocalDateTime;  
  
/\*\*  
 \* <p>  
 \* 用户表  
 \* </p>  
 \*  
 \* @author starnighter  
 \* @since 2025-11-08  
 \*/  
@Getter  
@Setter  
@ToString  
@TableName("user")  
@Builder  
public class User implements Serializable {  
  
 private static final long serialVersionUID = 1L;  
  
 /\*\*  
 \* 主键ID  
 \*/  
 @TableId(value = "id", type = IdType.AUTO)  
 private Long id;  
  
 /\*\*  
 \* 用户名  
 \*/  
 @TableField("username")  
 private String userName;  
  
 /\*\*  
 \* 密码 (BCrypt加密)  
 \*/  
 @TableField("password")  
 private String password;  
  
 /\*\*  
 \* 电子邮箱  
 \*/  
 @TableField("email")  
 private String email;  
  
 /\*\*  
 \* 角色 (USER, ADMIN)  
 \*/  
 @TableField("role")  
 private String role;  
  
 /\*\*  
 \* 逻辑删除 0:未删除 1:已删除  
 \*/  
 @TableField("is\_deleted")  
 private Boolean isDeleted;  
  
 /\*\*  
 \* 创建时间  
 \*/  
 @TableField("gmt\_create")  
 private LocalDateTime gmtCreate;  
  
 /\*\*  
 \* 修改时间  
 \*/  
 @TableField("gmt\_modified")  
 private LocalDateTime gmtModified;  
}

**Service**

**1.IUserService**

package com.skyfirst.library\_borrowing.service;  
  
import com.skyfirst.library\_borrowing.entity.User;  
import com.baomidou.mybatisplus.extension.service.IService;  
  
*/\*\*  
 \* <p>  
 \* 用户表 服务类  
 \* </p>  
 \*  
 \* @author starnighter  
 \* @since 2025-11-08  
 \*/*public interface IUserService extends IService<User> {  
 public User login(String userName,String password);  
  
 public boolean register(User user);  
  
 public User getUserInfo(Long userId);  
  
 String getAllUsersCount();  
}

**2.userServiceImpl**

1. package com.skyfirst.library\_borrowing.service.impl;  
     
   import com.baomidou.mybatisplus.core.conditions.query.LambdaQueryWrapper;  
   import com.skyfirst.library\_borrowing.entity.User;  
   import com.skyfirst.library\_borrowing.mapper.UserMapper;  
   import com.skyfirst.library\_borrowing.service.IUserService;  
   import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;  
   import lombok.RequiredArgsConstructor;  
   import org.springframework.stereotype.Service;  
   import org.springframework.util.DigestUtils;  
     
   */\*\*  
    \* <p>  
    \* 用户表 服务实现类  
    \* </p>  
    \*  
    \* @author starnighter  
    \* @since 2025-11-08  
    \*/*@Service  
   @RequiredArgsConstructor  
   public class UserServiceImpl extends ServiceImpl<UserMapper, User> implements IUserService {  
     
    private final UserMapper userMapper;  
     
     
    @Override  
    public User login(String userName, String password) {  
    LambdaQueryWrapper<User> queryWrapper = new LambdaQueryWrapper<>();  
    queryWrapper.eq(User::getUserName, userName);  
    User user = userMapper.selectOne(queryWrapper);  
     
    if(user == null){  
    throw new RuntimeException("用户名或密码错误");  
    }  
     
    String encryptedInput = DigestUtils.*md5DigestAsHex*(password.getBytes());  
    //String encryptedInput = password;  
    if(!encryptedInput.equals(user.getPassword())){  
    throw new RuntimeException("用户名或密码错误");  
    }  
     
    return user;  
    }  
     
    @Override  
    public boolean register(User user) {  
    LambdaQueryWrapper<User> queryWrapper = new LambdaQueryWrapper<>();  
    queryWrapper.eq(User::getUserName, user.getUserName());  
    Long count = userMapper.selectCount(queryWrapper);  
    if(count>0){  
    throw new RuntimeException("用户名已存在");  
    }  
     
     
    String encryptedPassword = DigestUtils.*md5DigestAsHex*(user.getPassword().getBytes());  
    user.setPassword(encryptedPassword);  
     
     
    return save(user);  
    }  
     
    @Override  
    public User getUserInfo(Long userId) {  
    return userMapper.selectById(userId);  
    }  
     
    @Override  
    public String getAllUsersCount() {  
    return query().eq("is\_deleted",0).count().toString();  
    }  
   }
2. **Controller**

**1 UserController**

package com.skyfirst.library\_borrowing.controller;  
  
import com.skyfirst.library\_borrowing.common.ApiResponse;  
import com.skyfirst.library\_borrowing.common.ResultCode;  
import com.skyfirst.library\_borrowing.common.context.BaseContext;  
import com.skyfirst.library\_borrowing.dto.UserLoginDTO;  
import com.skyfirst.library\_borrowing.dto.UserRegisterDTO;  
import com.skyfirst.library\_borrowing.entity.User;  
import com.skyfirst.library\_borrowing.exception.BusinessException;  
import com.skyfirst.library\_borrowing.service.IUserService;  
import com.skyfirst.library\_borrowing.util.JwtUtils;  
import com.skyfirst.library\_borrowing.vo.LoginVO;  
import com.skyfirst.library\_borrowing.vo.RegisterVO;  
import com.skyfirst.library\_borrowing.vo.UserVO;  
  
import lombok.RequiredArgsConstructor;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.\*;  
  
import java.time.LocalDateTime;  
import java.util.HashMap;  
import java.util.Map;  
  
*/\*\*  
 \* <p>  
 \* 用户表 前端控制器  
 \* </p>  
 \*  
 \* @author starnighter  
 \* @since 2025-11-08  
 \*/*@RestController  
@RequestMapping("/user")  
@RequiredArgsConstructor  
public class UserController {  
  
 @Autowired  
 private final IUserService userService;  
  
 @Autowired  
 private final JwtUtils jwtUtils;  
  
 @PostMapping("/login")  
 public ApiResponse<LoginVO> login(@RequestBody UserLoginDTO loginDTO){  
 User loginuser = userService.login(loginDTO.getUsername(), loginDTO.getPassword());  
 if(loginuser == null){  
 return ApiResponse.*failed*(ResultCode.*PASSWORD\_ERROR*);  
 }  
 Map<String,Object> claims = new HashMap<>();  
 claims.put("userId",loginuser.getId());  
 claims.put("role",loginuser.getRole());  
 String token = jwtUtils.generateJwt(claims);  
 UserVO userVO = UserVO.*builder*()  
 .id(String.*valueOf*(loginuser.getId()) )  
 .username(loginuser.getUserName())  
 .email(loginuser.getEmail())  
 .role(loginuser.getRole())  
 .build();  
 LoginVO loginVO = LoginVO.*builder*()  
 .userInfo(userVO)  
 .token(token)  
 .build();  
 return ApiResponse.*success*(loginVO);  
 }  
  
 @PostMapping("/register")  
 public ApiResponse<RegisterVO> register(@RequestBody UserRegisterDTO userDTO){  
 User user = User.*builder*()  
 .userName(userDTO.getUsername())  
 .password(userDTO.getPassword())  
 .email(userDTO.getEmail())  
 .role("user")  
 .build();  
 try{  
 boolean result = userService.register(user);  
 if(!result){  
 throw new BusinessException("注册失败");  
 }  
  
 RegisterVO registerVO = RegisterVO.*builder*()  
 .id(String.*valueOf*(user.getId()))  
 .userName(user.getUserName())  
 .email(user.getEmail())  
 .role(user.getRole())  
 .createTime(LocalDateTime.*now*())  
 .build();  
 return ApiResponse.*success*(registerVO,"注册成功");  
 }catch (BusinessException e){  
 return ApiResponse.*failed*(e.getCode(),e.getMessage());  
 }  
 }  
  
 @GetMapping("/info")  
 public ApiResponse<UserVO> info(){  
 Long id = BaseContext.*getCurrentId*();  
 User user = userService.getUserInfo(id);  
 if (user == null) {  
 return ApiResponse.*success*(null);  
 }  
 UserVO userVO = UserVO.*builder*()  
 .id(String.*valueOf*(user.getId()) )  
 .username(user.getUserName())  
 .email(user.getEmail())  
 .role(user.getRole())  
 .build();  
 return ApiResponse.*success*(userVO);  
 }  
  
}