

**《软件测试》**

**期末课程设计**

**怡运动——运动爱好者的运动与社交平台**

**单元测试文档**

小组成员

2253407 柳阳

2252721 韩坤甫

2254302 商一诚

2254305 夏济舟

目录

[1. 静态测试 2](#_Toc31483)

[1.1 MessageServiceTest 2](#_Toc4156)

[1.1.1 Constructor问题 2](#_Toc22157)

[1.1.2 本地变量声明为final 2](#_Toc11404)

[1.1.3 变量名过长 2](#_Toc15915)

[1.1.4 函数参数声明为final 3](#_Toc25383)

[1.1.5 只能有一个return 4](#_Toc26781)

[1.1.6 包的命名 4](#_Toc13616)

[1.1.7 其他warning 4](#_Toc18950)

[2. MessageServiceTest单元测试 5](#_Toc10119)

[2.1 测试对象概述 5](#_Toc22508)

[2.2 测试环境 5](#_Toc21811)

[2.3 测试方法 5](#_Toc13930)

[2.4测试用例分析与设计 5](#_Toc14925)

[2.4.1 测试类结构 5](#_Toc24569)

[2.4.2 sendMessage 方法 6](#_Toc9001)

[2.4.3 getChatHistory 方法 7](#_Toc9329)

[2.4.4 deleteMsg 方法 8](#_Toc18885)

[2.5 测试结果 9](#_Toc309)

[3. GroupControllerTest单元测试 9](#_Toc15312)

[3.1 测试对象概述 9](#_Toc9131)

[3.2 测试环境 10](#_Toc13766)

[3.3 测试方法 10](#_Toc12389)

[3.4 测试用例分析与设计 10](#_Toc17385)

[3.4.1 测试类结构 10](#_Toc31914)

[3.4.2 群组管理功能 13](#_Toc16428)

[3.4.3 成员管理功能 14](#_Toc5447)

[3.4.4 申请管理功能 15](#_Toc15430)

[3.4.5 记录查询功能 15](#_Toc1220)

[3.5 测试结果 16](#_Toc8583)

[4. UserServiceTest单元测试 16](#_Toc7288)

[4.1 测试对象概述 16](#_Toc23589)

[4.2 测试环境 17](#_Toc24340)

[4.3 测试方法 17](#_Toc22037)

[4.4 测试用例分析与设计 17](#_Toc24725)

[4.4.1测试类结构 17](#_Toc26044)

[4.4.2 用户认证功能 19](#_Toc20162)

[4.4.3 用户信息管理 20](#_Toc14062)

[4.4.4 通知管理功能 20](#_Toc11936)

[4.4.5 权限验证功能 21](#_Toc10517)

[4.5 测试结果 22](#_Toc32388)

1. **静态测试**

使用PMD插件进行静态测试，以MessageServiceTest为例

* 1. **MessageServiceTest**
     1. **Constructor问题**

**问题描述：**

Each class should declare at least one constructor Each non-static class should declare at least one constructor. Classes with solely static members are ignored, refer to [UseUtilityClassRule] (pmd\_rules\_java\_design. html#useutilityclass) to detect those.

**这条PMD规则要求：**

非静态类必须显式声明至少一个构造函数（即使是无参构造函数），纯静态工具类（只有静态成员的类）例外，这类情况应使用UseUtilityClassRule来检测。

**意义：**

**明确初始化逻辑：**构造函数明确表达了对象创建时的初始化要求

**防止默认构造函数的意外使用：**当类没有显式构造函数时，Java会提供默认无参构造函数，这可能隐藏设计问题

**更好的可维护性：**显式构造函数使类的实例化方式更清晰

* + 1. **本地变量声明为final**

**问题描述：**

Local variable ''{0}'' could be declared final A local variable assigned only once can be declared final.

**这条PMD规则建议：**

如果一个局部变量在初始化后从未被重新赋值（即只赋值一次）

应该将其声明为final，以明确其不可变性

**意义：**

代码清晰性：final明确标识该变量不可变，减少认知负担

防止意外修改：避免在复杂逻辑中意外修改变量值

线程安全：不可变变量在多线程环境中更安全

更好的可维护性：明确设计意图，便于后续修改

* + 1. **变量名过长**

**问题描述：**

Avoid excessively long variable names like {0} Fields, formal arguments, or local variable names that are too long can make the code difficult to follow.

**这条代码风格规则建议：**

避免使用过长的字段、方法参数或局部变量名称

变量名应该在清晰表达意图和简洁性之间保持平衡

**意义：**

可读性：过长的名称会使代码难以快速阅读和理解

维护性：短而清晰的名称更易于维护和修改

一致性：保持团队代码风格统一

IDE支持：现代IDE的代码补全减少了长名称的必要性

* + 1. **函数参数声明为final**

**问题描述：**

Parameter ''{0}'' is not assigned and could be declared final Reports method and constructor parameters that can be made final because they are never reassigned within the body of the method. This rule ignores unused parameters so as not to overlap with the rule {% rule java/bestpractices/UnusedFormalParameter %}. It will also ignore the parameters of abstract methods.

**这条PMD规则检测：**

方法参数或构造函数参数如果未被重新赋值

建议将其声明为final，明确标识其不可变性

**例外情况：**

忽略未使用的参数（由其他规则检查）

忽略抽象方法的参数

**意义：**

明确意图：final参数表明该值在方法内不会被修改

防止意外修改：避免在复杂方法中意外修改参数值

线程安全：不可变参数在多线程环境中更安全

代码可读性：清晰区分方法内会修改和不会修改的变量

* + 1. **只能有一个return**

**问题描述：**

A method should have only one exit point, and that should be the last statement in the method A method should have only one exit point, and that should be the last statement in the method.

**这条结构化编程原则要求**

每个方法只能有一个返回点（即return语句）

这个返回点必须是方法的最后一行语句

**禁止以下情况：**

方法中间的多处return

嵌套条件中的提前返回

**意义：**

可维护性：单一出口使控制流更清晰

可读性：不必追踪多个可能的返回路径

调试便利：更容易设置断点和日志记录

资源清理：确保清理代码（如finally块）总能执行

* + 1. **包的命名**

**描述：**

Package name contains upper case characters Detects when a package definition contains uppercase characters.

**这条Java编码规范要求：**

包名（package name）必须完全使用小写字母

不允许包含任何大写字母（A-Z）

这是Java语言官方规定的命名约定

**意义：**

跨平台兼容性：不同操作系统对大小写敏感度不同

统一代码风格：遵循Java官方代码规范

避免潜在问题：防止因大小写导致的类加载问题

行业标准：所有主流Java项目都遵循此约定

* + 1. **其他warning**

CouplingBetweenObjects (1 violation)

LawOfDemeter (2 violations)

SignatureDeclareThrowsException (9 violations)

SimplifyBooleanReturns (1 violation)

TooManyMethods (1 violation)

CommentRequired (21 violations)

CommentSize (1 violation)

AvoidInstantiatingObjectsInLoops (3 violations)

### MessageServiceTest单元测试

#### ****2.1 测试对象概述****

本次测试针对 **MessageService** 类中的核心功能进行单元测试，包括消息发送、聊天记录获取和消息撤回功能。测试代码覆盖了以下方法：

**sendMessage**：发送消息到指定群聊。

**getChatHistory**：获取指定群聊的历史消息。

**deleteMsg**：撤回指定用户发送的消息。

测试代码通过 Mockito 框架模拟依赖的 Repository 和 Service 层，确保单元测试的独立性和可靠性。

#### 2.2 ****测试环境****

**开发工具**：IntelliJ IDEA

**测试框架**：JUnit 5、Mockito

**构建工具**：Maven

**依赖管理**：Spring Boot

#### ****2.3 测试方法****

采用黑盒与白盒测试相结合的方法：

**黑盒测试**：通过输入输出验证功能逻辑。

**白盒测试**：覆盖代码分支和边界条件。

#### ****2.4测试用例分析与设计****

**2.4.1 测试类结构**

#### Mock 对象

| **Mock 对象** | **用途** |
| --- | --- |
| MessageRepository | 模拟消息数据库操作 |
| ChatMemberRepository | 模拟群聊成员数据库操作 |
| UserService | 模拟用户服务调用 |

#### 测试方法分类

| **功能模块** | **测试方法** |
| --- | --- |
| 消息发送 | sendMessage\_Success, sendMessage\_UserNotInChat |
| 消息历史查询 | getChatHistory\_Success, getChatHistory\_UserNotInChat |
| 消息撤回 | deleteMsg\_Success, deleteMsg\_Failure |

##### ****2.4.2****sendMessage****方法****

**被测特性**：

用户成功发送消息到群聊。

用户非群聊成员时发送失败。

消息内容为空或非法时的处理。

**测试用例设计**：

| **测试项标识符** | **测试项描述** | **优先级** | **预期结果** |
| --- | --- | --- | --- |
| UT\_MSG\_001\_001 | 用户成功发送消息 | 高 | 返回消息实体，包含正确信息 |
| UT\_MSG\_001\_002 | 用户非群聊成员时发送失败 | 高 | 抛出异常，提示“非群聊成员” |
| UT\_MSG\_001\_003 | 消息内容为空 | 中 | 抛出异常，提示“内容不能为空” |

**代码片段验证**：

@Testvoid sendMessage\_Success() {

when(chatMemberRepository.existsChatMemberByChatIdAndUserId(anyInt(), anyInt())).thenReturn(true);

Message result = messageService.sendMessage(messageDTO, userId);

assertNotNull(result);

assertEquals(content, result.getContent());}

##### ****2.4.3****getChatHistory****方法****

**被测特性**：

用户成功获取群聊历史消息。

用户非群聊成员时获取失败。

历史消息为空时的处理。

**测试用例设计**：

| **测试项标识符** | **测试项描述** | **优先级** | **预期结果** |
| --- | --- | --- | --- |
| UT\_MSG\_002\_001 | 用户成功获取历史消息 | 高 | 返回消息列表，包含用户信息 |
| UT\_MSG\_002\_002 | 用户非群聊成员时获取失败 | 高 | 抛出异常，提示“非群聊成员” |
| UT\_MSG\_002\_003 | 群聊无历史消息 | 中 | 返回空列表 |

**代码片段验证**：

@Testvoid getChatHistory\_Success() {

when(chatMemberRepository.existsChatMemberByChatIdAndUserId(anyInt(), anyInt())).thenReturn(true);

when(messageRepository.getHistoryByChatId(anyInt())).thenReturn(messages);

List<MessageUserDTO> result = messageService.getChatHistory(chatId, userId);

assertEquals(2, result.size());}

##### ****2.4.4****deleteMsg****方法****

**被测特性**：

用户成功撤回消息。

消息不存在或用户无权限时撤回失败。

**测试用例设计**：

| **测试项标识符** | **测试项描述** | **优先级** | **预期结果** |
| --- | --- | --- | --- |
| UT\_MSG\_003\_001 | 用户成功撤回消息 | 高 | 返回成功状态 |
| UT\_MSG\_003\_002 | 消息不存在时撤回失败 | 高 | 抛出异常，提示“撤回失败” |

**代码片段验证**：

@Testvoid deleteMsg\_Success() {

when(messageRepository.deleteByMessageIdAndUserIdAndTime(anyInt(), anyInt(), any())).thenReturn(1);

ResultMsg result = messageService.deleteMsg(userId, messageId);

assertEquals("消息撤回成功", result.getMsg());}

#### 2.5 ****测试结果****

| **模块** | **覆盖率** |
| --- | --- |
| 用户登录 | 100% |
| 用户注册 | 100% |
| 用户信息查询 | 100% |
| 用户信息更新 | 100% |
| 用户通知管理 | 100% |
| 权限检查 | 100% |

### GroupControllerTest单元测试

#### 3.1 ****测试对象概述****

本次测试针对 **GroupController** 类中的核心功能进行单元测试，主要覆盖以下功能模块：

**群组管理**：创建群组、获取群组信息、删除群组。

**成员管理**：加入/退出群组、成员角色设置、成员移除。

**申请管理**：发送申请、处理申请、获取申请列表。

**记录查询**：群组操作记录查询。

测试代码通过 MockMvc 模拟 HTTP 请求，结合 Mockito 框架模拟依赖的 Service 层，确保单元测试的独立性和可靠性。

#### 3.2 ****测试环境****

**开发工具**：IntelliJ IDEA

**测试框架**：JUnit 5、Mockito、Spring Test

**构建工具**：Maven

**依赖管理**：Spring Boot

#### 3.3 ****测试方法****

采用黑盒与白盒测试相结合的方法：

**黑盒测试**：通过模拟 HTTP 请求验证接口功能。

**白盒测试**：覆盖代码分支和异常场景。

#### 3.4 ****测试用例分析与设计****

**3.4.1 测试类结构**

#### Mock 对象

| **Mock 对象** | **用途** |
| --- | --- |
| GroupService | 模拟群组相关操作 |
| GroupApplicationService | 模拟群组申请管理 |
| GroupMemberService | 模拟群组成员管理 |
| GroupRecordService | 模拟群组记录管理 |
| GroupMemberRepository | 模拟群组成员数据库操作 |
| GroupRepository | 模拟群组数据库操作 |

#### 测试方法分类

| **功能模块** | **测试方法** |
| --- | --- |
| 群组管理 | testCreateGroup, testGetGroups, getGroupByUser\_Success, getGroupByUser\_NoGroups, getUserLeaderGroup\_Success, getUserLeaderGroup\_NoLeaderGroups, deleteGroup\_Success, deleteGroup\_NoPermission, deleteGroup\_GroupNotFound |
| 群组申请管理 | getGroupApplication\_Success, getGroupApplication\_NoApplications, sendGroupApplication\_Success,  sendGroupApplication\_InvalidGroup, sendGroupApplication\_DuplicateApplication, sendGroupApplication\_InvalidApplicant, inviteMember\_Success, inviteMember\_InvalidGroup, inviteMember\_NoPermission, inviteMember\_AlreadyMember, inviteMember\_DuplicateInvitation, updateGroupApplication\_Success, updateGroupApplication\_ApplicationNotFound, updateGroupApplication\_NoPermission, updateGroupApplication\_AlreadyProcessed, updateGroupApplication\_Expired |
| 群组信息查询 | getGroupByID\_Success, getGroupByID\_NotFound, getGroupByName\_Success,  getGroupByName\_NoResults, getGroupByName\_EmptyName |
| 群组成员管理 | deleteGroupMember\_Success, deleteGroupMember\_NotMember,  deleteGroupMember\_GroupNotFound, deleteGroupMember\_LastLeader,  removeGroupMember\_Success, removeGroupMember\_NoPermission,  removeGroupMember\_GroupNotFound, removeGroupMember\_MemberNotFound, removeGroupMember\_CannotRemoveLeader, setGroupMemberRole\_Success, setGroupMemberRole\_NoPermission, setGroupMemberRole\_GroupNotFound, setGroupMemberRole\_MemberNotFound, setGroupMemberRole\_InvalidRole |
| 群组记录管理 | getGroupRecords\_Success, getGroupRecords\_NoRecords, getGroupRecords\_GroupNotFound, getGroupRecords\_NoPermission |

##### ****3.4.2 群组管理功能****

**被测接口**：

**POST /api/groups**：创建群组。

**GET /api/groups**：获取所有群组。

**DELETE /api/groups**：删除群组。

**测试用例设计**：

| **测试项标识符** | **测试项描述** | **优先级** | **预期结果** |
| --- | --- | --- | --- |
| UT\_GROUP\_001\_001 | 成功创建群组 | 高 | 返回成功状态，状态码 200 |
| UT\_GROUP\_001\_002 | 获取所有群组（非空列表） | 高 | 返回群组列表，状态码 200 |
| UT\_GROUP\_001\_003 | 删除群组（无权限） | 中 | 返回错误信息，状态码 400 |

**代码片段验证**：

@Testvoid testCreateGroup() throws Exception {

mockMvc.perform(post("/api/groups")

.contentType(MediaType.APPLICATION\_JSON)

.content(objectMapper.writeValueAsString(dto)))

.andExpect(status().isOk());}

##### ****3.4.3 成员管理功能****

**被测接口**：

**DELETE /api/groups/members**：退出群组。

**PATCH /api/groups/members**：设置成员角色。

**测试用例设计**：

| **测试项标识符** | **测试项描述** | **优先级** | **预期结果** |
| --- | --- | --- | --- |
| UT\_MEMBER\_002\_001 | 成功退出群组 | 高 | 返回成功状态，状态码 200 |
| UT\_MEMBER\_002\_002 | 设置成员角色（无权限） | 中 | 返回错误信息，状态码 400 |

**代码片段验证**：

@Testvoid deleteGroupMember\_Success() throws Exception {

mockMvc.perform(delete("/api/groups/members")

.param("groupId", "1"))

.andExpect(status().isOk());}

##### ****3.4.4 申请管理功能****

**被测接口**：

**POST /api/groups/application**：发送申请。

**PATCH /api/groups/application**：处理申请。

**测试用例设计**：

| **测试项标识符** | **测试项描述** | **优先级** | **预期结果** |
| --- | --- | --- | --- |
| UT\_APPLY\_003\_001 | 成功发送申请 | 高 | 返回成功状态，状态码 200 |
| UT\_APPLY\_003\_002 | 处理申请（申请不存在） | 中 | 返回错误信息，状态码 400 |

**代码片段验证**：

@Testvoid sendGroupApplication\_Success() throws Exception {

mockMvc.perform(post("/api/groups/application")

.contentType(MediaType.APPLICATION\_JSON)

.content(objectMapper.writeValueAsString(applicationDTO)))

.andExpect(status().isOk());}

##### ****3.4.5 记录查询功能****

**被测接口**：

**GET /api/groups/records**：获取操作记录。

**测试用例设计**：

| **测试项标识符** | **测试项描述** | **优先级** | **预期结果** |
| --- | --- | --- | --- |
| UT\_RECORD\_004\_001 | 成功获取记录 | 高 | 返回记录列表，状态码 200 |
| UT\_RECORD\_004\_002 | 获取记录（无权限） | 中 | 返回错误信息，状态码 400 |

**代码片段验证**：

@Testvoid getGroupRecords\_Success() throws Exception {

mockMvc.perform(get("/api/groups/records")

.param("groupId", "1"))

.andExpect(status().isOk());}

#### 3.5 ****测试结果****

| **模块** | **覆盖率** |
| --- | --- |
| 群组管理 | 100% |
| 群组申请管理 | 100% |
| 群组信息查询 | 100% |
| 群组成员管理 | 100% |
| 群组记录管理 | 100% |

### UserServiceTest单元测试

#### 4.1 ****测试对象概述****

本次测试针对 **UserService** 类中的核心功能进行单元测试，主要覆盖以下功能模块：

**用户认证**：登录、注册

**用户信息管理**：获取/更新用户信息、修改密码、头像管理

**通知管理**：发送/获取/编辑通知

**权限验证**：管理员身份验证

测试代码通过 Mockito 框架模拟依赖的 Repository 和 Service 层，确保单元测试的独立性和可靠性。

#### 4.2 ****测试环境****

**开发工具**：IntelliJ IDEA

**测试框架**：JUnit 5、Mockito

**构建工具**：Maven

**依赖管理**：Spring Boot

#### 4.3 ****测试方法****

采用黑盒与白盒测试相结合的方法：

**黑盒测试**：通过输入输出验证功能逻辑

**白盒测试**：覆盖代码分支和异常场景

#### 4.4 ****测试用例分析与设计****

**4.4.1测试类结构**

#### Mock 对象

| **Mock 对象** | **用途** |
| --- | --- |
| UserRepository | 模拟数据库用户操作 |
| JwtTokenProvider | 模拟 JWT Token 生成与验证 |
| OssService | 模拟文件存储服务（如用户头像） |
| VenueService | 模拟场馆管理员相关操作 |
| ViolationService | 模拟违规记录管理 |
| NotificationRepository | 模拟用户通知管理 |

#### 测试方法分类

| **功能模块** | **测试方法** |  |
| --- | --- | --- |
| 用户登录 | login\_Success, login\_UserNotFound, login\_WrongPassword |  |
| 用户注册 | register\_Success\_RegularUser, register\_Success\_VenueAdmin,  register\_UserAlreadyExists |  |
| 用户信息查询 | getUserList\_Success, getUserList\_EmptyList,  getUsersByName\_Success,  getUsersByName\_NoUsersFound, getUserInfo\_Success,  getUserInfo\_UserNotFound, getUserProfile\_Success,  getUserProfile\_UserNotFound, getUserPhoto\_Success |  |
| 用户信息更新 | updateUserInfo\_Success, updateUserInfo\_UserNotFound,  updateUserPwd\_Success, updateUserPwd\_UserNotFound,  updateUserPwd\_WrongOldPassword, updateUserAvatar\_Success,  updateUserAvatar\_UploadError |  |
| 用户通知管理 | getUserNotification\_Success, getUserNotification\_EmptyList,  sendUserNotification\_Success, editUserNotification\_Success  , editUserNotification\_NotFound |  |
| 权限检查 | isUserAdmin\_True, isUserAdmin\_False,  isUserAdmin\_UserNotFound |  |
|  |  |  |

##### ****4.4.2 用户认证功能****

**被测方法**：

**login()**：用户登录

**register()**：用户注册

**测试用例设计**：

| **测试项标识符** | **测试项描述** | **优先级** | **预期结果** |
| --- | --- | --- | --- |
| UT\_AUTH\_001\_001 | 成功登录 | 高 | 返回完整用户信息和Token |
| UT\_AUTH\_001\_002 | 用户不存在 | 高 | 抛出异常（400错误） |
| UT\_AUTH\_001\_003 | 密码错误 | 高 | 抛出异常（400错误） |
| UT\_AUTH\_001\_004 | 普通用户注册成功 | 高 | 返回用户ID和用户名 |
| UT\_AUTH\_001\_005 | 场馆管理员注册成功 | 高 | 返回用户ID和用户名 |
| UT\_AUTH\_001\_006 | 用户名已存在 | 中 | 抛出异常（422错误） |

**代码片段验证**：

@Testvoid login\_Success() {

*// 测试数据准备和mock配置*

LoginResponseDTO response = userService.login(userName, password);

*// 断言验证*

assertEquals(token, response.getToken());}

##### ****4.4.3 用户信息管理****

**被测方法**：

**getUserInfo()**：获取用户详情

**updateUserInfo()**：更新用户信息

**updateUserPwd()**：修改密码

**测试用例设计**：

| **测试项标识符** | **测试项描述** | **优先级** | **预期结果** |
| --- | --- | --- | --- |
| UT\_INFO\_002\_001 | 成功获取用户信息 | 高 | 返回完整用户信息 |
| UT\_INFO\_002\_002 | 用户不存在 | 中 | 抛出异常（404错误） |
| UT\_INFO\_002\_003 | 成功更新用户信息 | 高 | 返回更新后的用户信息 |
| UT\_INFO\_002\_004 | 成功修改密码 | 高 | 返回用户ID |
| UT\_INFO\_002\_005 | 旧密码错误 | 中 | 抛出异常（400错误） |

**代码片段验证**：

@Test void updateUserPwd\_Success() {

IdResponseDTO result = userService.updateUserPwd(userId, updatePwdDTO);

assertEquals(userId, result.getUserId());}

##### ****4.4.4 通知管理功能****

**被测方法**：

**getUserNotification()**：获取用户通知

**sendUserNotification()**：发送通知

**editUserNotification()**：编辑通知状态

**测试用例设计**：

| **测试项标识符** | **测试项描述** | **优先级** | **预期结果** |
| --- | --- | --- | --- |
| UT\_NOTIFY\_003\_001 | 成功获取通知列表 | 高 | 返回通知列表 |
| UT\_NOTIFY\_003\_002 | 空通知列表 | 低 | 返回空列表 |
| UT\_NOTIFY\_003\_003 | 成功发送通知 | 高 | 返回成功状态 |
| UT\_NOTIFY\_003\_004 | 成功标记通知为已读 | 中 | 返回成功状态 |
| UT\_NOTIFY\_003\_005 | 通知不存在 | 低 | 抛出异常 |

**代码片段验证**：

@Testvoid sendUserNotification\_Success() {

ResultMsg result = userService.sendUserNotification(notificationContentDTO);

assertEquals("消息发送成功", result.getMsg());}

##### ****4.4.5 权限验证功能****

**被测方法**：

**isUserAdmin()**：验证管理员身份

**测试用例设计**：

| **测试项标识符** | **测试项描述** | **优先级** | **预期结果** |
| --- | --- | --- | --- |
| UT\_AUTH\_004\_001 | 用户是管理员 | 高 | 返回true |
| UT\_AUTH\_004\_002 | 用户不是管理员 | 高 | 返回false |
| UT\_AUTH\_004\_003 | 用户不存在 | 低 | 返回false |

**代码片段验证**：

@Testvoid isUserAdmin\_True() {

Boolean result = userService.isUserAdmin(userId);

assertTrue(result);}

#### 4.5 ****测试结果****

**通过率**：100%

**覆盖范围**：覆盖所有核心业务逻辑和异常场景

**问题发现**：无

| **模块** | **覆盖率** |
| --- | --- |
| 用户登录 | 100% |
| 用户注册 | 100% |
| 用户信息查询 | 100% |
| 用户信息更新 | 100% |
| 用户通知管理 | 100% |
| 权限检查 | 100% |