

# 软件详细设计说明书

《软件详细设计说明书》是对《软件概要设计说明书》的补充。

1. 本课程及本实验不涉及、不强调、不关注的部分可省略，但不允许删除已有条目。
2. 最终文档请删除所有灰色字体和斜体等说明部分。

## ▼ 软件详细设计说明书

### ▼ 1 引言

- 1.1 编写目的
- ▼ 1.2 背景
  - a. 系统名称
  - b. 项目相关方
- 1.3 术语表
- 1.4 参考文档

### ▼ 2 系统结构设计及子系统划分

- 2.1 系统整体架构
- 2.2 模块间依赖关系
- 2.3 子系统划分

### ▼ 3 功能模块设计

#### ▼ 3.1 Common模块

- 3.1.1 模块概述
- 3.1.2 组件结构
- 3.1.3 核心API接口

#### ▼ 3.2 Core模块

- 3.2.1 模块概述
- 3.2.2 组件结构
- 3.2.3 核心领域模型

#### ▼ 3.3 User-Service模块

- 3.3.1 模块概述
- 3.3.2 数据库表设计
- 3.3.3 组件结构
- 3.3.4 主要API接口
- 3.3.4 处理流程

#### ▼ 3.4 Discussion-Service模块

- 3.4.1 模块概述

- 3.4.2 数据库表设计
- 3.4.2 组件结构
- 3.4.3 主要API接口
- ▼ 3.5 Evaluation-Service模块
  - 3.5.1 模块概述
  - 3.5.2 数据库表设计
  - 3.5.2 组件结构
  - 3.5.3 主要API接口
- ▼ 3.6 Experiment-Service模块
  - 3.6.1 模块概述
  - 3.6.2 数据库表设计
  - 3.6.2 组件结构
  - 3.6.3 主要API接口
  - 3.6.4 处理流程
- ▼ 3.7 Support-Service模块
  - 3.7.1 模块概述
  - 3.7.2 数据库表设计
  - 3.7.2 组件结构
  - 3.7.3 主要API接口
- ▼ 4 界面设计
  - ▼ 4.1 外部界面设计
    - 4.1.1 API设计原则
    - 4.1.2 全局响应格式
    - 4.1.3 错误处理
  - ▼ 4.2 内部界面设计
    - 4.2.1 模块间通信
    - 4.2.2 数据接口
  - ▼ 4.3 用户界面设计
    - 4.3.1 界面风格
    - 4.3.2 主要页面布局
    - 4.3.3 交互设计
- ▼ 5 异常处理设计
  - ▼ 5.1 出错信息管理
    - 5.1.1 异常分类
    - 5.1.2 异常编码规则
    - 5.1.3 异常信息国际化
  - ▼ 5.2 故障预防与补救
    - 5.2.1 故障预防策略

- 5.2.2 故障恢复机制
- ▼ 5.3 系统维护设计
  - 5.3.1 监控告警
  - 5.3.2 运维支持
- ▼ 6 性能优化和安全设计
  - ▼ 6.1 性能优化设计
    - 6.1.1 数据库优化
    - 6.1.2 缓存策略
    - 6.1.3 JVM调优
    - 6.1.4 分布式优化
  - ▼ 6.2 安全设计
    - 6.2.1 认证与授权
    - 6.2.2 数据安全
    - 6.2.3 安全防护
- ▼ 7 项目测试计划
  - ▼ 7.1 测试策略
    - 7.1.1 测试方法论
    - 7.1.2 测试环境
  - 7.2 测试计划详情
  - ▼ 7.3 测试技术和工具
    - 7.3.1 单元测试
    - 7.3.2 集成测试
    - 7.3.3 性能测试
    - 7.3.4 安全测试
  - 7.4 测试覆盖率目标
  - 7.5 测试流程和规范

# 1 引言

介绍编写该设计文档的目的、背景和范围。说明文档的读者和主要参考资料。

## 1.1 编写目的

本文档的主要目的是详细描述"灵狐智验"系统的软件架构、各模块的功能设计、接口规范和实现细节。作为开发团队进行系统实现的技术指导文档，它基于《软件概要设计说明书》中定义的高层设计，并提供各组件的详细设计和交互规范。

本文档的预期读者包括：

- 开发团队成员（程序员、测试人员）
- 项目管理人员和技术负责人
- 系统维护人员
- 技术文档审核人员

## 1.2 背景

### a. 系统名称

待开发的软件系统名称为"灵狐智验"实验管理平台。

### b. 项目相关方

- **任务提出者**：高校实验教学示范中心
- **开发者**：软件工程系统开发团队
- **用户**：高校教师、学生、实验室管理人员
- **运行环境**：基于云服务架构，部署于校园数据中心

## 1.3 术语表

术语	定义
API	Application Programming Interface（应用程序接口），定义了软件组件之间的交互方式
DTO	Data Transfer Object（数据传输对象），用于在不同层次间传输数据的对象
RBAC	Role-Based Access Control（基于角色的访问控制），一种安全模型
JWT	JSON Web Token，用于在网络应用间安全传递声明的开放标准
CI/CD	Continuous Integration/Continuous Deployment，持续集成/持续部署
ORM	Object-Relational Mapping，对象关系映射
RESTful	Representational State Transfer，一种API设计风格
微服务	将应用程序构建为一组小型服务的架构风格

## 1.4 参考文档

1. 《灵狐智验需求描述文档》v1.2，2025年3月
2. 《软件需求规格说明书》v1.0，2025年4月
3. 《软件概要设计说明书》v1.0，2025年4月

4. Spring Boot 官方文档，<https://spring.io/projects/spring-boot>
5. Spring Security 参考手册，<https://docs.spring.io/spring-security/reference/>
6. 《微服务架构设计模式》，Chris Richardson 著

## 2 系统结构设计及子系统划分


系统采用微服务架构模式，将整个应用程序划分为七个核心模块，每个模块负责特定的功能区域，既能独立运行，又能通过定义良好的接口协同工作。

### 2.1 系统整体架构

系统由以下七个核心模块组成：

1. **Common模块**：公共组件库，提供系统共享的工具类、通用异常处理、常量定义等基础设施
2. **Core模块**：核心业务逻辑和领域模型，包含系统的核心业务规则和领域对象
3. **User-Service模块**：用户服务，负责用户管理、认证和授权
4. **Discussion-Service模块**：讨论服务，处理用户之间的交流与互动
5. **Evaluation-Service模块**：评估服务，提供评价与反馈功能
6. **Experiment-Service模块**：实验服务，管理实验相关的功能
7. **Support-Service模块**：支持服务，提供系统支持和辅助功能

系统架构如下图所示：

 系统总体架构图

### 2.2 模块间依赖关系

各模块之间的依赖关系如下：

- Common模块作为基础设施，被所有其他模块依赖
- Core模块依赖Common模块，并被所有业务服务模块依赖
- User-Service、Discussion-Service、Evaluation-Service、Experiment-Service和Support-Service相互独立，通过API接口进行通信
- 所有服务模块都依赖于Core和Common模块


### 2.3 子系统划分


系统根据功能划分为以下几个主要子系统：

1. **用户管理子系统**：负责用户注册、登录、权限控制等功能，由User-Service模块实现

- 2. **讨论交流子系统**：负责用户间的讨论、消息和通知等功能，由Discussion-Service模块实现
- 3. **评估反馈子系统**：负责收集和处理用户评价与反馈，由Evaluation-Service模块实现
- 4. **实验管理子系统**：负责创建、编辑、运行实验等功能，由Experiment-Service模块实现
- 5. **支持服务子系统**：负责提供系统辅助功能，由Support-Service模块实现

以实验管理子系统为例，其组件图和顺序图如下：

实验管理子系统-组件图

实验管理子系统-顺序图

## 3 功能模块设计

本系统包含7个主要功能模块，每个模块负责特定的功能，具有明确的输入、处理逻辑和输出。各模块设计遵循高内聚、低耦合的原则，保证系统的可维护性和可扩展性。

### 3.1 Common模块

#### 3.1.1 模块概述

Common模块是系统的公共组件库，为所有其他模块提供基础设施支持。它包含跨模块共享的工具类、通用异常处理机制、常量定义、通用数据模型和辅助功能。

#### 3.1.2 组件结构

组件名称	功能描述
utils	通用工具类集合，提供字符串、日期处理等功能
exception	异常处理框架，定义系统通用异常类型和处理机制
constants	系统常量定义，包括错误代码、状态码等
dto	数据传输对象（DTO）定义
validation	数据验证工具和注解
security	安全相关工具，如加密、解密等
config	公共配置类

### 3.1.3 核心API接口

```
// 结果包装类，用于统一接口返回格式
public class Result<T> {
    private int code;           // 状态码
    private String message;     // 消息
    private T data;             // 数据

    // 构造方法、getter和setter方法
    // 成功/失败静态工厂方法
}

// 分页结果包装类
public class PageResult<T> {
    private List<T> list;       // 数据列表
    private long total;         // 总记录数
    private int pageNum;        // 页码
    private int pageSize;       // 每页记录数

    // 构造方法、getter和setter方法
}

// 异常基类
public abstract class BaseException extends RuntimeException {
    private final int code;      // 错误码
    private final String message; // 错误信息

    // 构造方法、getter方法
}
```

## 3.2 Core模块

### 3.2.1 模块概述

Core模块包含系统的核心业务逻辑和领域模型，定义系统的基本业务规则和领域对象。它依赖于Common模块，并被所有业务服务模块依赖。

### 3.2.2 组件结构

组件名称	功能描述
domain	核心领域模型，定义业务实体和值对象

组件名称	功能描述
repository	仓储接口，定义数据访问抽象
service	核心业务服务，实现系统主要业务逻辑
event	领域事件定义，支持事件驱动架构
strategy	业务策略接口和实现，支持可替换的业务逻辑

### 3.2.3 核心领域模型

系统的主要领域模型包括：

```
// 用户领域模型
public class User {
    private String id;           // 用户ID
    private String username;     // 用户名
    private String email;       // 邮箱
    private String password;     // 密码（加密存储）
    private UserRole role;      // 用户角色
    private Date createTime;    // 创建时间
    private Date updateTime;    // 更新时间

    // 构造方法、getter和setter方法
    // 业务方法
}

// 实验领域模型
public class Experiment {
    private String id;           // 实验ID
    private String name;        // 实验名称
    private String description;  // 实验描述
    private String userId;      // 创建者ID
    private ExperimentStatus status; // 实验状态
    private Date createTime;    // 创建时间
    private Date updateTime;    // 更新时间
    private List<ExperimentStep> steps; // 实验步骤

    // 构造方法、getter和setter方法
    // 业务方法
}
```



### 3.3 User-Service模块

#### 3.3.1 模块概述

User-Service模块负责用户管理、认证和授权功能，是系统的身份验证中心。它管理用户信息，处理用户注册、登录、个人信息管理等功能。

#### 3.3.2 数据库表设计

表名	说明	主要字段
user	用户基本信息表	id, username, email, password, role, create_time, update_time
user_profile	用户详细资料表	id, user_id, real_name, avatar, gender, birthday, phone, address, education, introduction
role	角色表	id, role_name, description, create_time, update_time
permission	权限表	id, permission_name, description, resource_type, resource_path
role_permission	角色权限关系表	id, role_id, permission_id
user_token	用户令牌表	id, user_id, token, expire_time, create_time
login_log	登录日志表	id, user_id, login_ip, login_device, login_time, status

#### 3.3.3 组件结构

组件名称	功能描述
controller	REST API控制器，处理HTTP请求
service	业务服务层，实现用户管理核心逻辑
repository	数据访问层，与数据库交互
security	安全配置，包括认证、授权等
dto	数据传输对象，用于API请求和响应

### 3.3.4 主要API接口

POST /api/users/register	// 用户注册
POST /api/users/login	// 用户登录
GET /api/users/profile	// 获取用户个人资料
PUT /api/users/profile	// 更新用户个人资料
PUT /api/users/password	// 修改密码
GET /api/users/{id}	// 获取指定用户信息
GET /api/users	// 分页查询用户列表

### 3.3.4 处理流程

以用户注册流程为例：

1. 客户端发送包含用户名、密码等信息的注册请求
2. Controller接收请求，进行数据验证
3. Service层检查用户名是否已存在
4. 如果用户名不存在，对密码进行加密
5. 创建新用户记录并保存到数据库
6. 返回注册成功响应，包含用户ID和token

## 3.4 Discussion-Service模块

### 3.4.1 模块概述

Discussion-Service模块负责用户之间的交流与互动功能，包括讨论、评论、消息通知等。它支持实时和异步的通信方式，促进用户之间的协作和知识共享。

### 3.4.2 数据库表设计

表名	说明	主要字段
discussion	讨论主题表	id, title, content, user_id, category_id, status, view_count, like_count, create_time, update_time
discussion_category	讨论分类表	id, name, description, icon, sort_order
comment	评论表	id, discussion_id, user_id, content, parent_id, like_count, create_time
message	私信表	id, sender_id, receiver_id, content, read_status, create_time

表名	说明	主要字段
notification	通知表	id, user_id, content, type, related_id, read_status, create_time
user_follow	用户关注关系表	id, follower_id, followee_id, create_time

### 3.4.2 组件结构

组件名称	功能描述
controller	REST API控制器，处理HTTP请求
service	业务服务层，实现讨论功能核心逻辑
repository	数据访问层，与数据库交互
socket	WebSocket服务，支持实时通信
dto	数据传输对象，用于API请求和响应

### 3.4.3 主要API接口

```
POST /api/discussions           // 创建讨论主题
GET /api/discussions            // 获取讨论主题列表
GET /api/discussions/{id}       // 获取讨论主题详情
POST /api/discussions/{id}/comments // 添加评论
GET /api/discussions/{id}/comments // 获取评论列表
POST /api/messages              // 发送私信
GET /api/messages               // 获取私信列表
GET /api/notifications          // 获取通知列表
```

## 3.5 Evaluation-Service模块

### 3.5.1 模块概述

Evaluation-Service模块负责评价与反馈功能，包括实验评价、用户评分、反馈收集等。它提供了评价数据的收集、分析和展示功能，支持系统质量改进。

3.5.2 数据库表设计

表名	说明	主要字段
evaluation	评价表	id, target_id, target_type, user_id, score, content, create_time
evaluation_dimension	评价维度表	id, name, description, weight
evaluation_detail	评价详情表	id, evaluation_id, dimension_id, score, content
feedback	反馈表	id, user_id, type, content, status, create_time, handle_time
evaluation_statistics	评价统计表	id, target_id, target_type, avg_score, count, update_time

3.5.2 组件结构

组件名称	功能描述
controller	REST API控制器，处理HTTP请求
service	业务服务层，实现评价功能核心逻辑
repository	数据访问层，与数据库交互
analyzer	评价数据分析器，产生统计和分析结果
dto	数据传输对象，用于API请求和响应

3.5.3 主要API接口

```
POST /api/evaluations/experiments/{id} // 提交实验评价
GET  /api/evaluations/experiments/{id} // 获取实验评价
POST /api/feedback                      // 提交系统反馈
GET  /api/evaluations/statistics        // 获取评价统计数据
```

3.6 Experiment-Service模块

3.6.1 模块概述

Experiment-Service模块负责实验相关的功能，包括实验创建、编辑、执行、结果分析等。它是系统的核心模块之一，支持用户进行各类实验活动。

3.6.2 数据库表设计

表名	说明	主要字段
experiment	实验信息表	id, name, description, user_id, status, category_id, create_time, update_time
experiment_step	实验步骤表	id, experiment_id, title, description, order_num, required
experiment_parameter	实验参数表	id, experiment_id, name, type, default_value, description
experiment_resource	实验资源表	id, experiment_id, resource_type, resource_path, description
experiment_result	实验结果表	id, experiment_id, user_id, status, start_time, end_time, report_path
experiment_data	实验数据表	id, result_id, step_id, data_key, data_value, data_type, create_time
experiment_category	实验分类表	id, name, description, parent_id, level, sort_order
experiment_favorite	实验收藏表	id, experiment_id, user_id, create_time

3.6.2 组件结构

组件名称	功能描述
controller	REST API控制器，处理HTTP请求
service	业务服务层，实现实验功能核心逻辑
repository	数据访问层，与数据库交互
executor	实验执行器，负责运行实验并收集结果
analyzer	结果分析器，分析实验结果并生成报告
dto	数据传输对象，用于API请求和响应

### 3.6.3 主要API接口

```
POST /api/experiments           // 创建实验
GET  /api/experiments           // 获取实验列表
GET  /api/experiments/{id}      // 获取实验详情
PUT  /api/experiments/{id}      // 更新实验
POST /api/experiments/{id}/run  // 运行实验
GET  /api/experiments/{id}/results // 获取实验结果
POST /api/experiments/{id}/share // 分享实验
```

### 3.6.4 处理流程

以实验创建流程为例：

- 1. 用户提交实验创建请求，包含实验名称、描述、参数等
- 2. Controller接收请求并验证数据
- 3. Service层处理实验创建逻辑
- 4. 保存实验基本信息到数据库
- 5. 建立实验步骤和参数
- 6. 返回创建成功的实验ID和基本信息

## 3.7 Support-Service模块

### 3.7.1 模块概述

Support-Service模块提供系统支持和辅助功能，包括文件管理、数据导入导出、系统监控、审计日志等辅助功能，支持其他模块的正常运行。

### 3.7.2 数据库表设计

表名	说明	主要字段
file_storage	文件存储表	id, original_name, storage_path, file_size, file_type, upload_time, user_id, status
system_log	系统日志表	id, log_type, operation, operator_id, operation_time, ip_address, request_url, request_method, params, result
system_config	系统配置表	id, config_key, config_value, description, create_time, update_time

表名	说明	主要字段
export_task	导出任务表	id, task_type, parameters, status, file_path, user_id, create_time, finish_time
system_monitor	系统监控表	id, monitor_time, cpu_usage, memory_usage, disk_usage, network_io, active_users

### 3.7.2 组件结构

组件名称	功能描述
controller	REST API控制器，处理HTTP请求
service	业务服务层，实现支持功能核心逻辑
repository	数据访问层，与数据库交互
storage	存储服务，管理文件存储
monitor	系统监控服务，收集系统运行数据
dto	数据传输对象，用于API请求和响应

### 3.7.3 主要API接口

```
POST /api/files           // 上传文件
GET  /api/files/{id}      // 下载文件
POST /api/export          // 导出数据
POST /api/import          // 导入数据
GET  /api/system/status   // 获取系统状态
GET  /api/logs            // 获取审计日志
```

## 4 界面设计

本系统采用前后端分离架构，前端使用现代Web技术栈构建用户界面，后端提供RESTful API接口。界面设计遵循简洁、直观、一致的原则，提供良好的用户体验。

## 4.1 外部界面设计

### 4.1.1 API设计原则

系统所有外部接口遵循RESTful API设计规范：

1. 使用HTTP标准方法（GET、POST、PUT、DELETE）表示操作
2. 使用URL表示资源，使用HTTP方法表示行为
3. 使用HTTP状态码表示响应结果
4. 使用JSON格式进行数据交换
5. 提供版本控制（如通过URL路径或请求头）

### 4.1.2 全局响应格式

所有API响应都使用统一的数据格式：

```
{
  "code": 200,           // 状态码，200表示成功，其他表示失败
  "message": "success", // 消息描述
  "data": {              // 响应数据，可能是对象、数组或null
    // 具体数据
  }
}
```

分页响应格式：

```
{
  "code": 200,
  "message": "success",
  "data": {
    "list": [            // 数据列表
      // 列表项
    ],
    "total": 100,         // 总记录数
    "pageNum": 1,        // 当前页码
    "pageSize": 10       // 每页大小
  }
}
```

### 4.1.3 错误处理

系统使用HTTP状态码和业务状态码相结合的方式处理错误：



- 2xx: 成功
- 4xx: 客户端错误 (如参数错误、未授权)
- 5xx: 服务端错误

业务错误响应示例:

```
{
  "code": 400100,           // 业务错误码
  "message": "用户名已存在", // 错误描述
  "data": null              // 无数据返回
}
```

## 4.2 内部界面设计

### 4.2.1 模块间通信

系统内部各功能模块间通过以下方式进行通信:

1. **同步调用**: 模块间通过RESTful API进行同步调用
2. **异步消息**: 使用消息队列 (如RabbitMQ) 进行异步通信
3. **事件驱动**: 通过发布/订阅模式处理跨模块事件

### 4.2.2 数据接口

模块间共享的主要数据接口包括:

1. **用户认证信息**：由User-Service提供，其他模块通过JWT令牌获取用户信息
2. **实验数据**：由Experiment-Service提供，供其他模块引用
3. **评价数据**：由Evaluation-Service提供，供统计分析使用
4. **通知事件**：由各模块发布，Discussion-Service订阅并处理

## 4.3 用户界面设计

### 4.3.1 界面风格

系统前端界面采用现代化、扁平化设计风格，主要设计元素包括：

1. **配色方案：**以蓝色为主色调，白色为背景，辅以灰色和浅色调
2. **字体选择：**正文使用无衬线字体，标题使用粗体增强视觉层次
3. **响应式设计：**适应不同设备尺寸，包括桌面、平板和手机

## 4.3.2 主要页面布局

系统包含以下主要页面：

- 登录注册页：**简洁的表单设计，包含用户名/密码输入框和提交按钮
- 个人中心页：**左侧导航菜单，右侧内容区展示个人信息和操作选项
- 实验管理页：**顶部操作栏，中部列表/网格视图展示实验项目
- 实验详情页：**分步骤展示实验内容，支持交互操作
- 讨论页面：**类论坛样式，支持发帖、回复等社区互动功能

## 4.3.3 交互设计

系统交互设计遵循以下原则：

- 即时反馈：**用户操作后立即给予视觉或文字反馈
- 渐进式展示：**复杂功能分步骤引导用户完成
- 状态可见：**系统状态和进度清晰可见
- 容错设计：**提供操作撤销和错误恢复机制

# 5 异常处理设计

系统异常处理采用分层设计，确保异常能够被正确捕获和处理，提供清晰的错误信息给用户，并保持系统的稳定性和可用性。

## 5.1 出错信息管理

### 5.1.1 异常分类

系统异常分为以下几类：

- 业务异常：**与业务规则相关的异常，如用户名已存在、权限不足等
- 系统异常：**系统内部错误，如数据库连接失败、服务不可用等
- 参数异常：**请求参数错误，如参数格式不正确、必填参数缺失等
- 认证授权异常：**认证失败或授权不足的异常
- 外部服务异常：**调用外部服务失败的异常

### 5.1.2 异常编码规则

系统使用六位数字作为异常编码，格式为：XYZZZZ

- X：异常大类（1-业务异常，2-系统异常，3-参数异常，4-认证授权异常，5-外部服务异常）

- YY：模块编号（01-用户服务，02-实验服务，03-讨论服务，04-评估服务，05-支持服务）
- ZZZ：具体错误码（001-999）

例如：

- 100001：用户名已存在（业务异常-用户服务）
- 200001：数据库连接失败（系统异常-用户服务）
- 302001：实验参数格式错误（参数异常-实验服务）

### 5.1.3 异常信息国际化

系统支持异常信息的国际化，通过以下方式实现：

1. 定义国际化资源文件（中文、英文等）
2. 异常消息使用占位符设计，支持动态参数替换
3. 根据用户语言偏好返回对应语言的异常信息

## 5.2 故障预防与补救

### 5.2.1 故障预防策略

1. **入参校验**：使用Bean Validation框架对请求参数进行校验，防止非法参数
2. **幂等设计**：关键操作设计为幂等，防止重复提交导致的数据异常
3. **限流措施**：使用令牌桶或漏桶算法对API进行限流，防止过载
4. **熔断机制**：微服务调用使用熔断器模式，防止级联失败
5. **降级策略**：定义服务降级策略，在高负载或部分功能不可用时提供基本服务

### 5.2.2 故障恢复机制

1. **事务管理**：使用Spring事务管理确保数据一致性
2. **补偿事务**：分布式事务使用补偿模式，确保最终一致性
3. **重试机制**：对于暂时性故障，采用退避策略进行重试
4. **数据备份**：定期备份关键数据，支持数据恢复
5. **日志记录**：记录详细的操作日志，便于故障分析和恢复

## 5.3 系统维护设计

### 5.3.1 监控告警

1. **健康检查**：提供健康检查端点，监控服务状态
2. **性能指标**：收集关键性能指标（响应时间、吞吐量、错误率等）

3. **日志集中**：使用ELK Stack集中收集和分析日志
4. **告警机制**：设置阈值触发告警，通过邮件、短信等方式通知管理员

### 5.3.2 运维支持

1. **配置中心**：使用配置中心管理各环境配置，支持动态更新
2. **服务注册发现**：服务自动注册和发现，简化部署
3. **自动化部署**：使用CI/CD流程实现自动化构建和部署
4. **灰度发布**：支持灰度发布策略，降低版本更新风险

## 6 性能优化和安全设计

本系统在设计和实现过程中重视性能优化和安全防护，通过多层次的优化和防护措施，确保系统的高性能、高可用和安全可靠。

### 6.1 性能优化设计

#### 6.1.1 数据库优化

1. **索引优化**：
  - 对常用查询字段建立适当索引
  - 使用复合索引优化多字段查询
  - 定期分析索引使用情况并调整
2. **SQL优化**：
  - 避免全表扫描和复杂连接
  - 使用批量操作替代单条操作
  - 合理使用存储过程减少网络交互
3. **连接池配置**：
  - 根据系统负载配置适当的连接池大小
  - 设置合理的连接超时和最大等待时间
  - 配置连接检测和自动恢复机制

#### 6.1.2 缓存策略

1. **多级缓存**：
  - 本地缓存：使用Caffeine实现进程内缓存
  - 分布式缓存：使用Redis实现跨服务缓存
  - 内容缓存：对静态资源和不常变化的数据进行缓存
2. **缓存设计原则**：

- 热点数据优先缓存
- 合理设置缓存过期时间
- 使用缓存预热减少冷启动问题
- 实现缓存降级策略，防止缓存雪崩

### 6.1.3 JVM调优

#### 1. 内存管理：

- 根据负载调整堆内存大小
- 调整年轻代和老年代比例
- 选择适合业务特点的垃圾收集器

#### 2. 线程池优化：

- 根据CPU核心数和业务特点配置线程池
- 区分IO密集型和CPU密集型任务，使用不同的线程池
- 设置合理的任务队列和拒绝策略

### 6.1.4 分布式优化

#### 1. 负载均衡：

- 使用Nginx进行反向代理和负载均衡
- 支持基于权重、最少连接等多种负载均衡策略
- 实现健康检查和故障节点剔除

#### 2. 异步处理：

- 非关键路径使用异步处理
- 使用消息队列解耦服务
- 批量处理提高吞吐量

#### 3. 数据分片：

- 大表水平分片
- 使用一致性哈希算法减少再平衡影响
- 实现跨分片查询和事务支持

## 6.2 安全设计

### 6.2.1 认证与授权

#### 1. 认证机制：

- 基于JWT的令牌认证
- 支持OAuth2.0和OpenID Connect
- 多因素认证选项

#### 2. 权限控制：

- 基于RBAC（基于角色的访问控制）模型
- 精细粒度的API权限控制
- 动态权限分配和管理

### 3. 会话管理：

- 安全的会话创建和维护
- 会话超时和自动登出
- 会话劫持防护

## 6.2.2 数据安全

### 1. 数据传输安全：

- 全站HTTPS加密
- 敏感数据传输额外加密
- API签名机制防止篡改

### 2. 数据存储安全：

- 敏感信息加密存储（如密码使用BCrypt加密）
- 数据库加密
- 数据脱敏处理

### 3. 数据访问控制：

- 数据访问权限控制
- 操作审计日志
- 数据泄露防护

## 6.2.3 安全防护

### 1. 输入验证：

- 前后端双重验证
- 防止SQL注入、XSS攻击
- 防止CSRF攻击

### 2. 接口安全：

- API访问频率限制
- 防止暴力破解
- 异常访问检测和阻断

### 3. 安全监控：

- 安全日志记录与分析
- 异常行为检测
- 实时安全告警

# 7 项目测试计划

本系统采用多层次测试策略，确保软件质量和功能符合预期。测试过程涵盖单元测试、集成测试、系统测试和性能测试等多个阶段，按照"从小到大、从局部到整体"的原则进行。

## 7.1 测试策略

### 7.1.1 测试方法论

系统采用以下测试方法：

- 1. **TDD (测试驱动开发)**：核心业务逻辑采用测试驱动开发
- 2. **自动化测试**：尽可能多地采用自动化测试，减少人工测试的工作量
- 3. **持续集成测试**：在CI/CD流程中集成测试，实现开发、测试和部署的自动化
- 4. **分层测试**：包括单元测试、集成测试、系统测试和性能测试

### 7.1.2 测试环境

- 1. **开发环境**：供开发人员进行单元测试和集成测试
- 2. **测试环境**：独立的测试环境，模拟生产环境配置
- 3. **预生产环境**：与生产环境配置一致，用于系统测试和性能测试
- 4. **生产环境**：最终部署环境，用于验收测试和监控

## 7.2 测试计划详情

序号	测试类型	对应模块	测试内容	计划测试时间
1	单元测试	Common	工具类、异常处理、数据验证等基础功能	2025-05-15 ~ 2025-05-18
2	单元测试	Core	核心领域模型、业务规则、接口定义	2025-05-15 ~ 2025-05-20
3	单元测试	User-Service	用户注册、登录、授权等功能	2025-05-18 ~ 2025-05-22
4	单元测试	Experiment-Service	实验创建、执行、结果分析等功能	2025-05-20 ~ 2025-05-25
5	单元测试	Discussion-Service	讨论功能、消息通知等功能	2025-05-22 ~ 2025-05-26

序号	测试类型	对应模块	测试内容	计划测试时间
6	单元测试	Evaluation-Service	评价收集、数据分析等功能	2025-05-23 ~ 2025-05-27
7	单元测试	Support-Service	文件管理、系统支持等功能	2025-05-24 ~ 2025-05-28
8	集成测试	User + Core	用户模块与核心模块的集成	2025-05-26 ~ 2025-05-30
9	集成测试	Experiment + Core	实验模块与核心模块的集成	2025-05-27 ~ 2025-05-31
10	集成测试	Discussion + User	讨论模块与用户模块的集成	2025-05-28 ~ 2025-06-01
11	集成测试	Evaluation + Experiment	评价模块与实验模块的集成	2025-05-29 ~ 2025-06-02
12	集成测试	Support + 其他模块	支持模块与其他模块的集成	2025-05-30 ~ 2025-06-03
13	系统测试	全系统	功能测试、业务流程测试	2025-06-03 ~ 2025-06-10
14	性能测试	全系统	负载测试、压力测试、稳定性测试	2025-06-08 ~ 2025-06-15
15	安全测试	全系统	渗透测试、授权测试、数据安全测试	2025-06-10 ~ 2025-06-17
16	验收测试	全系统	用户验收测试、场景测试	2025-06-15 ~ 2025-06-20

## 7.3 测试技术和工具

### 7.3.1 单元测试

- 1. **JUnit 5**：Java单元测试框架
- 2. **Mockito**：模拟对象框架
- 3. **AssertJ**：断言库

单元测试示例：



```

@Test
void testUserRegistration() {
    // 准备测试数据
    UserRegistrationDTO registrationDTO = new UserRegistrationDTO();
    registrationDTO.setUsername("testuser");
    registrationDTO.setPassword("Password123");
    registrationDTO.setEmail("test@example.com");

    // 模拟依赖
    when(userRepository.findByUsername(anyString())).thenReturn(Optional.empty());
    when(passwordEncoder.encode(anyString())).thenReturn("encodedPassword");

    // 执行测试
    UserDTO result = userService.registerUser(registrationDTO);

    // 验证结果
    assertThat(result).isNotNull();
    assertThat(result.getUsername()).isEqualTo("testuser");
    assertThat(result.getEmail()).isEqualTo("test@example.com");

    // 验证交互
    verify(userRepository).save(any(User.class));
}

```

## 7.3.2 集成测试

1. **Spring Boot Test**: Spring Boot应用程序测试
2. **Testcontainers**: 提供轻量级、一次性的测试数据库实例
3. **WireMock**: 模拟外部服务

集成测试示例：

```

@SpringBootTest
@AutoConfigureMockMvc
class UserControllerIntegrationTest {

    @Autowired
    private MockMvc mockMvc;

    @Autowired
    private ObjectMapper objectMapper;

    @Test
    void testUserRegistrationAPI() throws Exception {
        // 准备测试数据
        UserRegistrationDTO registrationDTO = new UserRegistrationDTO();
        registrationDTO.setUsername("integrationUser");
        registrationDTO.setPassword("Password123");
        registrationDTO.setEmail("integration@example.com");

        // 执行API测试
        mockMvc.perform(post("/api/users/register")
            .contentType(MediaType.APPLICATION_JSON)
            .content(objectMapper.writeValueAsString(registrationDTO)))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$.code").value(200))
            .andExpect(jsonPath("$.data.username").value("integrationUser"));
    }
}

```

### 7.3.3 性能测试

1. **JMeter**: 负载测试工具
2. **Gatling**: 高性能负载测试工具
3. **Prometheus + Grafana**: 性能指标收集和可视化

### 7.3.4 安全测试

1. **OWASP ZAP**: Web应用程序安全扫描器
2. **SonarQube**: 代码质量和安全性分析
3. **Dependency Check**: 第三方依赖安全检查

## 7.4 测试覆盖率目标

1. **单元测试覆盖率**：核心业务逻辑 > 85%，工具类 > 90%
2. **集成测试覆盖率**：关键API和接口 > 80%
3. **功能测试覆盖率**：核心功能 100%，非核心功能 > 90%

## 7.5 测试流程和规范

1. **测试编写规范**：测试方法命名遵循"方法名\_条件\_预期结果"格式
2. **测试自动化集成**：测试集成到CI/CD流程中，代码合并前必须通过测试
3. **测试报告生成**：自动生成测试覆盖率和测试结果报告
4. **缺陷跟踪流程**：测试发现的缺陷记录到缺陷跟踪系统，并分配到责任人