软件设计原则评估与实践应用

# 一、结合项目开发历程对模块进行设计原则评估

## 单一职责原则（SRP）

评估与问题分析：在项目初期，部分模块如用户控制器同时处理登录验证与简历上传等逻辑，导致方法过长、逻辑混乱。  
解决方案：通过将用户模块划分为AuthController和ResumeController两个子模块，各自负责身份验证与简历管理功能，提升模块内聚性与代码可维护性。

## 开闭原则（OCP）

评估与问题分析：初期系统在扩展筛选岗位的方式时需频繁修改原有查询逻辑，存在高风险。  
解决方案：采用策略模式，将筛选功能提取为接口`JobFilterStrategy`，不同筛选策略作为实现类分别维护，便于后续扩展。

## 里氏替换原则（LSP）

评估与问题分析：继承自User的Admin用户由于权限逻辑不同，覆盖父类方法不一致，造成替换时异常。  
解决方案：重新设计抽象类User，剥离通用接口行为为接口，确保所有子类遵循统一接口契约，不破坏替换性。

## 迪米特法则（LoD）

评估与问题分析：前端直接访问多个后端对象嵌套属性（如user.profile.company.name），导致模块间强耦合。  
解决方案：在后端封装接口只返回必要字段，如CompanyDTO，使前端仅与直接对象交互，降低模块依赖。

## 依赖倒转原则（DIP）

评估与问题分析：业务逻辑层直接依赖MySQL数据库的具体实现，耦合度高。  
解决方案：定义抽象接口（如UserRepository），使用Spring注入具体实现，方便测试与数据库迁移。

## 合成复用原则（CRP）

评估与问题分析：早期多个通知方式类（如EmailNotification, SMSNotification）均继承同一父类，灵活性差。  
解决方案：改为将不同通知方式组合到NotificationService中，通过注入实现功能复用与动态控制。

# 二、六大设计原则定义与实践应用

## 1. 里氏替换原则（Liskov Substitution Principle）

定义：子类对象能够替代父类对象，且程序功能保持正确。  
应用：在本系统中，User类被AdminUser、EnterpriseUser继承，所有逻辑处理均调用User类型接口，确保替换时不影响系统行为。

## 2. 单一职责原则（Single Responsibility Principle）

定义：每个类只负责一项职责，变更原因应唯一。  
应用：ResumeService专注于简历相关处理，AuthService负责用户登录验证，职责清晰。

## 3. 开闭原则（Open/Closed Principle）

定义：软件应对扩展开放，对修改关闭。  
应用：新增岗位筛选条件时通过新增实现类实现FilterStrategy接口，无需修改原始JobService逻辑。

## 4. 迪米特法则（Law of Demeter）

定义：对象应避免与不直接相关的对象耦合，只与直接朋友通信。  
应用：控制器通过Service统一调用DAO层接口，不直接访问数据库对象属性。

## 5. 依赖倒转原则（Dependency Inversion Principle）

定义：高层模块依赖于抽象，不依赖具体实现。  
应用：业务层依赖UserRepository接口，Spring自动注入MySQL/Mock实现，便于扩展与测试。

## 6. 合成复用原则（Composite Reuse Principle）

定义：优先使用组合而非继承以实现复用。  
应用：UserProfileService通过注入多个功能服务组合实现复杂业务逻辑，提升模块重用性与可测性。