计算机学院 软件工程 课程实验报告

|  |  |  |
| --- | --- | --- |
| 实验题目：实验内容六 | | 学号：202100130028 |
| 日期：2024/5/20 | 班级： 计科2班 | 姓名：薛然然 |
| Email：1745047373@qq.com | | |
| 实验目的：  1. 培养设计原则实践的能力  2. 学习依赖注入（dependency injection） | | |
| 实验内容：   1. 参考教材6.2，结合项目的进程和开发历程，从设计原则的几个方面，组员对负责设计的模块进行评估，思考存在的问题和解决方案。  |  |  |  | | --- | --- | --- | | 设计原则 | 存在的问题 | 解决方案 | | 模块化 | 在故障定位和系统修复方面，暂未使每个模块具有唯一定位功能，也就是说，各模块之间的独立性还未完善。 | 可以在每个模块设计不同的错误代码，便于错误返回时及时找到相关的模块。 | | 接口 | 在个人用户模块中，可能存在接口定义不清晰的情况，导致不同子模块之间的交互困难。 | 明确定义每个子模块的接口，包括输入和输出，以确保模块之间的交互清晰可靠。使用文档或标准化的规范来描述接口，以便开发人员正确地使用和实现接口。 | | 信息隐藏 | 在企业后台管理模块中，可能存在信息暴露的风险，即企业敏感信息可能被未经授权的人员访问。 | 使用访问控制列表（ACL）或权限管理系统来限制对敏感信息的访问。对敏感信息进行加密存储，只有经过授权的用户可以解密访问。 | | 增量式开发 | 在系统后台管理模块中，可能存在一次性开发所有功能的风险，导致开发周期过长和风险高。 | 采用敏捷开发方法，将系统功能拆分为小任务，通过迭代逐步完善系统。每个迭代周期内，优先完成最核心和关键的功能，以降低项目失败的风险。 | | 抽象 | 在个人用户模块中，可能存在功能重复实现的情况，如个人信息和简历管理功能可能在不同子模块中重复实现。 | 将通用功能抽象为独立的组件或服务，以便多个子模块共享和复用。使用面向对象的设计原则，如继承和多态，来实现功能的抽象和扩展。 | | 通用性 | 在企业后台管理模块中，可能存在功能过于特定化的情况，导致无法满足其他行业或企业的需求。 | 设计灵活和可配置的模块，以便根据不同用户的需求进行定制和扩展。提供可插拔的插件架构，允许用户根据需要添加新功能或定制现有功能。 |  1. 阅读下面DI资料（或查阅其它相关资料），学习依赖注入技术。   Dependency injection - Wikipedia  Dependency Injection-A Practical Introduction.pdf  ——依赖注入(Dependency Injection, DI)是一种设计模式，也是Spring框架的核心概念之一。其作用是去除Java类之间的依赖关系，实现松耦合，以便于开发测试。  耦合太紧的问题：比如创建了一个类的实例如下：  A(){  this.B=new C();  }  这种情况下，A的B只能是C，如果想替换为D，所有涉及到的代码都要修改。那么依赖注入就是A要依赖C，但A不再直接创建C，而是把这种依赖关系配置在外部xml文件（或java config文件）中，然后由Spring容器根据配置信息创建、管理bean类。  传入哪个子类，可以在外部xml文件（或者java config文件）中配置，Spring容器根据配置信息创建所需子类实例，并注入Player类中，如下所示：  <bean id="A" class="com.qikegu.demo.A">  <construct-arg ref="B"/>  </bean>  <bean id="B" class="com.qikegu.demo.C">  </bean>  上面代码中<construct-arg ref="B"/> ref指向id="B"的bean，传入的类型是C，如果想改为D，可以作如下修改：  <bean id="weapon" class="com.qikegu.demo.D">  </bean>  只需修改这一处配置就可以。松耦合，并不是不要耦合。A类依赖B类，A类和B类之间存在紧密耦合，如果把依赖关系变为A类依赖B的父类B0类，在A类与B0类的依赖关系下，A类可使用B0类的任意子类，A类与B0类的子类之间的依赖关系是松耦合的。 | | |
| 结论分析与体会：  1.模块评估从设计原则出发，思考可能出现的问题及解决内容，这样可以有效规避软件开发设计中可能出现的问题，并对项目有个更好的把握。  2.学习依赖注入技术  ——依赖注入（DI）是一种设计模式，通过外部配置来管理类之间的依赖关系，从而实现松耦合的设计，例如在Spring框架中广泛应用。通过DI，可以动态地注入依赖对象，而不是在类内部直接创建依赖对象。这种模式使得代码更易于维护和测试，因为依赖可以在外部配置文件中进行管理和修改，而不需要修改类本身的代码。  通过DI，可以将依赖关系的管理交由容器（如Spring容器）来处理，从而提高代码的灵活性和可维护性。例如，可以通过配置文件指定依赖对象的具体实现类，使得系统在不同情况下能够轻松切换依赖实现，而不需要修改大量代码。  DI不是完全消除耦合，而是通过将依赖关系的管理移到外部来实现松耦合，这样类之间的依赖关系更清晰和可控。 | | |