实验十二 设计模块（一 ）

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。**

## 接口

设计原则是指把系统功能和行为分解成模块的指导方针，依据Davis开发的软件开发原则，软件开发的设计原则从六个领域展开：模块化，接口，信息隐藏，增量式开发，抽象以及与通用性，这里我们讨论模块化，接口，以及增量式开发，根据这些原则对我们的模块进行评估，并加以融合帮助我们设计出更加高效，健壮的设计。

每个软件单元和其他单元分开，用接口给其他单元提供在这个单元内定义的一些服务，以及获取这些服务的方法。软件单元的接口需要规格说明，对软件单元做出描述包括单元的访问的函数，参数，返回值以及异常的生命，接口的规格说明要向项目开发人员传递准确的项目软件单元的信息，帮助开发人员进行进一步开发。此外在接口部分还要写清楚每个可访问的函数的详细功能的文档，前置条件，后置条件等等。

在本项目中关于接口设计存在一些问题如下：

首先系统中的各个模块之间存在紧耦合的现象，也就是对于两个不同的模块，我们修改了其中一个模块中的内容之后，另外一个和他有关系的模块也可能收到影响，尤其是对我们的项目模块和模块之间的关系比较紧密，而且耦合关系较多，这种紧耦合问题的出现对我们作为开发者来说增加了系统的复杂性以及维护的成本。在看书之后，解决这个接口之间紧耦合问题的方法是采用接口隔离原则，将系统中的不同功能模块进行解耦，每个模块通过接口进行通信，降低模块之间的依赖性，提高系统的灵活性和可维护性。

其次，本项目中各个单元之间存在的接口定义不够清晰，的确出现了开发人员之间难以理解接口的用途和范围的问题，对最开始的开发以及集成增加了不少难度。对此考虑采用明确的接口设计，小组成员将对此进一步补充，定义更清晰的接口规范和文档，包括几口的输入输出，调用方式，异常处理，每个模块定义的详细功能文件等等。

也出现了部分接口设计过于臃肿的现象，最初设计的时候先设计各个模块的各个功能，但是部分功能和参数在最终集成的时候并没有使用到，这些参数不仅让所属的部分的接口更加臃肿，也导致项目的接口不易被理解，对此，我们会考虑采用接口分离的方法，也就是将接口拆分为多个小而精确的接口，每个接口只关注单一功能，提高了接口的可复用性和灵活性。

## 增量式开发

通过增量式开发，我们可以在确保系统基础功能稳定的前提下，逐步引入新功能和改进，降低项目风险，提升系统质量和用户满意度。

从满足增量式开发的角度考虑，使用图也可以帮助我们确定可改善的设计范围。设计良好的使用图应具有树型结构或者是树型结构的森林。在这样的结构中，每棵子树都是系统的一部分，所以我们可以一次一个软件单元地增量开发我们的系统，每个完成的单元都是我们系统的部分实现。在开发过程中，每一次的增量都会越来越易于测试和修改，因为错误只可能出现在新代码中，而不是在已经过测试和验证的被调用单元中。此外，我们总有一个可运行的系统版本用来展示给客户。更多地，系统频繁且可见的进展也鼓舞了开发人员的士气(Brooks1995)。和其他方法相比，增量式开发有着不可多得的优势，因为前者只有当每个单元都能工作时系统才能工作。

各模块评估：

用户管理模块：

功能: 注册、登录、用户信息管理、权限管理。

评估: 初期可实现基本的注册和登录功能，逐步增加用户信息管理和权限管理功能。

存在的问题: 初期版本可能安全性较弱，权限管理功能不完善。

解决方案: 增量开发中，首先实现基础功能，再逐步增加密码加密、验证机制和角色权限管理功能，确保每个增量都经过充分测试。

书籍管理模块：

功能: 书籍信息添加、编辑、删除、分类管理。

评估: 先实现基本的书籍添加和查看功能，逐步增加编辑、删除和分类管理功能。

存在的问题: 数据一致性和并发操作处理不足。

解决方案: 初期使用简单的CRUD操作，逐步引入事务管理和锁机制，确保数据一致性和并发操作的安全性。

购物车模块：

功能: 添加书籍到购物车、查看购物车、删除购物车中的书籍、结算。

评估: 初期可实现书籍添加和查看功能，逐步增加删除和结算功能。

存在的问题: 数据同步和状态管理可能存在问题。

解决方案: 使用状态管理库（如Redux）来管理购物车状态，确保不同页面的同步。逐步引入用户会话管理，防止数据丢失。

订单模块：

功能: 创建订单、查看订单、取消订单、支付订单。

评估: 初期可实现订单创建和查看功能，逐步增加取消和支付功能。

存在的问题: 数据同步和状态管理可能存在问题，特别是在多用户同时操作订单时。

解决方案:

使用状态管理库（如Redux）来管理订单状态，以确保不同页面和组件之间订单数据的同步和一致性。引入合适的权限管理机制，确保用户只能对自己的订单进行操作，防止数据混乱和安全问题。

## 模块化

模块化原则在系统中能够起到是各部分实现独立研究的作用。它把系统中各不相关的部分进行分离，把他们放置于各自的模块中。实现模块化，能够让每个模块的理解和开发更加简单，让故障的定位和系统的修改也更加方便。

耦合度和内聚度是用来度量模块的独立程度的两个概念。当两个模块间有大量依赖关系时，这两个模块是紧密耦合的。当两个模块间相互连接比较弱时，这两个模块是松散耦合的。非耦合是指模块间没有任何相互连接，完全独立。内聚度是指模块内部各个组成部分之间的紧密程度。高内聚度意味着模块内部功能单一，所有组成部分都为实现该功能而紧密协作。模块化设计原则旨在通过降低耦合度和提高内聚度，增强系统的可维护性、可扩展性和可重用性。在网上书店系统中，应用这些原则可以显著改善系统的结构和性能。

降低耦合度可以通过定义清晰的接口，使模块间通过接口通信，而不是直接依赖具体实现；还可以利用事件驱动或消息传递机制，减少模块之间的直接调用。为了提高内聚度可以提高让单个模块的功能单一化，避免多职责混合；可以提高模块内部的逻辑，使其紧密相关，所有操作围绕模块的主要功能实现；

在我们的网上书店系统中，模块化设计原则起着至关重要的作用。模块化设计的关键在于将系统分解为多个功能独立、相对独立的模块，每个模块完成特定的功能，从而提高系统的可维护性、可扩展性和可重用性。以下是网上书店的各个模块及其可能存在的问题和解决方案：

1. 用户管理模块

功能：注册、登录、用户信息管理、用户权限管理

存在问题：

- 用户数据安全性问题：用户的敏感信息如密码可能被泄露。

- 模块间依赖过重：用户管理模块可能与其他模块耦合过紧，导致难以独立修改和测试。

解决方案：

- 使用加密技术如哈希函数来存储用户密码，确保用户数据安全。

- 采用微服务架构，将用户管理模块独立出来，通过API与其他模块通信，降低耦合度。

2. 图书管理模块

功能：图书信息管理、库存管理、分类管理

存在问题：

- 数据一致性问题：多个用户同时操作库存时，可能会出现数据不一致的情况。

- 复杂的分类和检索：随着图书种类的增加，分类和检索功能可能变得复杂且性能下降。

解决方案：

- 使用事务管理和乐观锁机制来保证数据一致性。

- 引入缓存机制来优化检索性能。

1. 订单管理模块

功能：订单创建、订单支付、订单跟踪、订单取消

存在问题：

- 支付过程中的安全问题：支付信息传输过程中可能被窃取。

- 订单状态管理复杂：订单状态的管理和更新容易出错，导致订单处理混乱。

解决方案：

- 使用安全的支付网关和HTTPS协议来保护支付信息。

- 采用状态机设计模式来管理订单状态，确保状态转换的正确性和可维护性。

4. 购物车模块

功能：商品加入购物车、购物车查看、购物车结算

存在问题：

- 购物车数据同步问题：用户在不同设备上操作购物车时，数据可能不同步。

- 购物车数据量大时性能问题：大量商品加入购物车后，操作性能可能下降。

解决方案：

- 将购物车数据存储在服务器端，并通过用户会话进行同步，确保多设备同步。

- 使用分布式缓存系统（如Redis）来提高购物车操作的性能。

5. 评论和评分模块

功能：用户评论、用户评分、评论管理

存在问题：

- 评论审核难度大：需要防止恶意评论和垃圾评论。

- 评分系统的公平性：评分系统可能被刷分或恶意评分。

解决方案：

- 引入机器学习算法或人工审核机制来识别和过滤恶意评论和垃圾评论。

- 设计合理的评分算法，结合用户的购买行为和历史评分，提高评分系统的公平性。

6. 搜索和推荐模块

功能：图书搜索、个性化推荐、热门推荐

存在问题：

- 搜索功能的准确性：搜索结果可能不准确或不相关。

- 推荐系统的准确性：推荐的图书可能不符合用户兴趣，影响用户体验。

解决方案：

- 使用自然语言处理（NLP）技术和搜索引擎优化（SEO）来提高搜索结果的准确性。

- 采用基于协同过滤和内容的推荐算法，结合用户行为数据进行个性化推荐，提高推荐的相关性和准确性。

总结

模块化设计原则的应用可以显著提高网上书店系统的可维护性、可扩展性和可重用性。通过独立设计和实现各个功能模块，并通过接口进行通信，可以降低模块间的耦合度，增强系统的灵活性和可扩展性。同时，针对每个模块存在的问题，采用合适的解决方案可以提高系统的整体性能和用户体验。

**2. 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**](https://en.wikipedia.org/wiki/Dependency_injection#:~:text=In%20software%20engineering%2C%20dependency%20injection,object%20is%20called%20a%20service.)

**Dependency Injection-A Practical Introduction.pdf**

依赖注入，也就是 DI，英文 dependency injection 的缩写，是一系列用于开发解耦合代码的软件设计原则和模式。这个定义不仅解释了什么是依赖注入：软件设计原则和模式。同时也告诉了我们依赖注入的作用：开发解耦合的代码。

所以，依赖注入是设计模式范畴的技术，和其他的设计模式一样，主要是用于降低应用程序代码的耦合度。在依赖注入中，一个组件依赖另一个组件时，不再直接创建这个依赖的组件实例，而是通过外部方式（如构造函数注入、属性注入）获得这个依赖的组件实例。

依赖注入的主要作用包括：

1. 提高可维护性：降低组件间的耦合，使代码更易维护。当程序出错的时候，我们能够更加容易发现我们需要修改哪部分内容。

2. 提高可扩展性：可以轻松替换依赖的组件，实现功能的扩展。

3. 提高可测试性：通过注入测试替身（如Mock对象），可以方便地测试组件。

4. 实现延迟绑定：可以在运行时动态选择依赖的组件，而无需重新编译。

5. 解耦合的代码更加易于扩展。我们能够在不改变大量对象的情况下增加功能。

6. 能够将功能独立开来，以便编写简短的，易于阅读的单元测试。

7. 团队协作开发的过程中，比如提交合并代码，通常不希望也应该避免团队成员之间的代码存在冲突，而解耦合有利于团队成员各自维护自己的代码片段而互相不受影响。

8. 解耦合可以使延迟绑定变得更加容易。延迟绑定，或者运行时绑定，是在运行时做决定而不是编译时，这在特定场合下很有用。

在网上书店项目中，依赖注入技术应用于多个方面：

1. 用户管理：注入用户服务接口，实现用户注册、登录等功能。

2. 图书浏览：注入图书服务接口，实现不同用户的图书浏览或选购。

3. 店主管理：注入进书服务、售书服务、库存服务、账目服务等接口，实现店主管理功能。

4. 网站维护：注入网站维护服务接口，实现网站内容的日常维护。

5. 客户管理：注入客户服务接口，实现对客户的管理。

6. 购物流程：注入订单服务接口，实现会员购书流程。

通过依赖注入技术，网上书店项目可以实现不同功能模块之间的解耦，提高项目的可维护性、可扩展性和可测试性。同时，通过DI容器框架，如Unity或Ninject，可以进一步简化DI的使用，提高开发效率。

Dependency Injection-A Practical Introduction.pdf内容概述：

DI概述：DI是一种设计模式，它允许在运行时而不是编译时选择要使用的组件。其目标是实现松耦合的代码。

使用DI的好处：DI可以提高代码的可扩展性、可维护性、可测试性，并实现延迟绑定。具体来说，DI可以轻松添加新功能，简化多个团队并行开发，提高单元测试的可操作性，并在运行时选择不同的组件而不需要重新编译代码。

DI模式：DI有多种模式，包括构造函数注入、属性注入、方法注入、环境注入、服务定位器等。文档详细介绍了构造函数注入和属性注入两种主要模式。

示例代码分析：文档通过一个简单的示例代码，展示了DI如何解决代码耦合问题。首先，添加一个仓库接口IPersonRepository，然后用DI将仓库注入到ViewModel中。最后，使用DI将ViewModel注入到View中，这样代码的耦合关系被打破了。

DI容器：文档介绍了Unity和Ninject两种DI容器框架的使用方法。容器可以自动管理依赖关系，简化了DI的使用。通过在容器中注册类型，可以轻松获取注入后的对象。

XML配置：文档介绍了如何使用XML配置实现延迟绑定。通过XML配置文件，可以在运行时动态加载不同的组件，而无需重新编译代码。然而，XML配置也存在脆弱性的缺点。

单元测试：DI可以帮助进行单元测试，因为它可以注入测试替身（如mock对象），从而隔离要测试的代码。