

**软件体系结构作业18**

姓 名 ： 洪伟鑫

专 业 ： 软件工程

年 级 ： 2022级

学 号 ： 37220222203612

**2025年5月7日**

**1、阅读：Java SE Application Design With MVC**

[**https://www.oracle.com/technical-resources/articles/javase/application-design-with-mvc.html**](https://www.oracle.com/technical-resources/articles/javase/application-design-with-mvc.html)

通过阅读了 《Java SE Application Design With MVC》这篇文章。我大概理解了 MVC 模式的核心思想，以及它为何对于构建结构清晰、易于维护的 Java 应用至关重要。

文章首先介绍了 MVC 设计模式的三个核心组件。**模型（Model）** 被描绘为应用的核心，是数据和业务规则的“家”。它负责管理应用的数据及相关的业务处理逻辑，就像是应用的“大脑”，储存着所有重要的信息和状态，并且其运作独立于用户最终看到什么或如何操作。一个关键的特性是，当模型内部的数据发生变化时，它会主动通知那些关注这些变化的部分（通常是视图），以便它们能够相应地更新显示内容。重要的是，模型本身并不知道视图的具体呈现形式，也不了解控制器是如何工作的。例如，在文章提到的通讯录应用中，模型就包含了联系人的详细信息，以及如何增加、删除、修改这些信息和如何将这些信息持久化存储的逻辑。

接着是**视图（View）**，这部分是用户能直接看到并与之交互的界面。视图的主要工作是从模型获取数据，然后以一种直观易懂的方式呈现给用户。视图本身通常不包含复杂的应用逻辑，它更像一个纯粹的“展示者”。值得注意的是，同一个模型的数据可以有多种不同的视图来展示，比如可以是一个列表、一组卡片或者一个详细的信息页面。当用户在视图上进行操作时，这些操作通常会传递给控制器进行后续处理。在通讯录的例子中，我们看到的联系人列表、用于添加或编辑联系人的表单界面，这些都属于视图的范畴。

最后是**控制器（Controller）**，它在模型和视图之间扮演着“交通警察”的角色，负责协调两者的交互。控制器会接收来自用户的输入（通常通过视图传递过来，如点击按钮、提交表单等），解析这些用户操作的意图，并据此决定是去更新模型的数据，还是通知视图改变其显示方式。可以说，控制器是连接模型和视图的桥梁，它确保了模型和视图能够各司其职，同时又避免了它们之间产生不必要的紧密耦合。在通讯录的场景下，如果用户点击了“保存联系人”按钮，控制器就会接收到这个操作信号，从视图的输入字段中获取用户填写的信息，然后指示模型去保存或更新相应的联系人数据。操作完成后，控制器可能还会通知视图刷新，以便用户能够看到最新的数据状态。

那么，为什么要采用 MVC 模式呢？文章总结了几点核心优势。首先是**关注点分离**，即“各管各的，不乱套”，每个组件都有自己明确的职责，这使得代码逻辑更加清晰，系统整体的复杂度也随之降低。其次，它能**提高可维护性**，“改东西方便”，例如，如果你想调整用户界面（视图），通常情况下不会影响到核心的业务逻辑（模型）。再次，MVC 模式能**增强可重用性**，“能重复用”，同一个模型可以被多个不同的视图所使用，控制器的某些逻辑也可能在不同场景下得到复用。此外，由于各组件相对独立，**可测试性也得到提升**，“好测试”，可以更方便地对模型、视图和控制器进行独立的单元测试，从而更容易发现和修复潜在问题。最后，它还**支持并行开发**，“大家可以一起干活”，不同的开发人员或团队可以同时专注于不同组件的开发，从而提高整体的开发效率。

文章还特别强调了几个在实现 MVC 模式时的关键点。其中之一是有效利用**观察者模式**，即“模型一变，视图就得知道”。模型经常会采用这种机制，使得当其内部数据发生更新时，所有相关的视图都能自动收到通知并进行刷新，而模型本身则无需关心视图的具体实现细节。同时，**职责分明**也是非常重要的一环：模型专注于数据和业务逻辑，视图专注于展示，而控制器则专注于协调。最终的目标是实现各组件间的“松耦合”，让它们之间的依赖尽可能少，这样构建出来的系统才会更加灵活，也更容易在未来进行扩展和迭代。

总的来说，通过阅读这篇文章，我对 MVC 模式在 Java SE 应用中的设计理念、各部分的角色分工、它们之间如何互动协作，以及这种模式能为软件开发带来的诸多实际好处，都有了更为深入和全面的理解。

**2、 LoD原则强调“只和朋友通信，不和陌生人说话”。请举例说明 “朋友圈” 认定依据是啥？**

迪米特法则，也常被称为“最少知识原则”，其核心思想是：**一个对象应该对其他对象有尽可能少的了解。** 通俗地讲，就是“只和你的直接朋友通信，不要和陌生人说话。”

那么，如何界定一个对象的“朋友圈”呢？或者说，在一个对象 O 的方法 M 内部，它可以合法调用哪些对象的方法？根据迪米特法则，对象 O 的方法 M 可以调用的“朋友”对象包括以下几类：

1. **对象 O 自身 (this)：**

**依据：** 对象当然可以调用自己的方法或访问自己的成员。

**示例：** 在一个 Order 类中，calculateTotalPrice() 方法可以调用同一个 Order 实例的 getLineItems() 方法。

1. **方法 M 的参数对象：**

**依据：** 这些对象是作为明确的输入传递给方法 M 的，方法 M 需要直接与它们协作来完成其职责。

**示例：** Order 类有一个方法 addItem(Product product, int quantity)。在这个方法内部，Order 对象可以通过参数 product 调用 Product 类的方法，比如 product.getPrice()。因为 product 是作为参数直接传递进来的。

1. **在方法 M 内部创建/实例化的对象：**

**依据：** 这些对象是由方法 M 局部创建和管理的，方法 M 对它们有完全的控制权。

**示例：** 在 Order 类的 generateInvoice() 方法中，如果创建了一个 InvoiceGenerator invGen = new InvoiceGenerator();，那么 Order 对象可以通过 invGen 调用 InvoiceGenerator 的方法，比如 invGen.createPDF(this)。

1. **对象 O 的直接成员变量/组件对象（实例变量或类变量）：**

**依据：** 这些是对象 O 在结构上直接拥有的、作为其一部分的对象。对象 O 依赖这些组件来构成其整体。

**示例：** Order 类可能有一个成员变量 Customer customer;。那么 Order 对象的方法可以调用 customer 对象的方法，比如 customer.getShippingAddress()。

**以一个网上书店的例子说明如下：**

**包含如下几个类**

** Customer (顾客)**

** Book (书籍)**

** ShoppingCart (购物车 - 顾客的直接朋友)**

** PaymentGateway (支付网关 - 购物车处理支付时的朋友)**

** InventorySystem (库存系统 - 书籍信息的朋友，或者支付成功后更新库存的朋友)**









**在Demo中“朋友圈”认定依据分析如下：**

**Customer 的朋友圈有：**

this (Customer 自身)

cart (成员变量，是 Customer 的直接朋友)

book (作为 addItemToCart 方法的参数)

inventory (作为 placeOrder 方法的参数)

**不和陌生人说话：** Customer 不应该直接去获取 cart.getPaymentGateway() 然后调用支付网关的方法。PaymentGateway 是 ShoppingCart 的朋友，但不是 Customer 的直接朋友。Customer 只需要告诉 ShoppingCart “我要结账”，具体怎么支付是 ShoppingCart 的职责。

**ShoppingCart 的朋友圈有：**

this (ShoppingCart 自身)

items (成员变量，是 ShoppingCart 的直接朋友)

paymentGateway (成员变量，是 ShoppingCart 的直接朋友)

book (作为 addBook 方法的参数)

inventory (作为 checkout 方法的参数)

item (在 calculateTotal 和 checkout 方法的循环中，是局部迭代变量，其类型 Book 是 ShoppingCart 知道的类型)

**不和陌生人说话：** ShoppingCart 不会去关心 Customer 的姓名，它只负责管理商品和结账流程。

**总结 LoD “朋友圈”的认定依据：**

一个对象的方法应该只调用以下几类对象的方法：

1. 该对象本身 (this)。
2. 作为方法参数传递进来的对象。
3. 在方法内部创建或实例化的对象。
4. 该对象的直接组件（成员变量）。

遵循迪米特法则有助于创建低耦合、高内聚的系统，提高模块的独立性和可维护性。当一个模块的内部实现发生变化时，只要其对外的“朋友”接口不变，就不太会影响到其他模块。