《Head First 设计模式》笔记

-- Dawei Min

1 模式

1.1 策略模式

策略模式：定义了算法簇，封别封装起来，让他们之间可以互相替换，此模式让算法的变化独立于使用算法的客户。

1.2 观察者（Observer）模式

观察者模式定义了对象之间的一对多依赖，这样一来，当一个对象改变状态时，它的所有依赖者都会收到通知并自动更新。

1.3 外观模式

外观模式（Facade）：提供了一个统一的接口，用来访问子系统同中的一群接口。外观定义了一个高层接口，让子系统更容易使用。

特点：

外观没有“封装”子系统，外观只是提供简化的接口。所以用户如果觉得有必要，依然可以使用子系统的类。

外观可以附加“聪明”的功能，让使用子系统更方便（这里的“聪明”的功能个人理解为：通过子系统也能实现的，但是外观把调用的逻辑等实现了）。

每个子系统可以创建多个外观。

外观模式允许你将客户从组件的子系统中解耦。如果客户只是使用外观，那么修改子系统的实现，只要外观不变即可。

与适配器模式的差异主要在意图上。适配器的意图是将接口转换成不同接口，而外观意图是简化接口。

1.4 装饰者模式

装饰者模式：动态地将责任附加到对象上。若要扩展功能，装饰者提供了比继承更有弹性的替代方案。

装饰者一般对组件的客户是透明的，除非客户程序依赖于组件的具体类型。

装饰者会导致设计中出现许多小对象，如果过度使用，会让程序变得很复杂。

经典案例：Java I/O

1.5 工厂模式

工厂方法模式：定义了一个创建对象的接口，但由子类决定要实例化的类是哪一个。工厂方法让类的实例化推迟到子类。

抽象工厂模式：提供一个接口，用于创建相关或依赖对象的家族，而不需要明确指定具体类。

创建对象的区别：工厂方法模式采用的方法是继承，抽象工厂模式通过对象组合。

1.6 单件模式

单件模式（Singleton）：确保一个类只有一个实例，并提供一个全局访问点。

1.7 代理模式

代理模式为另一个对象提供一个替身或占位符以控制对这个对象的访问。

使用代理模式创建代表（representative）对象，让代表对象控制某对象的访问，被代表的对象可以是远程对象、创建开销大的对象或需要安全控制的对象。

装饰者模式、适配器模式、代理模式的区别：

装饰者模式为对象增加行为，适配器会改变对象的接口，代理实现相同的接口控制对象的访问。

2 原则

2.1 针对接口编程

针对接口编程而不是针对实现编程。“针对接口编程”这句话，可以更明确地说成“变量的声明类型应该是超类型，通常是一个抽象类或者接口或者是一个接口”。针对接口编程时利用多态特性。

2.2 多用组合，少用继承

组合可以是系统具有很大的弹性，可以“在运行时动态得改变行为”。

2.3 松耦合设计

为了交互对象之间的松耦合设计而努力。松耦合的设计之所以能让我们建立有弹性的 OO 系统，能够应对变化，是因为**对象之间的互相依赖降到了最低**。

观察者模式提供了一种对象设计，让主题和观察者之间松耦合。

2.4 开放-关闭原则

类应该对扩展开放，对修改关闭。

我们的目标是运行类容易扩展，在不修改现有代码的情况下，就可以搭配新的行为。虽然似乎有点矛盾，但是确实有些技术可以实现。在选择需要被扩展的代码部分要小心。每个地方都采用开放-关闭原则是一种浪费，也没有必要，还会导致代码变得复杂且难以理解。

比如：装饰者模式、观察者模式。

2.5 依赖倒置原则

依赖倒置原则（Dependency Inversion）：要依赖抽象，不要依赖具体类。

这个原则说明了：不能让高层组件依赖低层组件，“两者”都应该依赖于抽象。

比如：工厂模式。

2.6 最小知识原则

最小知识原则（Least Knowledge）：当我们设计一个系统时，对任何对象都要注意它交互的类有哪些，不要让多个类耦合在一起，免得修改一部分会影响到其它部分。

反例：return　a.getB().getC();

正例：return a.getC();

如果我们调用从另一个调用中返回的对象的方法，相当于向另一个对象的子部分发请求，增加了我们直接认识的对象数目。

做法

在对象方法内，我们只应该调用属于以下范围的方法：

（1）该对象本身

（2）方法传递进来参数的对象

（3）此方法创建或实例化的任何对象（方法内的局部对象）、

（4）对象的任何组件（组件：HAS-A 关系）