**[《Head First 设计模式》读书笔记](https://www.cnblogs.com/GO-NO-1/p/7396768.html)**

**1、欢迎来到设计模式的世界：策略模式**

策略模式：定义算法族，分别封装起来，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。

OO原则：封装变化，多用组合，少用继承

针对接口编程，不针对实现编程

OO基础：抽象 封装 多态 继承

**2、让你的对象知悉现况：观察者模式**

观察者模式：在对象之间定义一对多的依赖，这样一来，当一个对象改变状态，依赖它的对象都会收到通知，并自动更新。

OO原则：为交互对象之间的松耦合设计而努力

**3、装饰对象：装饰者模式**

装饰者模式：动态地将责任附加到对象上。想扩展功能，装饰者提供有别于继承的另一种选择。

OO原则：对扩展开放，对修改关闭。

**4、烘烤OO的精华：工厂模式**

简单工厂模式:

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。工厂方法让类把实例化推迟到子类。

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。

OO原则：依赖抽象，不要依赖具体类。

**5、独一无二的对象：单件模式**

单件模式：确保一个类只有一个实例，并提供全局访问点。

适用场景：

1.需要生成唯一序列的环境

2.需要频繁实例化然后销毁的对象

3.创建对象时耗时过多或者耗资源过多，但又经常用到的对象。

4.方便资源相互通信的环境

优点：

1.实现了对唯一实例访问的可控

2.对于一些需要频繁创建和销毁的对象来说可以提高系统的性能。

缺点：

1. 不适用于变化频繁的对象

2.滥用单例将带来一些负面问题，如为了节省资源将数据库连接池对象设计为的单例类，可能会导致共享连接池对象的程序过多而出现连接池溢出。

3.如果实例化的对象长时间不被利用，系统会认为该对象是垃圾而被回收，这可能会导致对象状态的丢失。

**6、封装调用：命令模式**

命令模式：将请求封装成对象，这可以让你使用不同的请求、队列，或日志请求来参数化其他对象。命令模式也可以支持撤销操作。

四人组：将请求封装为一个对象，从而使你可用不同的请求对客户进行参数化;对请求排队或记录请求日志，以及支持可撤回的操作。

**7、随遇而安：适配器与外观模式（facade）**

适配器模式：将一个类的接口，转换成客户期望的另一个接口。适配器让原本不兼容的类可以合作无间。

外观模式：提供一个统一的接口，用来访问子系统中的一群接口。外观定义了一个高层接口，让子系统更容易使用。

OO原则：只和朋友交谈。

**8、封装算法：模板方法模式**

模板方法模式：在一个方法中定义一个算法的骨架，而将一些步骤延迟到子类中。模板方法使得子类可以在不改变算法结构的情况下，重新定义算法中的某些步骤。

OO原则：别找我，我会找你。

**9、管理良好的集合：迭代器与组合模式**

迭代器模式：提供一种方法顺序访问一个聚合对象中的各个元素，而不暴露其内部的表示。

组合模式：允许你将对象组成树形结构来表现“整体/部分”的层次结构。组合能让客户以一致的方式处理个别对象和对象组合。

OO原则：类应该只有一个改变的理由。

**10、事物的状态：状态模式**

状态模式：允许对象在内部状态改变时改变它的行为，对象看起来好像修改了它的类。

**11、控制对象访问：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访问这个对象。

代理：远程代理、虚拟代理、保护代理、防火墙代理、智能引用代理、缓存代理、同步代理、复杂隐藏代理、写入时复制代理。

**12、模式中的模式：复合模式**

复合模式：结合两个或以上的模式，组成一个解决方案，解决一再发生的一般性问题。

MVC：结合观察者模式、策略模式、组合模式的复合模式。

**13、模式中的模式：门面模式**

提供一个统一的接口去访问多个子系统的多个不同的接口，它为子系统中的一组接口提供一个统一的高层接口。使用子系统更容易使用。

**14、真实世界中的模式：与设计模式相处**

模式：在某情境下，针对某问题的某种解决方案。

“力”：塑造并控制解决方案。

模式分类：

创建型：涉及将对象实例化，提供一个方法将客户从所需要实例化的对象中解耦。

行为型：涉及到类和对象如何交互及分配职责。

结构型：把类或对象组合到更大的结构中。

反模式：告诉你如何采用一个不好的解决方案解决一个问题。

**15、剩下的模式**

桥接模式：不止改变你的实现，也改变你的抽象。适合使用在需要跨越多个平台的图形和窗口系统上。

生成器模式：封装一个产品的构造过程，并允许按步骤构造。适合创建组合结构。

责任链模式：让一个以上的对象有机会能够处理某个请求。适用于 窗口系统中，处理鼠标和键盘之类的事件。

蝇量模式：让某个类的一个势力能用来提供许多“虚拟实例”。适用于一个类的许多实例能被同一方法控制的情况。

解释器模式：为语言创建解析器。适用于创建一个简单的语言。

中介者模式：集中相关对间的复杂的沟通和控制方式。适用于协调相关的GUI组件。

备忘录模式：让对象返回之前的状态。适用于储存状态。

原型模式：创建给定类的实例的过程很昂贵或很复杂时。适用于处理复杂的类层次。

访问者模式：让一个对象的组合增加新的能力，且封装并不重要时。

1. **[模式PK](https://www.cnblogs.com/zhaoxinshanwei/p/8858844.html): [命令模式VS策略模式](https://www.cnblogs.com/zhaoxinshanwei/p/8858844.html)**

命令模式和策略模式的类图确实很相似，只是命令模式多了一个接收者（Receiver）角色。它们虽然同为行为类模式，但是两者的区别还是很明显的。策略模式的意图是封装算法，它认为“算法”已经是一个完整的、不可拆分的原子业务（注意这里是原子业务，而不是原子对象），即其意图是让这些算法独立，并且可以相互替换，让行为的变化独立于拥有行为的客户；而命令模式则是对动作的解耦，把一个动作的执行分为执行对象（接收者角色）、执行行为（命令角色），让两者相互独立而不相互影响。

# **17、[模式PK](https://www.cnblogs.com/zhaoxinshanwei/p/8858844.html): [代理模式VS策略模式](https://www.cnblogs.com/zhaoxinshanwei/p/8858844.html)**

简单代理模式中，代理类知道被代理类的行为，因为代理类与被代理类实现的是同一个接口，因此代理类与被代理类的结构是相同的；

而策略模式中，策略容器并不知道内部策略的详细信息，因为容器并没有实现与内部策略相同的接口，即容器与内部策略只是简单的组合关系，容器只是将内部策略的行为抽取出来，进行了统一的实现。

**文档**

# **[模式PK：命令模式VS策略模式](https://www.cnblogs.com/zhaoxinshanwei/p/8858844.html)**