**设计模式**

<https://www.cnblogs.com/maowang1991/archive/2013/04/15/3023236.html>

软件开发不变的真理：

CHANGE, 不论你自认为， 你设计的多么好，想的有多远， 但是经过时间的验证，你总是要更改你的代码，去做业务上的适应。你更改的过程中，会遇到很多问题， 而你遇到的问题是 有些人已经解决的了，那么就可以共享这些经验，建立弹性的设计，易于维护，应对多变的需求。

设计模式（Design pattern）是一套被反复使用、多数人知晓的、经过分类编目的、代码设计经验的总结。

以往是代码的复用，现在是经验的复用。我们现在要做的就是把这些模式装进我们脑子里，尽量用到我们的工作中。

**设计模式的分类**

总体来说设计模式分为三大类：

创建型模式，共五种：工厂方法模式、抽象工厂模式、单例模式、建造者模式、原型模式。

结构型模式，共七种：适配器模式、装饰器模式、代理模式、外观模式、桥接模式、组合模式、享元模式。

行为型模式，共十一种：策略模式、模板方法模式、观察者模式、迭代子模式、责任链模式、命令模式、备忘录模式、状态模式、访问者模式、中介者模式、解释器模式。

其实还有两类：并发型模式和线程池模式。

**设计模式的好处：**

1. 经验复用， 看的远不是你非得视力多好， 站在别人肩膀上也未尝不可。

2. 共享词汇（装）

用一个图片来整体描述一下：



**策略模式**

**概念**：策略模式是指对一系列的[算法](https://baike.baidu.com/item/%E7%AE%97%E6%B3%95)定义，并将每一个算法封装起来，而且使它们还可以相互替换。策略模式让算法独立于使用它的客户而独立变化。

（独立，我爱咋改咋改，对你没有影响）

**组成**

a. 策略类，通常由一个接口或者抽象类实现。

b. 具体策略角色：包装了相关的算法和行为。

c. 环境角色：持有一个策略类的引用，最终给客户端调用。

继承（extend,IS-A）

实现(implement)

有一个(IS-A)

**此处应该有张图**

在策略模式的设计中我们会提到这样几个概念： **继承，多态，接口，抽象类**

面向对象的三大特性：**封装、继承、多态**

封装：

1.定义：隐藏对象的属性和实现细节，仅对外公开接口,控制在程序中属性的读和修改的访问级别。首先是抽象，把事物抽象成一个类，其次才是封装，将事物拥有的属性和动作隐藏起来，只保留特定的方法与外界联系

2.封装的目的是：增强安全性和简化编程.封装符合面向对象设计原则的第一条：单一性原则，一个类把自己该做的事情封装起来，而不是暴露给其他类去处理，当内部的逻辑发生变化时，外部调用不用因此而修改，他们只调用开放的(一特定的访问权限)接口，而不用去关心内部的实现

**继承：**

1.目的：实现代码的复用。

2.介绍：当两个类具有相同的特征（属性）和行为（方法）时，可以将相同的部分抽取出来放到一个类中作为父类，其它两个类继承这个父类

**多态 :**

1.概念：相同的事物，调用其相同的方法，参数也相同时，但表现的行为却不同。

2.Java实现多态有三个必要条件：继承、重写、向上转型。

3.多态的实现方式：

（1）基于继承实现的多态（2）基于接口实现的多态

**原则：**

1.会变化的独立出来，不要和经常变化的混在一起。

2.多用组合，少用继承。 （方法/函数不是继承来的，而是和适当的行为对象“组合”来的）

**优缺点**

**优点**

1、 策略模式提供了管理相关的算法族的办法。策略类的等级结构定义了一个算法或行为族。恰当使用继承可以把公共的代码转移到父类里面，从而避免重复的代码。

2、 策略模式提供了可以替换继承关系的办法。继承可以处理多种算法或行为。如果不是用策略模式，那么使用算法或行为的环境类就可能会有一些子类，每一个子类提供一个不同的算法或行为。但是，这样一来算法或行为的使用者就和算法或行为本身混在一起。决定使用哪一种算法或采取哪一种行为的逻辑就和算法或行为的逻辑混合在一起，从而不可能再独立演化。继承使得动态改变算法或行为变得不可能。

3、 使用策略模式可以避免使用多重条件转移语句。多重转移语句不易维护，它把采取哪一种算法或采取哪一种行为的逻辑与算法或行为的逻辑混合在一起，统统列在一个多重转移语句里面，比使用继承的办法还要原始和落后。

**缺点：**

1、客户端必须知道所有的策略类，并自行决定使用哪一个策略类。这就意味着客户端必须理解这些算法的区别，以便适时选择恰当的算法类。换言之，策略模式只适用于客户端知道所有的算法或行为的情况。

2、 策略模式造成很多的策略类，每个具体策略类都会产生一个新类。有时候可以通过把依赖于环境的状态保存到客户端里面，而将策略类设计成可共享的，这样策略类实例可以被不同客户端使用。换言之，可以使用享元模式来减少对象的数量。

**设计模式的六大原则**

**1、开闭原则（Open Close Principle）**

开闭原则就是说**对扩展开放，对修改关闭**。在程序需要进行拓展的时候，不能去修改原有的代码，实现一个热插拔的效果。所以一句话概括就是：为了使程序的扩展性好，易于维护和升级。想要达到这样的效果，我们需要使用接口和抽象类，后面的具体设计中我们会提到这点。

**2、里氏代换原则（Liskov Substitution Principle）**

里氏代换原则(Liskov Substitution Principle LSP)面向对象设计的基本原则之一。 里氏代换原则中说，任何基类可以出现的地方，子类一定可以出现。 LSP是继承复用的基石，只有当衍生类可以替换掉基类，软件单位的功能不受到影响时，基类才能真正被复用，而衍生类也能够在基类的基础上增加新的行为。里氏代换原则是对“开-闭”原则的补充。实现“开-闭”原则的关键步骤就是抽象化。而基类与子类的继承关系就是抽象化的具体实现，所以里氏代换原则是对实现抽象化的具体步骤的规范。—— From Baidu 百科

**3、依赖倒转原则（Dependence Inversion Principle）**

这个是开闭原则的基础，具体内容：真对接口编程，依赖于抽象而不依赖于具体。

**4、接口隔离原则（Interface Segregation Principle）**

这个原则的意思是：使用多个隔离的接口，比使用单个接口要好。还是一个降低类之间的耦合度的意思，从这儿我们看出，其实设计模式就是一个软件的设计思想，从大型软件架构出发，为了升级和维护方便。所以上文中多次出现：降低依赖，降低耦合。

**5、迪米特法则（最少知道原则）（Demeter Principle）**

为什么叫最少知道原则，就是说：一个实体应当尽量少的与其他实体之间发生相互作用，使得系统功能模块相对独立。

**6、合成复用原则（Composite Reuse Principle）**

原则是尽量使用合成/聚合的方式，而不是使用继承。