# **设计模式（Design Patterns）**

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。使用设计模式是为了可重用代码、让代码更容易被他人理解、保证代码可靠性。 毫无疑问，设计模式于己于他人于系统都是多赢的，设计模式使代码编制真正工程化，设计模式是软件工程的基石，如同大厦的一块块砖石一样。项目中合理的运用设计模式可以完美的解决很多问题，每种模式在现在中都有相应的原理来与之对应，每一个模式描述了一个在我们周围不断重复发生的问题，以及该问题的核心解决方案，这也是它能被广泛应用的原因。

## 设计模式的分类

### 创建型模式（5）

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（7）

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（11）

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。

## 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。使程序的扩展性好，易于维护和升级，即接口和抽象类的使用

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

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。

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

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）

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从大型软件架构出发，为了升级和维护方便。降低依赖，降低耦合。

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

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。

## java中的23种常见设计模式

### 工厂模式（Factory Method）

对实现了同一接口的一些类进行实例的创建

### 抽象工厂（Abstract Factory）

创建多个工厂类，这样一旦需要增加新的功能，直接增加新的工厂类就可以了，不需要修改之前的代码

### 单例模式（Singleton）

保证在一个JVM中，该对象只有一个实例存在

### 建造者（Builder）

将各种产品集中起来进行管理，用来创建复合对象

### 原型模式（Prototype）

将一个对象作为原型，对其进行复制、克隆，产生一个和原对象类似的新对象

### 适配器模式（Adapter）

将某个类的接口转换成客户端期望的另一个接口表示，目的是消除由于接口不匹配所造成的类的兼容性问题

### 装饰模式（Decorate）

装饰模式就是给一个对象增加一些新的功能，而且是动态的，要求装饰对象和被装饰对象实现同一个接口

### 代理模式（Proxy）

代理模式就是多一个代理类出来，替原对象进行一些操作，为其他对象提供一种代理以控制对这个对象的访问。客户访问不到被代理对象，可以通过代理对象间接的访问

用代理模式，代理类（proxy class）可以对它的客户隐藏一个对象的具体信息。因此，当使用代理模式的时候，我们常常在一个代理类中创建一个对象的实例。并且，当我们使用装饰器模 式的时候，我们通常的做法是将原始对象作为一个参数传给装饰者的构造器。

### 外观模式OR门面模式（Facade）

为子系统中的一组接口提供一个一致的界面，此模式定义了一个高层接口，简化了系统的使用，简单的理解就是比如说每个系统提供一个独特的功能，但是一个业务现需要多个系统共同工作完成，那么把一个业务线定义一个简单调用，自主调用多个系统功能完成一个业务线，简化用户体验

### 桥接模式（Bridge）

桥接模式就是把事物和其具体实现分开，使他们可以各自独立的变化

### 组合模式（Composite）

用预处理类似于树形结构数据，递归调用方式解决问题，把类与类构建结合关系，统一处理

### 享元模式（Flyweight）

实现对象的共享，即共享池，当系统中对象多的时候可以减少内存的开销，通常与工厂模式一起使用。类似于数据库连接池，想象一下String s1 = “abc”; String s2 = “abc”; s1 == s2; 结果为true 因为s1和s2引用同一个地址空间，这就是最简单的享元模式



### 策略模式（Strategy）

并将每个算法封装起来，使他们可以相互替换，且算法的变化不会影响到使用算法的客户。需要设计一个接口，为一系列实现类提供统一的方法，多个实现类实现该接口，设计一个抽象类（可有可无，属于辅助类），提供辅助函数

### 模板模式（Template）

个抽象类中，有一个主方法，再定义1...n个方法，可以是抽象的，也可以是实际的方法，定义一个类，继承该抽象类，重写抽象方法，通过调用抽象类，实现对子类的调用，

完成一件事情，有固定的数个步骤，但是每个步骤根据对象的不同，而实现细节不同；就可以在父类中定义一个完成该事情的总方法，按照完成事件需要的步骤去调用其每个步骤的实现方法。每个步骤的具体实现，由子类完成。

### 观察者模式（Observer）

类似于邮件订阅和RSS订阅，当我们浏览一些博客或wiki时，经常会看到RSS图标，就这的意思是，当你订阅了该文章，如果后续有更新，会及时通知你。其实，简单来讲就一句话：当一个对象变化时，其它依赖该对象的对象都会收到通知，并且随着变化

### 迭代子模式（Iterator）

迭代器模式就是顺序访问聚集中的对象

### 责任链模式（Chain of Responsibility）

有多个对象，每个对象持有对下一个对象的引用，这样就会形成一条链，请求在这条链上传递，直到某一对象决定处理该请求。但是发出者并不清楚到底最终那个对象会处理该请求，所以，责任链模式可以实现，在隐瞒客户端的情况下，对系统进行动态的调整

### 命令模式（Command）

司令员下令让士兵去干件事情，从整个事情的角度来考虑，司令员的作用是，发出口令，口令经过传递，传到了士兵耳朵里，士兵去执行。这个过程好在，三者相互解耦，任何一方都不用去依赖其他人，只需要做好自己的事儿就行，司令员要的是结果，不会去关注到底士兵是怎么实现的

### 状态模式（State）

当对象的状态改变时，同时改变其行为

### 备忘录模式（Memento）

主要目的是保存一个对象的某个状态，以便在适当的时候恢复对象

### 访问者模式（Visitor）

访问者模式就是一种分离对象数据结构与行为的方法，通过这种分离，可达到为一个被访问者动态添加新的操作而无需做其它的修改的效果

### 中介者模式（Mediator）

中介者模式也是用来降低类类之间的耦合的，因为如果类类之间有依赖关系的话，不利于功能的拓展和维护，因为只要修改一个对象，其它关联的对象都得进行修改。

### 解释其模式（Interpreter）

分离实现，解释执行。通过一个解释器对象处理一个语法规则的方式，把复杂的功能分离开，然后选择需要被执行的功能，并把这些功能组合成需要解释执行的抽象语法树，再按照抽象语法树来解释执行，实现相应的功能。从本质上看，解释器模式的思路仍然是分离、封装和简化，这与很多其他模式是一样的