# **设计模式**

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。

****一、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。



****二、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。

****三、Java的23中设计模式****

从这一块开始，我们详细介绍Java中23种设计模式的概念，应用场景等情况，并结合他们的特点及设计模式的原则进行分析。

****1、工厂方法模式（Factory Method）****

工厂模式：就是建立一个工厂类，对实现了同一接口的一些类进行实例的创建。

调用工厂的一个生产方法，方法根据不同入参可以创建不同的对象。如果传递的字符串出错，则不能正确创建对象，可以优化为：一个生产方法变为多个生产特定对象方法，只供调用分别创建对象。

如果该工厂的生产方法变为静态的，就不用创建工厂实例，变为静态工厂方法。

**2**、**抽象工厂模式（Abstract Factory）**

工厂方法模式有一个问题就是，类的创建依赖工厂类，也就是说，如果想要拓展程序，必须对原工厂类进行修改，这违背了闭包原则，所以，从设计角度考虑，有一定的问题，解决就用到抽象工厂模式，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。

把工厂抽象成接口，当有新的对象要创建时，创先新的工厂实现类。无需去改动现成的代码。这样做，拓展性较好！

****3、单例模式（Singleton）****

在Java应用中，单例对象能保证在一个JVM中，该对象只有一个实例存在。这样的模式有几个好处：

1、某些类创建比较频繁，对于一些大型的对象，这是一笔很大的系统开销。

2、省去了new操作符，降低了系统内存的使用频率，减轻GC压力。

3、业务要求只能是一个对象，创建多个的话，系统完全乱了。（比如一个军队出现了多个司令员同时指挥，肯定会乱成一团），所以只有使用单例模式。

****4、建造者模式（Builder）****

抽象工厂模式与建造者模式相似，因为它也可以创建复杂对象。主要的区别是建造者模式着重于一步步得构造出复杂对象。而抽象工厂模式着重于多个系列的产品对象（简单的或是复杂的）。建造者是在最后的一步返回对象，而对于抽象工厂来说，对象是立即返回的。

****5、原型模式（Prototype）****

1. ****适配器模式 (Adapter)****

**我们需要一个类A来实现接口B，但是类A并没有实现接口B中的所有方法，而类A是不能被改变的，这时我们可以创建一个类C，它继承类A并实现接口B，这个类C就是一个适配器。适配器中的代码将接受你所拥有的接口，并产生你所需要的接口。适配器模式有两种：类适配器模式和对象适配器模式。适配即为继承关系。**

**对象适配存放原对象的一个应用实例，直接委派方法即可。**

1. ****装饰模式（Decorator）****

**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，装饰对象持有被装饰对象的实例，直接委派方法即可。**

1. ****代理模式（Proxy）****

**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。**

1. ****外观模式（Facade）****

**外观模式是为了解决类与类之间的依赖关系的，将他们的关系放在一个Facade类中，降低了类类之间的耦合度，该模式中没有涉及到接口。**

1. ****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。桥接的用意是：将抽象化与实现化解耦，使得二者可以独立变化，像我们常用的JDBC桥DriverManager一样，JDBC进行连接数据库的时候，在各个数据库之间进行切换，基本不需要动太多的代码，甚至丝毫不用动，原因就是JDBC提供统一接口，每个数据库提供各自的实现，用一个叫做数据库驱动的程序来桥接就行了。**

1. ****组合模式（Composite）****
2. ****享元模式（Flyweight）****
3. ****策略模式（strategy）****
4. ****模板方法模式（Template Method）****
5. ****观察者模式（Observer）****
6. ****迭代子模式（Iterator）****
7. ****责任链模式（Chain of Responsibility）****
8. ****命令模式（Command）****
9. ****备忘录模式（Memento）****
10. ****状态模式（State）****
11. ****访问者模式（Visitor）****
12. ****中介者模式（Mediator）****
13. ****解释器模式（Interpreter）****