23种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。

设计模式六大原则

1 开闭原则(Open Close Principle)

2 里氏代换原则(Liskov Substitution Principle)

3 依赖倒转原则(Dependence Inversion Principle)

4 接口隔离原则(Interface Segregation Principle)

5 迪米特法原则(Demeter Principle)

6 合成复用原则(Composite Reuse Principle)

状态模式(State Pattern)

先画一张UML类图

类与类，类与接口之间的关系

继承关系关系用空心三角+实线表示

实现关系接口用空心三角+虚线表示

关联关系用实线箭头表示

是一种拥有关系，它使一个类知道另一个类的属性和方法。

聚合关系用空心菱形+实线箭头表示

表示一种弱的‘拥有’关系，体现的是A对象可以包含B对象，但B对象不是A对象的一部分。是整体和部分的关系，且部分可以离开整体单独存在。

组合关系用实心菱形+实线箭头表示

表示一种强的‘拥有关系’，体现了严格的部分和整体的关系，部分和整体的生命周期一样。是整体和部分的关系，但部分不能离开整体而单独存在。

依赖关系用虚线箭头表示

是一种使用关系，及一个类的实现需要另一个的协助。

描述

在《大话设计模式》中说当一个对象的内在状态改变时，允许改变其行为，这个对象看起来像是改变了其类。在代码实现部分一个对象内在状态对应一个State接口，当State接口发生改变时，其doAction方法也发生改变，如此一来状态模式和类多态的特性不就类似了吗？

状态模式的使用场景

一个对象的行为取决于它的状态，并且它必须在运行时刻根据状态改变它的行为。

一个操作中含有庞大的多分支条件语句，且这些分支依赖于该对象的状态。这个状态通常用一个或多个枚举变量表示，通常，有多个操作包含这一相同条件结构。State模式将每一个条件分支放入一个独立的类中。这使得你可以根据对象自身的情况将对象的状态作为一个对象，这一对象可以不依赖于其他对象而独立变化。

状态模式的优点

将与特定状态相关的行为局部化，并且将不同状态的行为进行分割，通过把各种状态转移逻辑分不到State的子类之间，来减少相互依赖。

组合模式的缺点

不同状态的行为分布在多个State子类中，增加了子类的数目。

策略模式(Strategy Pattern)

意图：定义算法家族，分别封装起来，让它们之间可以互相转换，此模式让算法变化，不会

影响使用算法的客户。

适用性

许多相关的类仅仅是行为有异。“策略”提供了一种用多个行为中的一行行为来配置一个类的方法。

需要使用一个算法的不同变体。

算法的使用客户不应该知道的数据。可以使用策略模式以避免暴露复杂的、与算法相关的数据结构。

一个类定义了多种行为，并且这些行为在这个类的操作中以多个条件语句的形式出现。将相关的条件分支移入他们各自的Strategy类中以代替条件语句。

策略模式的优点

简化了单元测试，因为每个算法都有自己的类，可以通过自己的接口段都测试。

一个替代继承的方法 继承提供了另一种支持多种算法或者欣慰的方法。你可以直接生成一个Context类的子类，从而给它以不同的行为。但这会将行为硬行编制到Context中，而将算法的实现与Context的实现混合起来，从而使Context难以理解、难以维护、难以扩展，而且还不能动态的改变算法。最后你得到一堆相关的类，它们之间唯一的差别是它们所使用的算法或行为。将算法封装在独立的Strategy类中使得你可以独立于Context改变它，使它易扩展、易于切换、易于理解。

消除了一些条件语句 Strategy模式提供了用条件语句选择所需的行为以外的另一种选择。当不同的行为堆砌在一个类中，很难避免使用条件语句来选择合适的行为。将行为封装在一个独立的Strategy类中消除这些条件语句。

策略模式的缺点

客户必须了解不同的Strategy

增加了对象的数目

模板方法(Template Method)

意图：定义一个操作中的算法骨架，而将一些步骤延迟到子类中，TemplateMethod使得子类可以不改变一个算法的结构即可以重新定义该算法的某些特定步骤。

动机：

适用性：

一次性实现一个算法的不变部分，并将可变的行为留给子类来实现。

各子类中公共的行为应该被提取出来集中到一个公共父类中以避免代码重复。

控制子类的扩展

访问者模式(Visitor Pattern)

意图

表示一个作用于某对象结构中的各元素的操作。它使你可以在不改变各元素的类的前提下定义作用于这些元素的新操作

动机

将每个类中相关的操作包装在一个独立的对象(称之为Visitor)中，并在传递给当前访问元素。当一个元素“接受”该访问者时，该元素向访问者发送一个包含自身星系的请求。该请求同时也将元素本身作为一个参数。然后访问者将为该元素执行该操作——这一操作以前是在该元素的类中的。

适用性

一个对象结构包含很多类对象，他们有不同的接口，而你想对这些对象实施一些依赖于其具体类的操作

需要对一个对象结构中的对象进行很多不同的并且不相关的操作，而你想避免让这些操作“污染”这些对象的类。Visitor使得你可以将相关的操作集中起来定义在一个类中。当该对象结构被很多应用共享式，用Visitor模式让每个应用仅包含需要用到的操作。

定义对象结构的类很少改变，但经常需要在机构上定义新的操作。改变对象结构需要重定义对对所有访问者的接口，这可能需要很大的代价。如果对象结构经常改变，那么可能还是在这些类中定义这些操作比较好。