版本号：v1.1 阶段：初稿

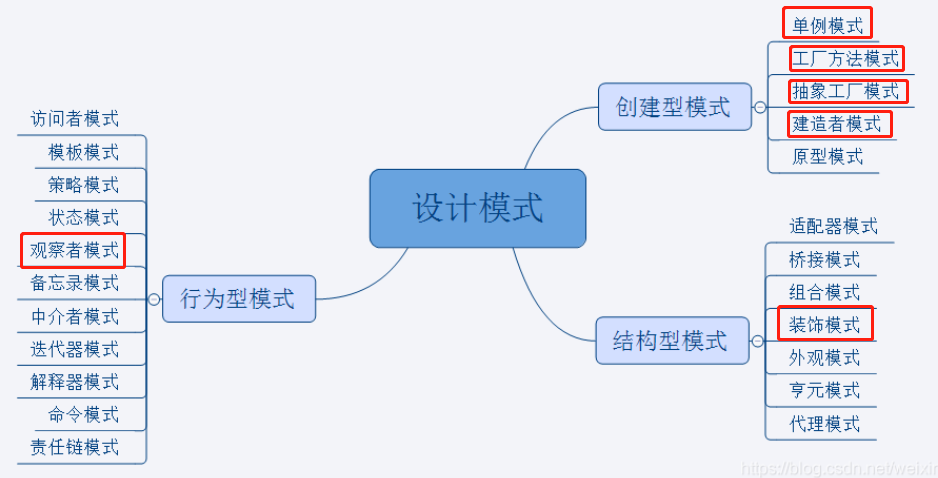
# 设计模式面试宝典

MRL Liu

（未经许可，不得传播）

2022年02月02日

**设计模式**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、程序的重用性。**设计模式**一共有23种，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。如下



## 一、设计模式的六大原则

### 1、开放封闭原则

思想：尽量通过扩展软件实体来解决需求变化，而不是通过修改已有的代码来完成变化。

描述：一个软件产品在生命周期内，都会发生变化，既然变化是一个既定的事实，我们就应该在设计的时候尽量适应这些变化，以提高项目的稳定性和灵活性。

优点：单一原则告诉我们，每个类都有自己负责的职责，里氏替换原则不能破坏继承关系的体系。

### 2、里氏代换原则

思想：使用的基类可以在任何地方使用继承的子类，完美的替换基类。

描述：子类可以扩展父类的功能，但不能改变父类原有的功能。子类可以实现父类的抽象方法，但不能覆盖父类的非抽象方法，子类中可以增加自己特有的方法。

优点：增加程序的健壮性，即使增加了子类，原有的子类还可以继续运行，互不影响。

### 3、依赖倒转原则

思想：面向接口编程，依赖于抽象而不依赖于具体

描述：它要求我们在程序代码中传递参数时或在关联关系中，尽量引用层次高的抽象层类，这个是开放封闭原则的基础。

### 4、接口隔离原则

描述：使用多个隔离的接口，比使用单个接口要好。

例如：支付类的接口和订单类的接口，需要把这俩个类别的接口变成俩个隔离的接口

优点：降低依赖，降低耦合。

### 5、迪米特法则（最少知道原则）

思想：一个对象应当对其他对象有尽可能少地了解，简称类间解耦

描述：一个类尽量减少自己对其他对象的依赖，原则是低耦合，高内聚，只有使各个模块之间的耦合尽量的低，才能提高代码的复用率。

优点：低耦合，高内聚。

### 6、单一职责原则

思想：一个方法只负责一件事情。

描述：单一职责原则很简单，一个方法 一个类只负责一个职责，各个职责的程序改动，不影响其它程序。 这是常识，几乎所有程序员都会遵循这个原则。

优点：降低类和类的耦合，提高可读性，增加可维护性和可拓展性，降低可变性的风险。

## 二、面试常见的设计模式

### 1、创建型模式

#### （1）单例模式

**单例模式**（Singleton）可能是最简单、最常见也最有效的一种设计模式。**单例模式用于创建整个程序全局唯一的一个对象**，例如整个Windows系统只能打开一个任务管理器。

单例模式的实现借助了类的static成员变量，其实现可以简单分为如下三点：

1. 将构造函数私有化，外部无法调用
2. 在类中添加一个static的类指针，用来指向构造的唯一一个对象。
3. 提供一个公有的创建对象方法，外部只能通过该方法创建其对象。

代码如下：



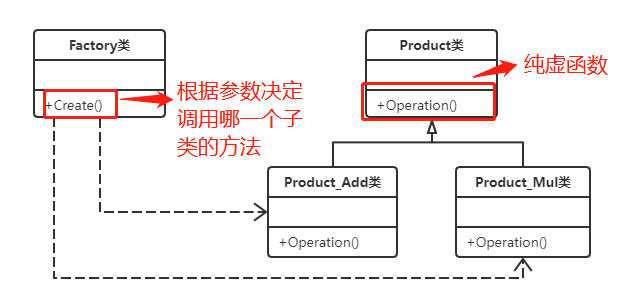
#### （2）工厂模式

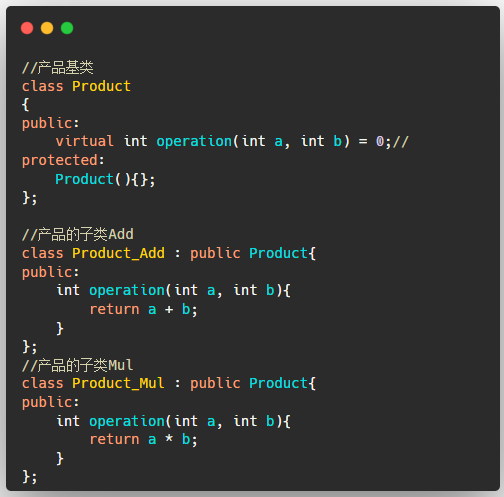
工厂模式（Factory）简单来讲就是让创建对象有了统一的接口，具体化类的工作延迟到了子类中，其实现用到了C++的继承和纯虚函数的知识。

工厂模式严格来讲有三种类型：简单工厂模式、工厂模式和抽象工厂模式。

##### 简单工厂模式

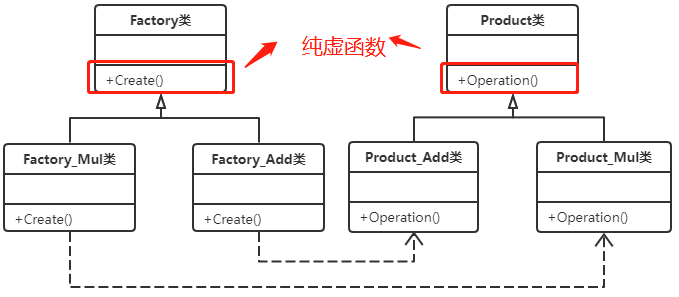
我们将被创建对象类命名为Product，创建对象的类命名为Factory。Factory调用Product的统一接口operation()，其具体实现由各自的子类决定。

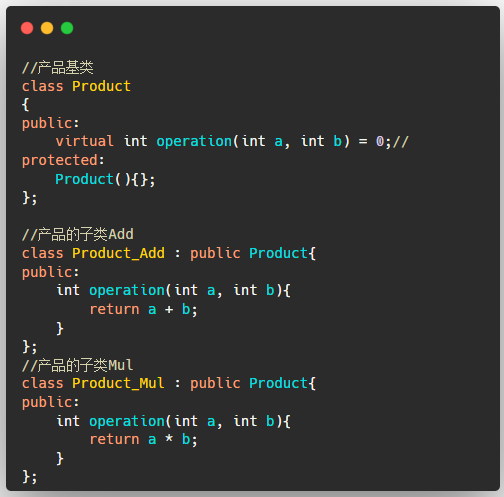




##### 工厂模式

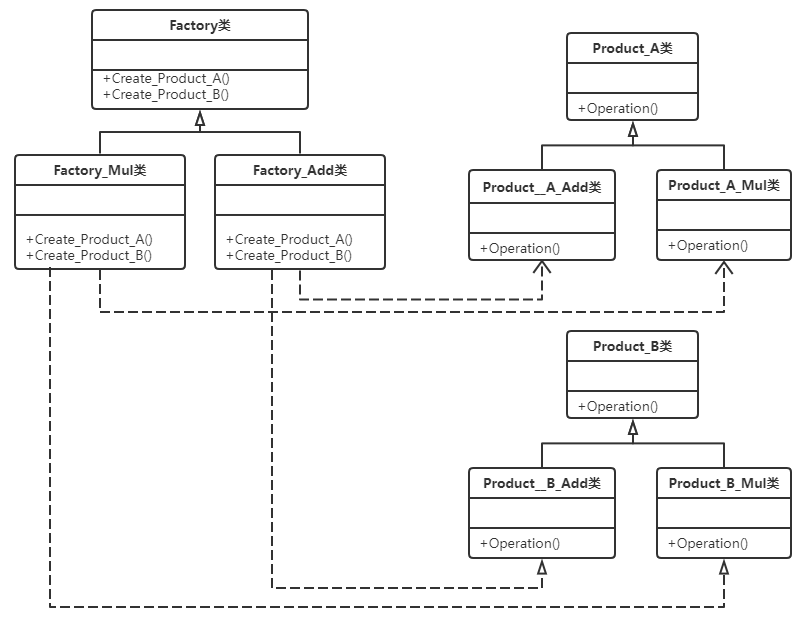
上述的简单工厂模式中还存在不足就是每次增加一个新的Product子类，都需要修改Factory的内部代码，这违背了开放-封闭原则（对扩展开放，对修改封闭）。工厂模式抽象出了Factory基类，不同的Factory子类负责创建一个对应的产品，所以可以修改如下。

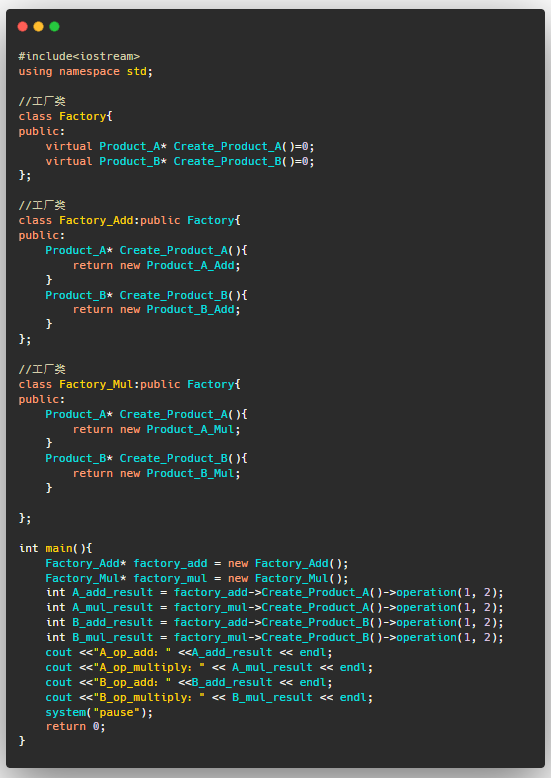
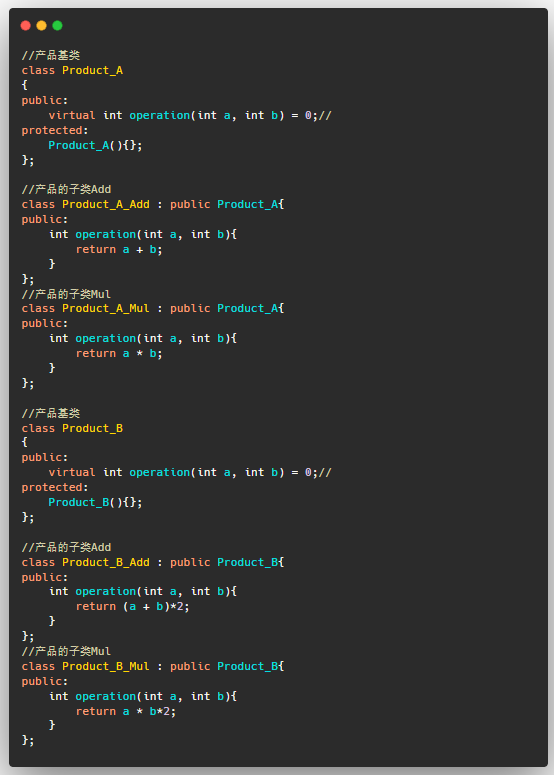




##### 抽象工厂模式

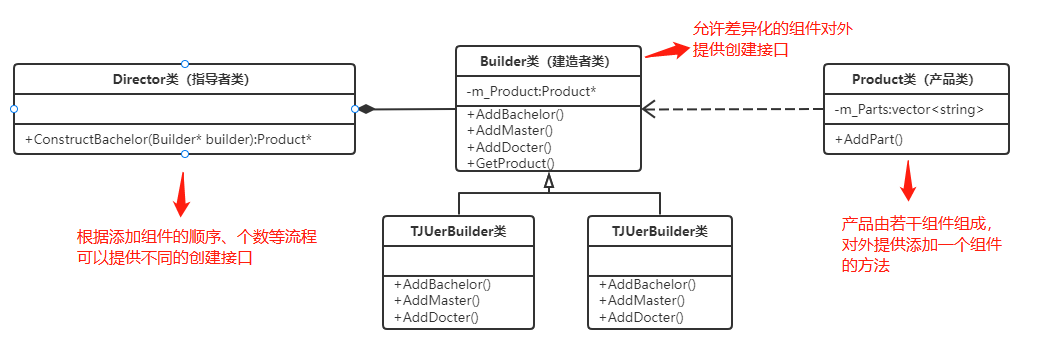
工厂模式中，一个工厂只生产一种产品，如果想要一个工厂生产多个产品，就需要抽象工厂模式，抽象工厂模式提供了创建一系列相互依赖对象的接口。抽象工厂模式本质上是简单工厂模式和工厂模式的组合。

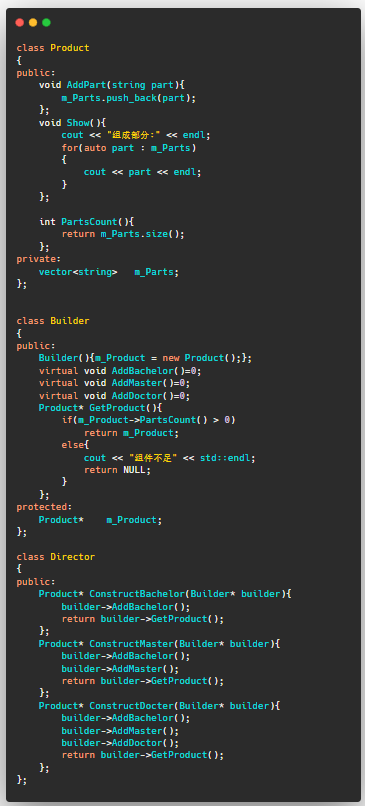




#### （3）建造者模式

建造者模式（Builder）适用于创建包含不同组件的对象，它分为Product、Builder和Director三部分，分别定义了产品组件的添加方式、差异组件的扩展方式、组件装配流程的对外接口，如下：



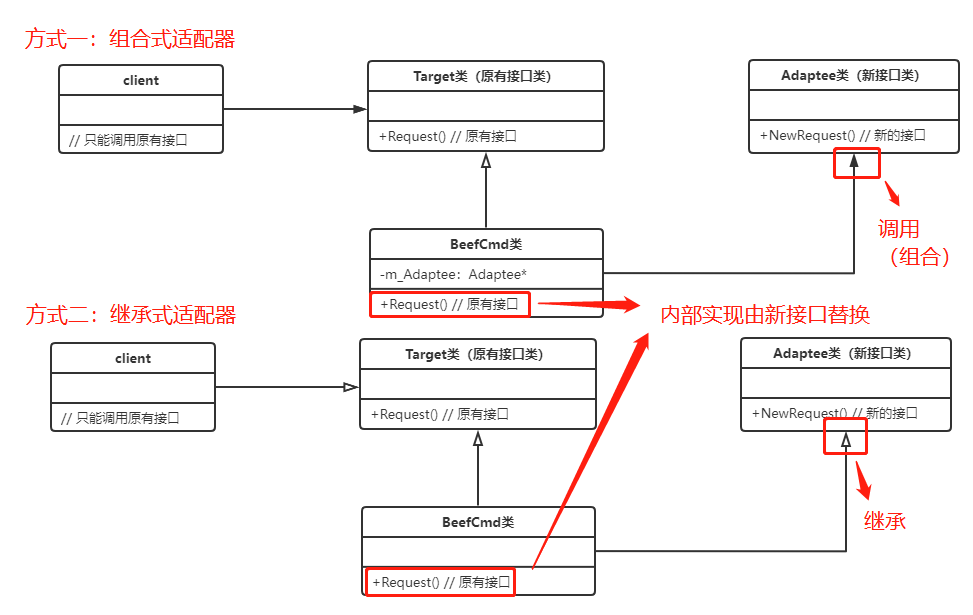


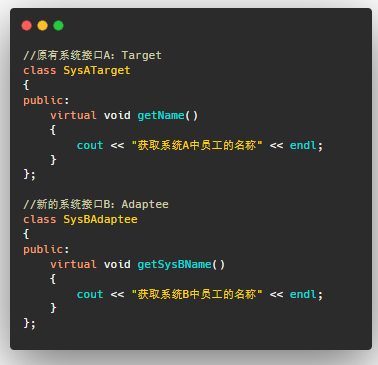
### 2、结构型模式

#### （1）适配器模式

适配器模式（Adapter）是用来在旧的系统接口中替换成新的系统接口的设计模式，简单说对于同一个功能存在一个老版本的类对象A，因为需求升级开发了新的类对象B，A和B实现相同的功能但是类的接口名称不一致，原有程序无法直接调用类B。而且由于系统生态的原因，大部分程序只能调用类A的接口，所以也不能舍弃A。

这种情况下就得使用适配器模式。假设存在一个原有接口Target和一个新的接口Adaptee，现在要使Target中的接口的实现被替换成Adaptee的实现。适配器模式的做法是首先从Target派生出一个子类Adapter，Adapter可以调用Adaptee的函数（组合方式）或继承Adaptee的函数来替换自身父类的实现。

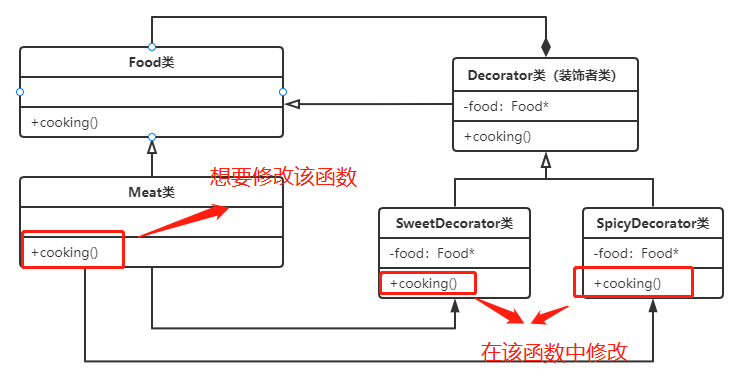




#### （2）装饰者模式

装饰者模式（Decorator）也叫包装模式。如果给一个已有的类添加新的实现方法，一种思路是继承该类，在子类中添加；另一种思路是组合。装饰者模式就是利用组合的方式在已由类添加新的功能。

假如现在有个Food类，其有一个公有接口Cooking()，Food类派生出了1个子类Meat，其Cooking()会输出肉。



这时候由于新的需求，需要修改Meat类的Cooking()，使其输出加了辣味或甜味的肉，如果不直接修改Meat类，一种方式是继承自Meat类，重写其Cooking()。装饰者模式的思路是不继承Meat()类，而是在Food派生出的新的子类Decorator，这个Decorator具有一个Food指针，从Decorator派生出更多子类，例如SweetDecorator。

由于Decorator和Meat的父类都是Food，所以SweetDecorator也可修饰Decorator类及其子类。如果想要一块添加了辣味和甜味的肉，不需要再新加装饰类，只需要用将SweetDecorator作为参数构造出一个SpicyDecorator类即可，可见代码。

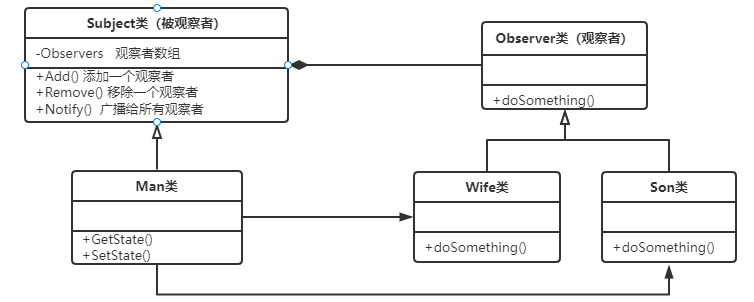


### 3、行为型模式

#### （1）观察者模式

观察者模式（Observer）很可能是应用最多、影响最广的设计模式之一，也叫发布-订阅模式。观察者模式的核心是一对多的通知，当被观察者状态发生变化时（发生某个事件）通知所有观察者做出响应。

经典应用场景：微信朋友圈动态通知、消息通知、邮件通知等。经典的MVC（Model/View/Control）结构就采用了其思路，其将业务逻辑和表示层进行接口。



代码实现思路：被观察者发生某个事件时所有观察者都要做出相应，这就要求被观察者要维护一个观察者的数组，然后当某个函数（暂定为notify）被调用时就遍历该数组调用观察者的函数，这就要求所有的观察者提供一个统一的接口（暂定为doSomething）。

观察者Observer和被观察者Subject的接口实现也非常简单，主要是被观察者要维护一个vector数组。



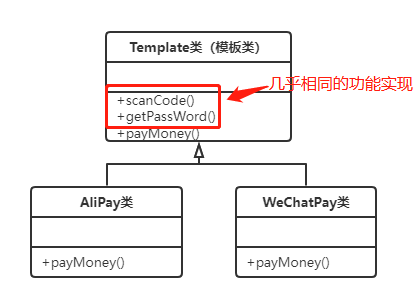
测试场景，丈夫回家发通知给妻子和孩子，丈夫是被观察者，妻子和孩子是观察者。如下：



#### （2）模板模式

模板模式（template）是一个非常自然也非常简单的设计模式，我们甚至不知不觉都在使用它。它解决的是如何组织业务（算法）逻辑相似的对象，例如微信支付、支付宝支付。模板模式采用继承的方式实现这一点，为算法逻辑相似的对象定义一个共同的抽象基类，子类负责实现具体的细节。

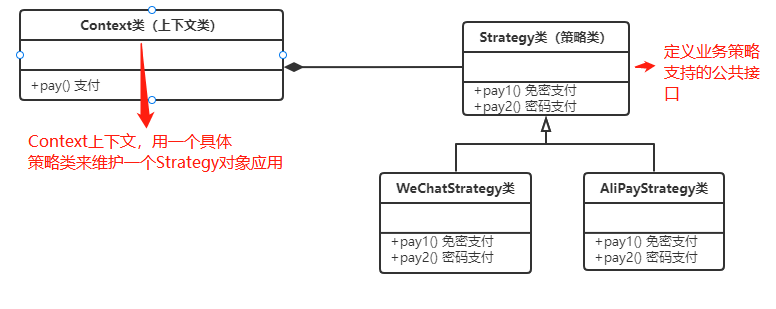
生活中常见的模板模式的例子：付款购物的流程：扫描获取二维码、获取支付密码、支付金额，其中支付金额可以由微信支付和支付宝支付两种接口来完成，其他的则可以使用同一套方法。





#### （3）策略模式

策略模式（Strategy）和模板模式解决的都是同一种问题，即如何组织业务（算法）逻辑相似的对象。例如开发一个支付模块，有微信支付和支付宝支付2种软件包的各种接口供调用，由于这两种支付是不同厂商开发的，接口名称可能都不统一。前面所述的模板模式直接为这2种支付定义一个共同基类；策略模式的方式是定义一个Strategy类，其中定义支付业务都会用到的公共接口，由此派生出两个具体的支付策略类。Context类负责根据Strategy类提供的公共接口来开发具体业务：例如支付时选用免密支付方式还是密码支付方式。



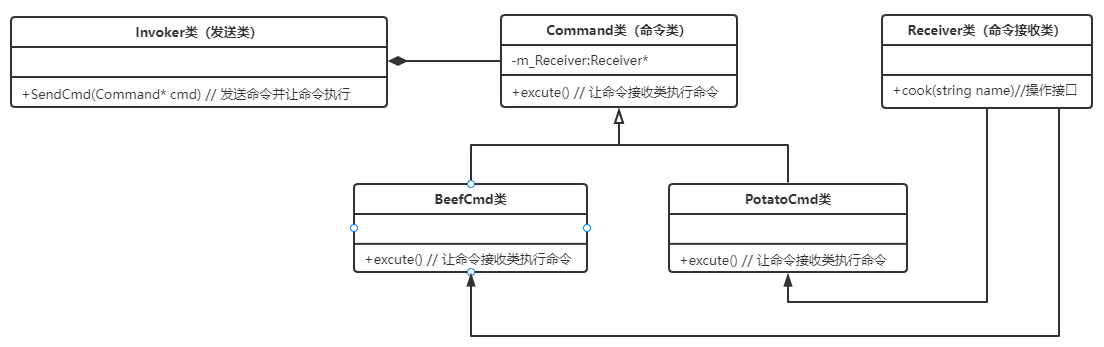
策略模式的优点是使用继承的方式已于修改和扩展；缺点是破坏了类的封装性，继承中父类的实现细节暴露给了子类，策略类会增多，所有的策略类都会对外暴露。



#### （4）命令模式

命令模式（Command）可以实现消息通信的发送者和接收者的完全解耦，发送者和接收者之间没有直接饮用官谢，发送者只需要知道如何发送请求。

例如服务员和厨师的沟通场景，厨师是命令接收者Receiver，服务员是命令发送者Invoker。普通写法下，Invoker根据不同的命令直接调用Receiver的不同操作函数，这样子开发者需要频繁修改Invoker类。命令模式下，Receiver和Invoker之间添加一个Command类，Invoker只调用Command的一个统一接口，不同的命令由Command不同的子类内部进行处理，新加一个命令时，开发者不需要改动Invoker，只需要继承一个Command子类，然后在其内部调用Receiver的合适操作接口即可。





命令模式的优点是将命令调用者和命令执行者的对象进行了解耦，添加新的命令只需要继承命令类即可。缺点是增加了类的数量，有几个具体命令就有多少具体类，可能需要大量具体命令类。

# 迭代日志表

|  |  |  |
| --- | --- | --- |
| **迭代版本** | **迭代工作** | **迭代日期** |
| V1.0 | 建立初稿文档，完成第1章和第2章的整理。 | 2022-02.02-02.05 |
| V1.1 |  |  |
| V1.2 |  |  |