设计的广义概念

1. 面向对象
2. 面向服务
3. 框架：分析经典架构
4. 设计原则：单一职责、开放封闭、依赖倒置、接口隔离、Liskov替换
5. 设计模式：需要掌握：Abstract Factory、Iterator、Singleton、Adapter、Decorator、Observer、Façade、Template、Command。
6. 模式之外：MVC、Repository等

典型的设计模式

参考网址：<https://design-patterns.readthedocs.io/zh_CN/latest/structural_patterns/proxy.html>

<https://www.cnblogs.com/zhili/p/DesignPatternSummery.html>

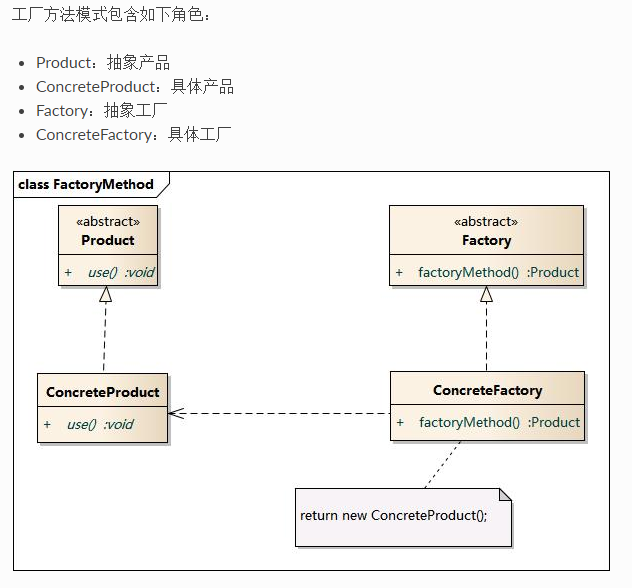
1. 创建型模式：

含义：创建型模式核心关注点就在于对象创建的依赖关系上，将对象的依赖从new操作中解脱出来，隔离应用系统和类型实例之间的依赖。例如通过引入工厂，将对象创建职责委托于工厂类，解除了应用系统与类型对象在实例化过程中的直接引用。

模式：工厂方法、抽象工厂、单例模式、建造者模式、原型模式

UML

工厂方法

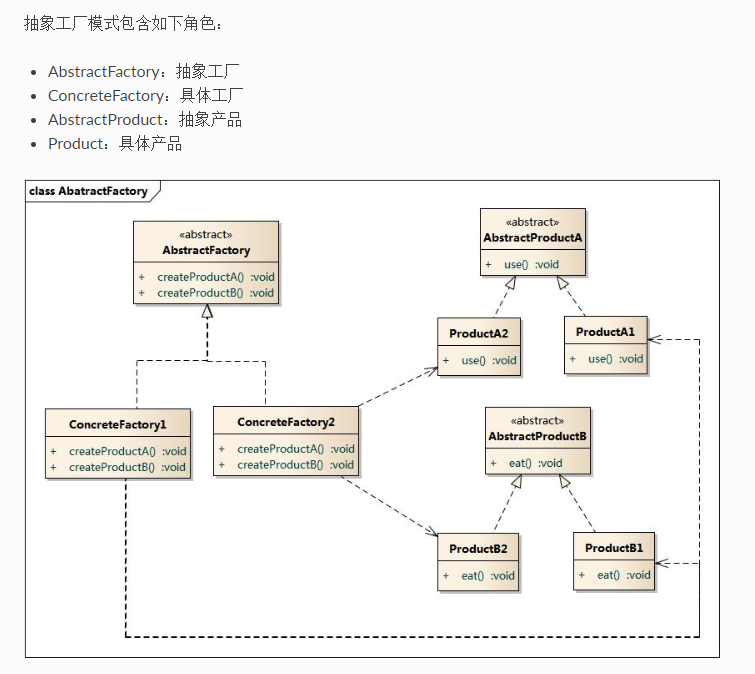


工厂方法模式是将简单工厂模式进一步抽象和扩展。由于面向对象的多态性，工厂方法模式保持了简单工厂的优点，使得具体产品由对应的工厂类实现，这样就可以对扩展开放。

抽象工厂

动机：在工厂方法中，具体工厂负责生产具体产品，每一个具体工厂对应一种具体产品。一般情况下，一个具体工厂中只有一个工厂方法或者一组重载的工厂方法。但是有时候我们需要一个工厂可以提供多个产品对象，而不是单一产品。

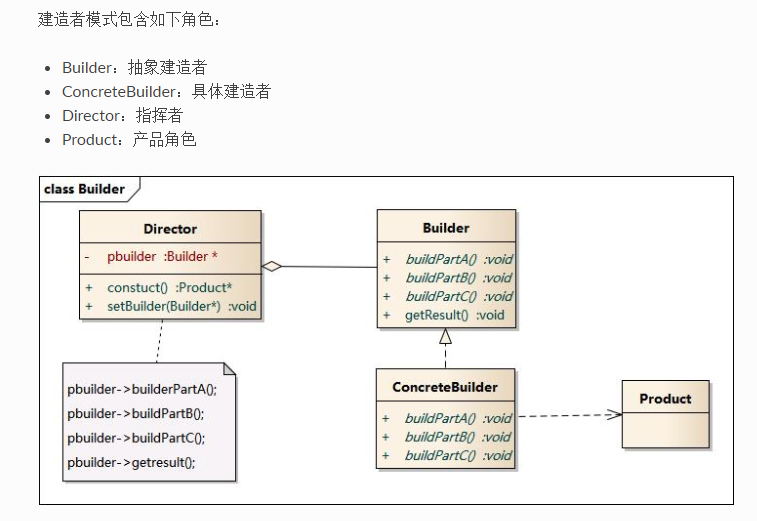
抽象工厂和工厂方法最大的区别是，工厂方法针对的是一个产品等级结构，而抽象工厂则需要面对多个差评等级结构，一个工厂等级结构可以负责生产多个不同产品的等级结构。



建造者模式

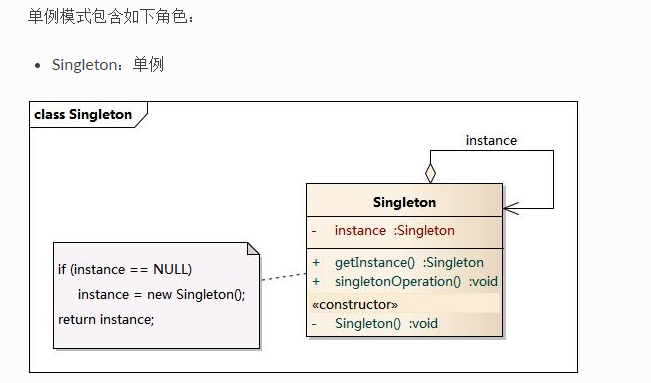
动机：无论现实世界还是软件系统中，都存在一些复杂的对象，他们拥有多个组成部分，如汽车就可以包括车轮，方向盘，发送机等。

优点：客户端不必知道产品内部组成细节，将产品本身与产品创建过程解耦，使得相同的创建过程可以创建不同的产品对象。



单例模式：

确保：一个类只有一个实例，这个实例由自身创建并且微系统提供访问该实例的方法接口。



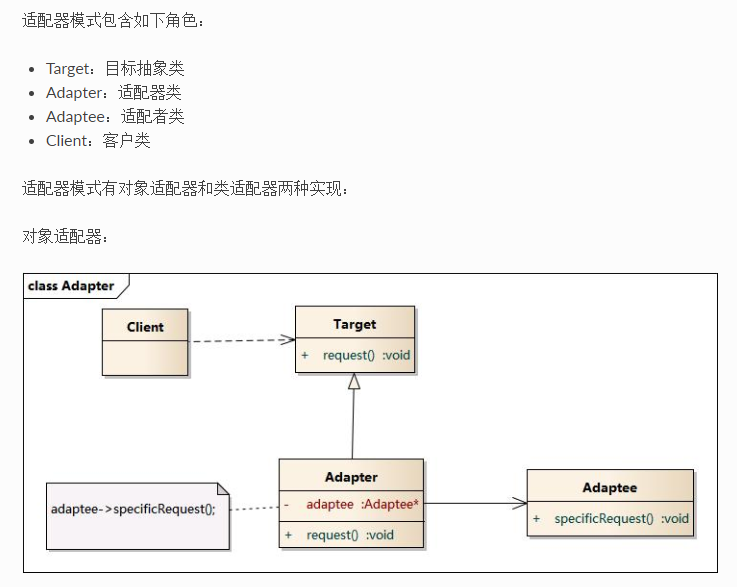
1. 结构型模式：

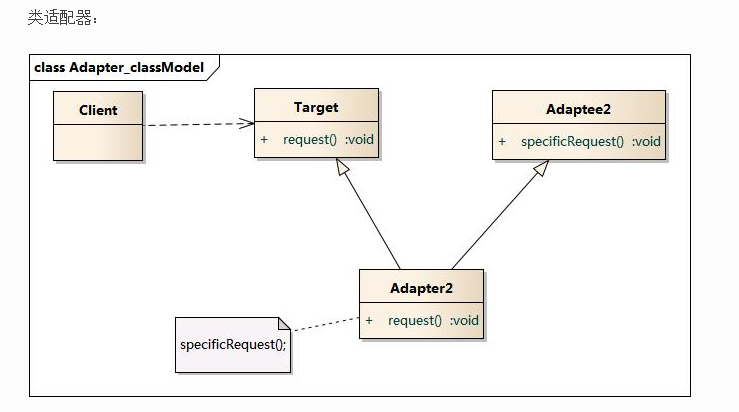
含义：结构性模式是将简单类型组合为负责类型的过程，通过灵活的设计要素，最终保证不同类型间保持尽量间接地引用和尽量松散的耦合，在复杂类型有更多变化时，以最小的代价进行兼容变化和扩展。例如：适配器模式的两种不同适配方式，分别代表通过继承和组合的方式实现；而代理模式则是通过引入代理来隔离不同层次见得依赖，同时保证真是对象的安全性和封装性。

代表模式：桥接、适配器、组合、外观、装饰、享元模式、代理

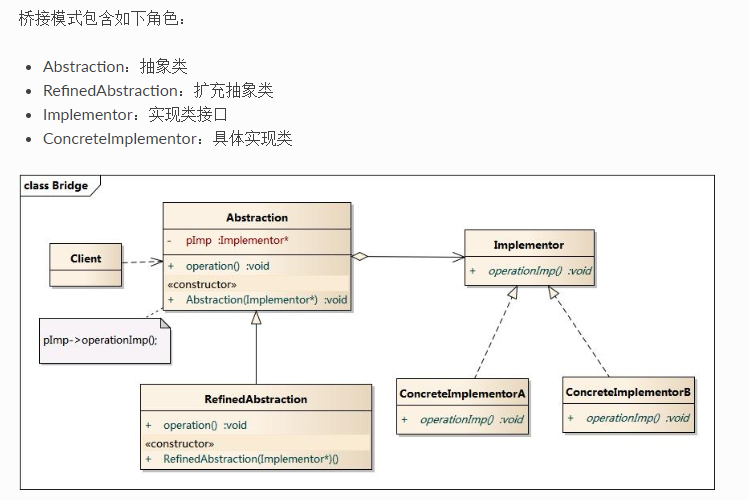
UML

适配器模式



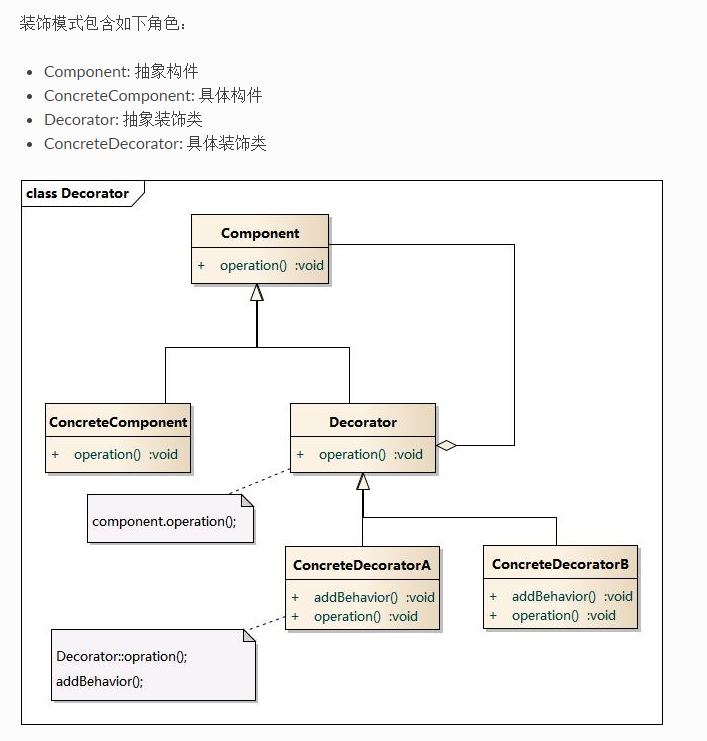


桥接模式

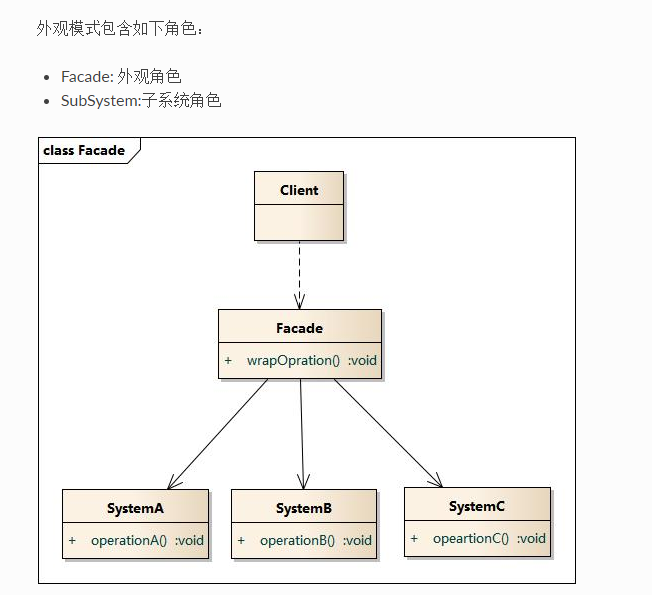


装饰者模式

动机：一般情况下，给类或者对象增加行为有两种方式：1）继承机制，这种方式是静态的，用户不能控制增加行为的方式和时机；2）关联机制，将一个类的对象嵌入到另一个对象中，由另一个对象来决定是否调用嵌入对象的行为，以便扩展自己的行为，我们称这个嵌入对象为装饰器（Decorator）



外观模式



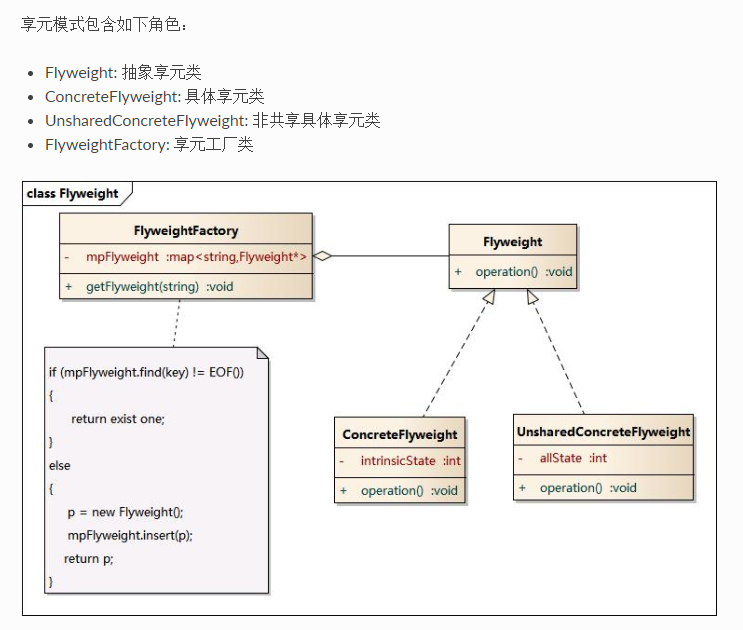
享元模式

该模式是考虑系统性能的设计模式，通过使用享元模式可以节约内存空间，提高系统性能。

享元模式的核心在于享元工厂类，享元工厂类的作用在于提供一个用于存储享元对象的享元池，用户需要对象是，首先从享元池中获取，如果享元池没有不存在，则创建提供一个新的享元对象返回给用户，并且在享元池中保存该新增对象。

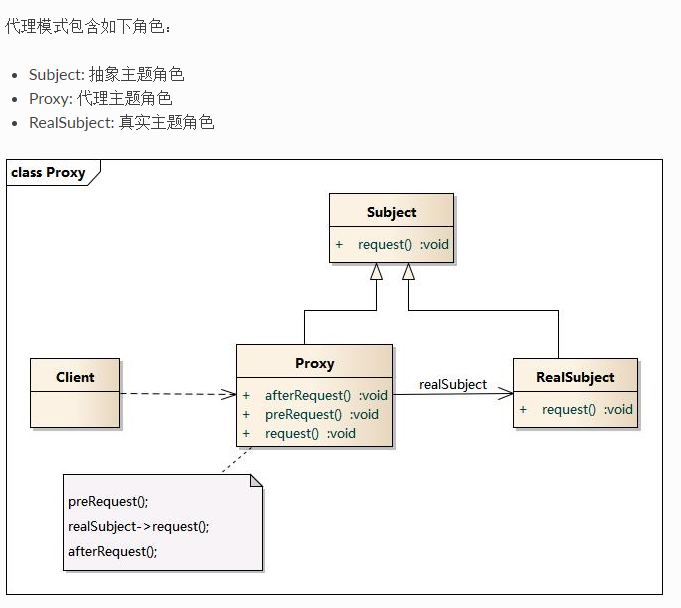
享元对象能做到共享的关键是区分内部状态和外部状态。

1. 内部状态是存在咋享元对象内部并且不随环境的改变而改变，因此内部状态可以共享
2. 外部状态是随环境改变而改变，不可以共享的状态。外部状态由客户端保存



代理模式

有时候，一个客户端不能或者不想直接引用一个对象，此时可以通过一个称之为“代理”的第三方来实现间接引用。代理对象可以在客户端和目标对象之间起到中介作用，并且可以通过代理去掉客户不能看到的内容或者添加额外服务。



1. 行为型模式：

含义：行为模式不仅仅关注类和对象的结构，而且重点关注他们之间的相互作用。通过行为型模式可以清晰地划分类与对象的职责，例如：职责连避免请求的发送者和接受者的直接耦合，而是将多个对象连成一条处理链条；命令模式核心则是将行为的请求者和行为实现者之间通过封装的命令对象解耦。

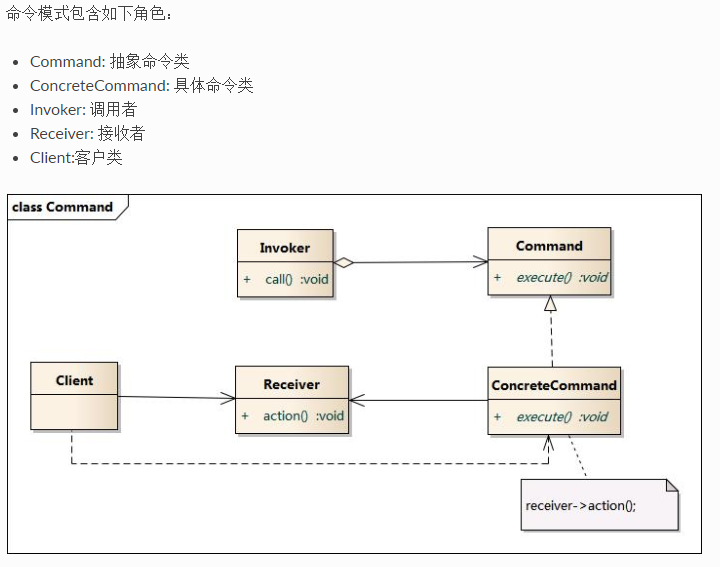
行为型模式分为类行为型模式和对象行为型模式两种：

* 类行为型模式：类的行为型模式使用继承关系在几个类之间分配行为，类行为型模式主要通过多态等方式来分配父类与子类的职责。
* 对象行为型模式：对象的行为型模式则使用对象的聚合关联关系来分配行为，对象行为型模式主要是通过对象关联等方式来分配两个或多个类的职责。根据“合成复用原则”，系统中要尽量使用关联关系来取代继承关系，因此大部分行为型设计模式都属于对象行为型设计模式。

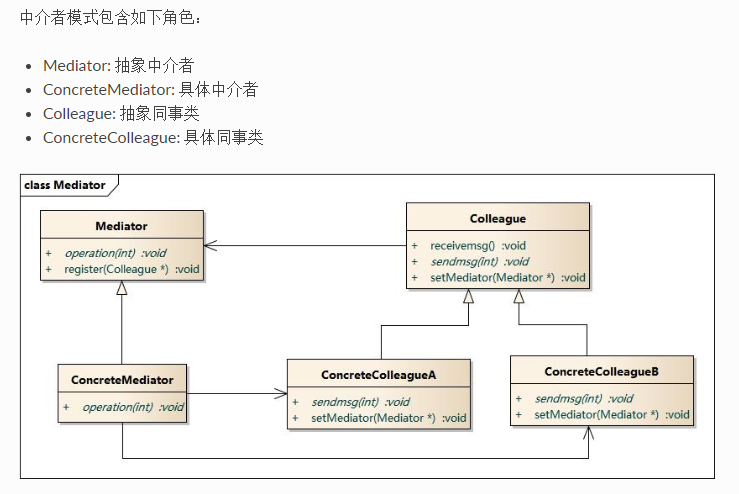
代表模式：模板方法、迭代器、中介者、职责链、解释器、命令、观察者、备忘录、状态、策略、访问者。

UML

命令模式

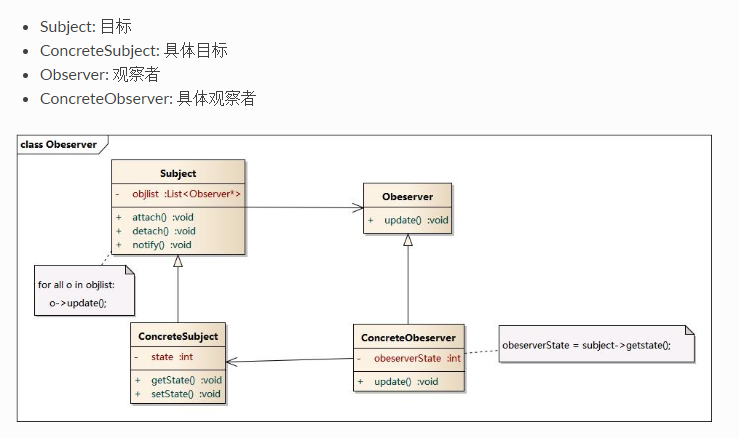


中介者模式



观察者模式

定义对象间的一种一对多依赖关系，使得每当一个对象状态发生改变时，其相关依赖对象皆得到通知并被自动更新。



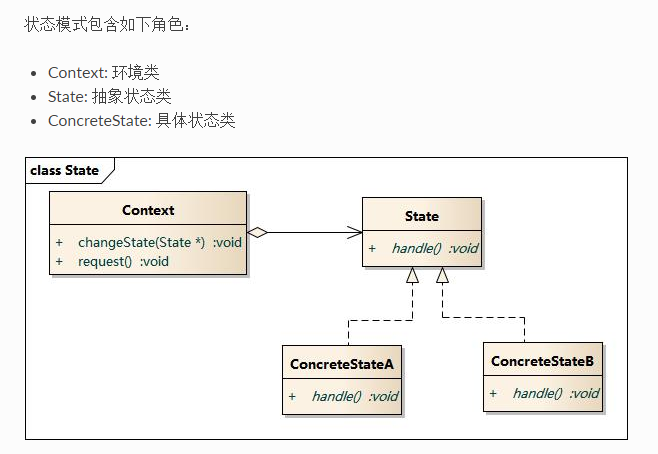
状态模式

很多情况是，一个对象的状态行为取决于一个或者多个动态变化的属性，这样的属性就叫做状态。

状态模式最关键的一点是引入了一个抽象类来专门表示对象状态，这个类就叫做抽象状态类。而对象的每一种具体状态类都继承了该类，并在不同具体状态类中实现了不同状态的行为，包括各种状态之间的转换。

在状态模式结构中需要理解环境类与抽象状态类的作用：

1. 环境类实际上就是拥有状态的对象，环境类有时候可以充当状态管理器的角色，可以在环境类中对状态进行切换操作。
2. 抽象状态类可以是抽象类，也可以是接口类，不同状态类就是集成这个父类的不同子类，状态类的产生是由于环境类存在多个状态，同时满足两个条件：这些状态经常常需要切换，在不同的状态下对象的行为不同。因此可以将不同对象下的行为单独提取出来封装在具体的状态类中，使得环境类对象在其内部状态改变时可以改变他的行为，对象看起来似乎修改了他的类，而实际上是由于切换到不同的具体状态类实现的。由于环境类可以设置为任一具体状态类，因此它针对抽象状态类进行编程，在程序运行时可以将任一具体状态类的对象设置到环境类中，从而使得环境类可以改变内部状态，并且改变行为。



策略模式

