strategy模式

场景

（1）我们可以根据不同的参数来选择一种我们想要执行的优惠计价方式

public **interface Strategy** {

   void execute();

}

public class **StrategyA implements Strategy** {

   public void execute() {

       System.out.println("策略A的功能逻辑");

  }

}

public class **StrategyB implements Strategy** {

   public void execute() {

       System.out.println("策略B的功能逻辑");

  }

}

public class **StrategyC implements Strategy** {

   public void execute() {

       System.out.println("策略C的功能逻辑");

  }

}

public class **Context** {

   private **Strategy** strategy;

   public Context(Strategy strategy) {

       this.strategy = strategy;

  }

   public void execute() {

       strategy.execute();

  }

}

策略模式，将会成为**最最高频**使用的一种设计模式，他的常见应用场景，就是替换掉那一大坨复杂难懂的if else if else。对于那种过于复杂的选择判断逻辑，完全可以将选择哪种策略的过程放到工厂里去。工厂，可以是简单工厂，也可以是工厂方法，也可以是抽象工厂。

我们这里打算用抽象工厂模式，跟策略模式结合起来，大家可以想象一下复杂的场景，一个复杂的业务逻辑里面，每个if else判断之后，要跟一堆复杂的业务逻辑，我们可以将不同的业务逻辑抽取成不同的策略，然后一个具体的工厂实现，可以创建出一个策略组合来。

到时候会结合电商的业务来设计

**策略模式跟命令模式的区别？**看起来一样的，但是用处不一样。命令是可以发送出去，然后可以经过一些队列的流转，比如先把命令发送到MQ，接着再处理。策略是说选择了一组策略，立即就要执行的，不会经过其他别的什么处理。而且策略逻辑基本就是用在复杂的if else代码中的

命令模式，可以用在更多别的场景中

思想是不一样的，也许实现上，都是接口、实现类、工厂来做的，适合的场景是不一样的

853769620