简述

在策略模式中，我们创建表示各种策略的对象和一个行为随着策略对象改变而改变的 context 对象。策略对象改变 context 对象的执行算法

主要用途：在有多种算法相似的情况下，使用 if...else 所带来的复杂和难以维护

举例说明：旅行的出游方式，选择骑自行车、坐汽车，每一种旅行方式都是一个策略

角色

策略接口、策略接口实现类、上下文（策略执行者）

代码

```
/**
 * 策略接口
 */
public interface Strategy {
   public int doOperation(int num1, int num2);//对num1和num2进行计算
}

/**
 * 增加策略
 */
public class OperationAdd implements Strategy {
   @Override
   public int doOperation(int num1, int num2) {
      return num1 + num2;
   }
}

/**
 * 相减策略
 */
public class OperationSubstract implements Strategy {
   @Override
   public int doOperation(int num1, int num2) {
      return num1 - num2;
   }
}

/**
 * 相乘策略
 */
public class OperationMultiply implements Strategy {
   @Override
   public int doOperation(int num1, int num2) {
      return num1 * num2;
```

```
    }
}

/**
 * 上下文：策略的执行者
 */
public class Context {
    private Strategy strategy;

    public Context(Strategy strategy) {
        this.strategy = strategy;
    }

    public int executeStrategy(int num1, int num2) {//执行策略
        return strategy.doOperation(num1, num2);
    }
}

public class Main {

    public static void main(String[] args) {
        System.out.print("相加");
        Context c = new Context(new OperationAdd());
        int i = c.executeStrategy(10, 20);
        System.out.println(i);
        System.out.println("----");
        System.out.print("相减");
        c = new Context(new OperationSubstract());
        i = c.executeStrategy(10, 20);
        System.out.println(i);
        System.out.println("----");
        System.out.print("相乘");
        c = new Context(new OperationSubstract());
        i = c.executeStrategy(10, 20);
        System.out.println(i);
    }

}
```

总结

优点： 1、算法可以自由切换。 2、避免使用多重条件判断。 3、扩展性良好。
缺点： 1、策略类会增多。 2、所有策略类都需要对外暴露