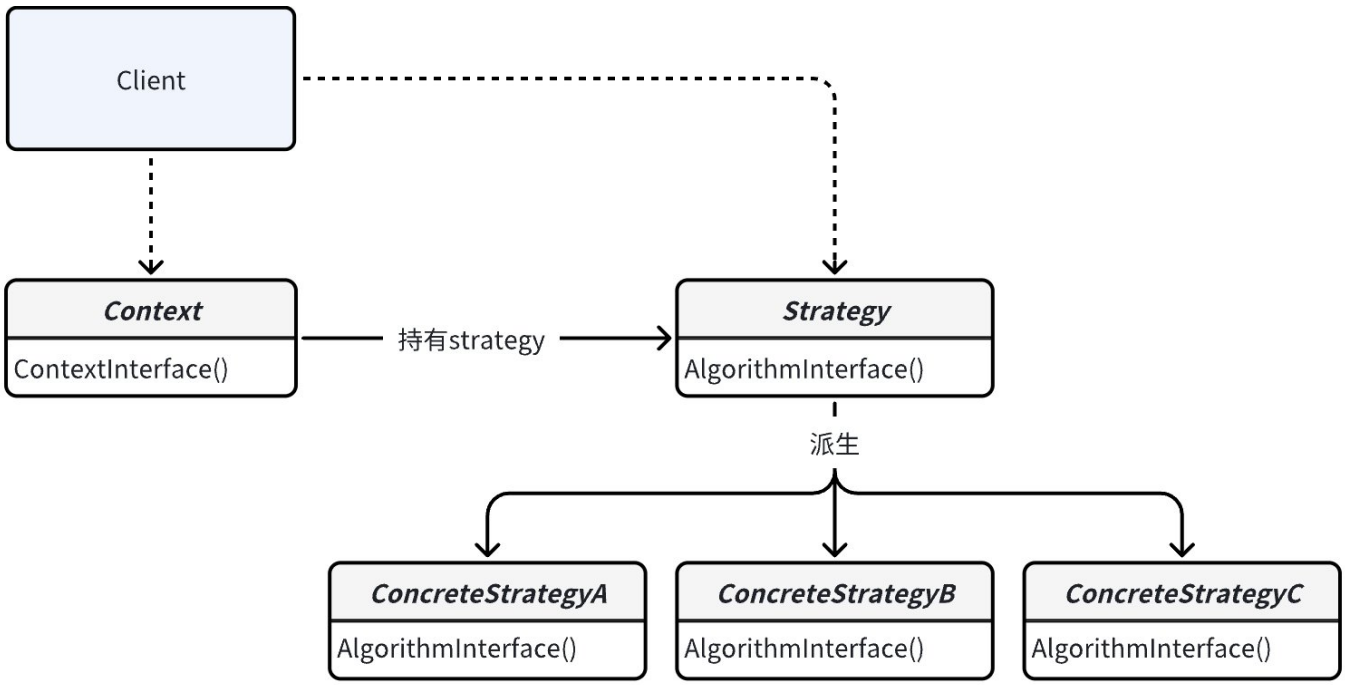
策略模式

概念

- 定义：策略模式是一种行为设计模式，它允许在运行时选择算法的行为。通过将算法封装在独立的策略类中，可以在不修改客户端代码的情况下更改算法。
- 设计意图：策略模式的设计意图是将算法的定义与使用算法的客户类分离开来，使得算法独立于客户类变化。这有助于提高代码的灵活性和可维护性。
- 策略模式适用于以下场景：
  - 当一个类的行为或其算法可以在运行时更改时。
  - 当系统中存在大量类似的类，而它们之间的区别仅在于它们的行为时。
  - 需要避免使用多重条件选择语句（如if-else或switch-case）来决定使用哪种算法时。
- 策略模式主要由以下几个部分构成：
  - 策略接口（Strategy）：定义所有支持的算法的通用接口。
  - 具体策略类（ConcreteStrategy）：实现策略接口，提供具体的算法实现。
  - 上下文类（Context）：持有一个策略接口的引用，并在运行时调用具体策略类的方法。
- 优点
  - 算法可以自由切换：由于策略类都实现了相同的接口，所以它们之间可以自由切换。
  - 易于扩展：增加一个新的策略只需要添加一个具体的策略类即可，基本不需要修改原有代码。
  - 避免使用多重条件选择语句：充分体现了面向对象设计思想。



实例

- 策略接口

```
1  public interface ICalculationStrategy
2  {
3      int Calculate(int value1, int value2);
4  }
```

- 具体策略：加法和减法

```
1  public class AdditionStrategy : ICalculationStrategy
2  {
3      public int Calculate(int value1, int value2)
4      {
5          return value1 + value2;
6      }
7  }
8
9  public class SubtractionStrategy : ICalculationStrategy
10 {
11     public int Calculate(int value1, int value2)
12     {
13         return value1 - value2;
14     }
15 }
```

- 上下文类

```
public class Calculator
{
    private ICalculationStrategy _strategy;

    public Calculator(ICalculationStrategy strategy)
    {
        _strategy = strategy;
    }

    public int ExecuteCalculation(int value1, int value2)
    {
        return _strategy.Calculate(value1, value2);
    }
}
```

- 客户端

```
static void Main(string[] args)
{
    ICalculationStrategy addition = new AdditionStrategy();
    ICalculationStrategy subtraction = new SubtractionStrategy();

    Calculator calculator = new Calculator(addition);
    Console.WriteLine("Addition: " + calculator.ExecuteCalculation(5, 3)); // 输出: 8

    calculator = new Calculator(subtraction);
    Console.WriteLine("Subtraction: " + calculator.ExecuteCalculation(5, 3)); // 输出: 2
}
```