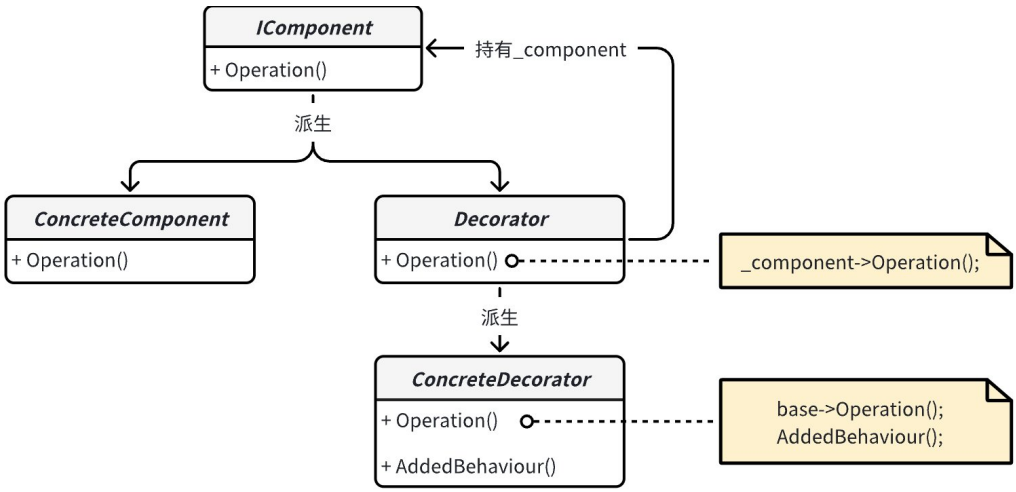
装饰模式

概念

- 装饰模式（Decorator Pattern）是一种结构型设计模式，可以动态地给对象添加新的行为，无需修改其代码。
- 实现：装饰模式通过创建一个装饰类来包装原始类，从而在不改变原始类的情况下扩展其功能。装饰类和原始类具有相同的接口，这样客户端就可以透明地使用装饰类。
- 装饰模式的设计意图：
  - 动态地给对象添加职责。
  - 通过使用多个不同的装饰类，可以在运行时组合出各种不同的行为。
- 装饰模式由以下几个部分构成：
  - 组件接口（Component）：定义对象的接口，可以是抽象类或接口。
  - 具体组件（ConcreteComponent）：实现组件接口的类，表示被装饰的原始对象。
  - 装饰器（Decorator）：实现组件接口的抽象类，包含一个指向组件对象的引用。
  - 具体装饰器（ConcreteDecorator）：继承装饰器类，向组件添加新的行为。
- 装饰模式适用于以下场景：
  - 需要在不修改原始类代码的情况下扩展类的功能。
  - 需要动态地添加或撤销对象的职责。
  - 需要通过组合不同的装饰类来实现复杂的功能。
- 优点
  - 灵活性高：可以动态地添加或撤销对象的职责。
  - 遵循开闭原则：可以在不修改原始类的情况下扩展其功能。
  - 组合性强：可以通过组合多个装饰类来实现复杂的功能。
- 缺点
  - 增加复杂性：由于使用了多个小对象，系统可能会变得复杂。
  - 调试困难：由于装饰器的嵌套，调试时可能难以追踪对象的行为。



实例

- 组件接口

```
1 public interface IComponent
2 {
3     void Operation();
4 }
```

- 具体组件

```
1 public class ConcreteComponent : IComponent
2 {
3     public void Operation()
4     {
5         Console.WriteLine("ConcreteComponent Operation");
6     }
7 }
```

- 装饰器抽象类，继承组件接口并持有组件成员变量

```
1 public abstract class Decorator : IComponent
2 {
3     protected IComponent _component;
4 
5     public Decorator(IComponent component)
6     {
7         _component = component;
```

```
    }

    public virtual void Operation()
    {
        _component.Operation();
    }
}
```

- 具体装饰器

```
// 具体装饰器A
public class ConcreteDecoratorA : Decorator
{
    public ConcreteDecoratorA(IComponent component) : base(component) { }

    public override void Operation()
    {
        base.Operation();
        AddedBehavior();
    }

    void AddedBehavior()
    {
        Console.WriteLine("ConcreteDecoratorA AddedBehavior");
    }
}

// 具体装饰器B
public class ConcreteDecoratorB : Decorator
{
    public ConcreteDecoratorB(IComponent component) : base(component) { }

    public override void Operation()
    {
        base.Operation();
        AddedBehavior();
    }

    void AddedBehavior()
    {
        Console.WriteLine("ConcreteDecoratorB AddedBehavior");
    }
}
```

- 客户端代码

```
static void Main(string[] args)
{
    IComponent component = new ConcreteComponent();
    IComponent decoratorA = new ConcreteDecoratorA(component);
    IComponent decoratorB = new ConcreteDecoratorB(decoratorA);

    decoratorB.Operation();
}
```

- 打印结果

```
ConcreteComponent Operation
ConcreteDecoratorA AddedBehavior
ConcreteDecoratorB AddedBehavior
```