观察者模式

概念

- 定义：观察者模式是一种**行为型模式**，创建了对象间的**一对多**的依赖关系，当一个对象状态改变时，所有依赖于它的对象都会得到通知并自动更新
- 结构：观察者模式包含以下几个核心角色
  - 主题（Subject）：也称为被观察者，它是有状态的对象，并维护着一个观察者列表；主题提供了添加、删除和通知观察者的方法
  - 观察者（Observer）：观察者是接收主题通知的对象；观察者需要实现一个更新方法，当收到主题的通知时，调用该方法进行更新操作
  - 具体主题（Concrete Subject）：具体主题是主题的具体实现类；它维护着观察者列表，并在状态发生改变时通知观察者
  - 具体观察者（Concrete Observer）：具体观察者是观察者的具体实现类；它实现了更新方法，定义了在收到主题通知时需要执行的具体操作
- 实现方式
  - 定义观察者接口：包含一个更新方法
  - 创建具体观察者：实现观察者接口，定义接收到通知时的行为
  - 定义主题接口：包含添加、删除和通知观察者的方法
  - 创建具体主题：实现主题接口，管理观察者列表，并在状态改变时通知它们
- 优点
  - 抽象耦合：观察者和主题之间是抽象耦合的
  - 触发机制：建立了一套状态改变时的触发和通知机制
- 缺点
  - 性能问题：如果观察者众多，通知过程可能耗时
  - 循环依赖：观察者和主题互相依赖，可能导致循环调用

实例

- 观察者接口

```
1 public interface IObserver
2 {
```

```
    void Update();
}
```

- 观察者类

```
public class ConcreteObserver : IObserver
{
    private string _name;
    private ConcreteSubject _subject;

// 构造函数，传入name和关联的subject
public ConcreteObserver(string name, ConcreteSubject subject)
{
    _name = name;
    _subject = subject;
    _subject.Attach(this);
}

public void Update()
{
    Console.WriteLine($"Observer {_name} has been notified. New State:
{_subject.State}");
    }
}
```

- 主题接口

```
public interface ISubject
{
    void Attach(IObserver observer);
    void Detach(IObserver observer);
    void Notify();
}
```

- 主题类

```
public class ConcreteSubject : ISubject
{
    // 维护的观察者列表
    private List<IObserver> _observers = new List<IObserver>();
    private int _state;
```

```
6
7       public int State
8       {
9           get { return _state; }
10          set
11          {
12              _state = value;
13              Notify(); // 设置状态时通知所有观察者
14          }
15      }
16
17      public void Attach(IObserver observer)
18      {
19          _observers.Add(observer);
20      }
21
22      public void Detach(IObserver observer)
23      {
24          _observers.Remove(observer);
25      }
26
27      public void Notify()
28      {
29          foreach (var observer in _observers)
30          {
31              observer.Update();
32          }
33      }
34  }
```

- 使用实例

```
1   class Program
2   {
3       static void Main(string[] args)
4       {
5           // 实例化一个主题对象
6           ConcreteSubject subject = new ConcreteSubject();
7
8           // 实例化两个观察者
9           ConcreteObserver observer1 = new ConcreteObserver("Observer 1",
    subject);
10          ConcreteObserver observer2 = new ConcreteObserver("Observer 2",
    subject);
11
```

```
        subject.State = 1;
        subject.State = 2;

        subject.Detach(observer1);
        subject.State = 3;
    }
}
```

- 打印结果

```
Observer Observer 1 has been notified. New State: 1
Observer Observer 2 has been notified. New State: 1
Observer Observer 1 has been notified. New State: 2
Observer Observer 2 has been notified. New State: 2
Observer Observer 2 has been notified. New State: 3
```