河北师范大学软件学院
Software College of Hebei Normal University

第二章 观察者模式

任课教师：武永亮
wuyongliang@edu2act.org

上节回顾

- 课程主要介绍设计模式和体系结构模式
- 学习方式为环境、问题、解决方案
- 单例模式解决的问题是“独生子女”
- 单例模式的解决方案是利用Static
- 好设计的原则

课程内容

- 环境及问题
- 观察者模式详解
- 观察者模式实现
- 扩展练习

课程内容

- 环境及问题
- 观察者模式详解
- 观察者模式实现
- 扩展练习

环境

- 学期初上课教师做自我介绍公布联系方式
- 学生记录教师联系方式
- 教师联系方式更改时学生更新本地记录的内容

请利用15分钟时间对该系统进行设计

环境

```
class Student
{
    private string _name;

    public string Name
    {
        get { return _name; }
        set { _name = value; }
    }
    private string _tPhone;

    public string TPhone
    {
        get { return _tPhone; }
        set { _tPhone = value; }
    }

    public Student(string name)
    {
        this.Name = name;
    }

    public void show()
    {
        Console.WriteLine("Name:" + Name + "\nTeacher's Phone:" + TPhone);
    }
}
```

```
class Teacher
{
    private string _phone;

    public string Phone
    {
        get { return _phone; }
        set { _phone = value; }
    }

    public Teacher(string phone)
    {
        Phone = phone;
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        Teacher zhangWeizhi = new Teacher("12345");
        Student linLei = new Student("linLei");
        Student lianJie = new Student("lianJie");
        Student wangCongZhao = new Student("wangCongZhao");

        linLei.TPhone = zhangWeizhi.Phone;
        lianJie.TPhone = zhangWeizhi.Phone;
        wangCongZhao.TPhone = zhangWeizhi.Phone;

        linLei.show();
        lianJie.show();
        wangCongZhao.show();

        //电话号码变更
        zhangWeizhi.Phone = "67890";

        linLei.TPhone = zhangWeizhi.Phone;
        lianJie.TPhone = zhangWeizhi.Phone;
        wangCongZhao.TPhone = zhangWeizhi.Phone;

        linLei.show();
        lianJie.show();
        wangCongZhao.show();
    }
}
```

环境

```
class Student
{
    private string _name;

    public string Name
    {
        get { return _name; }
        set { _name = value; }
    }
    private Teacher _teacher;

    internal Teacher Teacher
    {
        get { return _teacher; }
        set { _teacher = value; }
    }

    public Student(string name,Teacher t)
    {
        this.Name = name;
        this.Teacher = t;
    }

    public void show()
    {
        Console.WriteLine("Name:" + Name + "\nTeacher's Phone:" + Teacher.Phone);
    }
}
```

```
class Teacher
{
    private string _phone;

    public string Phone
    {
        get { return _phone; }
        set { _phone = value; }
    }

    public Teacher(string phone)
    {
        Phone = phone;
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        Teacher zhangWeizhi = new Teacher("12345");
        Student linLei = new Student("linLei",zhangWeizhi);
        Student lianJie = new Student("lianJie", zhangWeizhi);
        Student wangCongZhao = new Student("wangCongZhao", zhangWeizhi);

        linLei.show();
        lianJie.show();
        wangCongZhao.show();

        //电话号码变更
        Console.WriteLine("==================");
        Console.WriteLine("电话号码变更为67890");
        zhangWeizhi.Phone = "67890";

        linLei.show();
        lianJie.show();
        wangCongZhao.show();
    }
}
```

问题

- **背景**：某对象发生变化，需其他对象做出调整。
- 应用程序的可维护性和重用性。
- 互动关系不能体现成类之间的直接调用，对象之间关系的解耦。

观察者模式（Observer）

课程内容

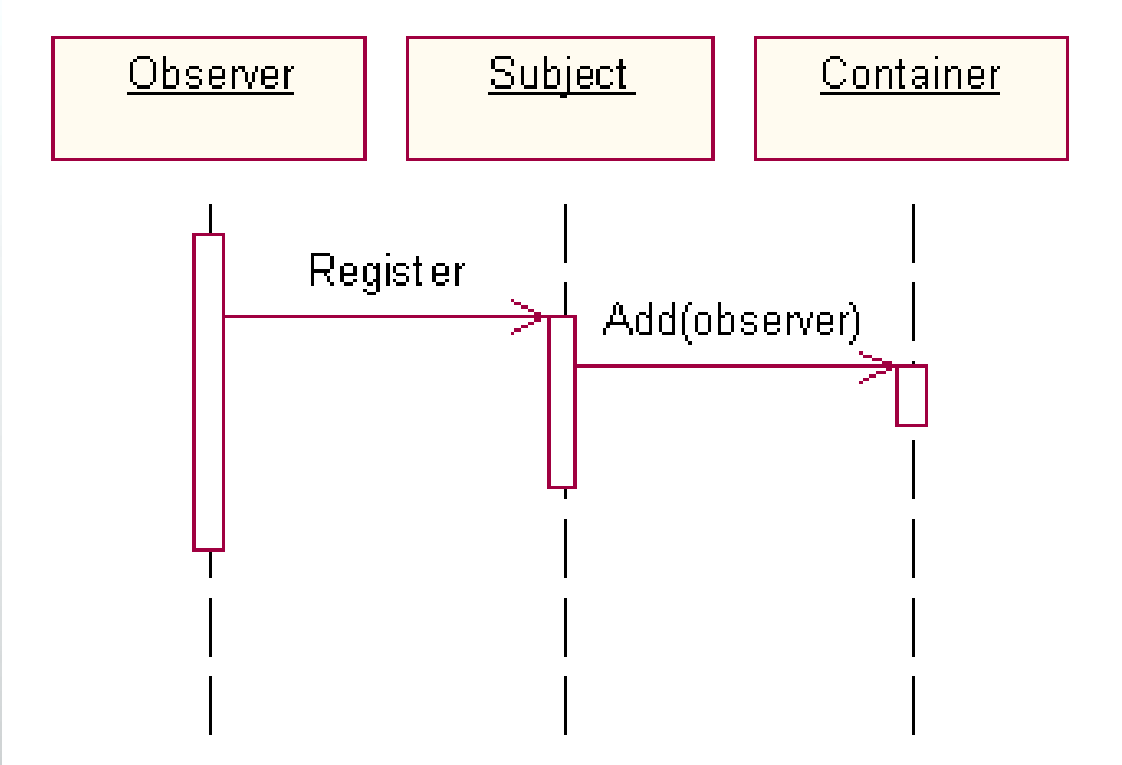
- 环境及问题
- 观察者模式详解
- 观察者模式实现
- 扩展练习

观察者模式（Observer Pattern）

- 又叫发布-订阅模式。
- 两个角色：观察者和被观察对象
- 两者之间存在“观察”的逻辑关联
- 当被观察者发生改变的时候，观察者就会观察到这样的变化，并且做出相应的响应
- “观察”不是“直接调用”
- 实现观察者模式有很多形式，比较直观的一种是使用一种“注册——通知——撤销注册”的形式。

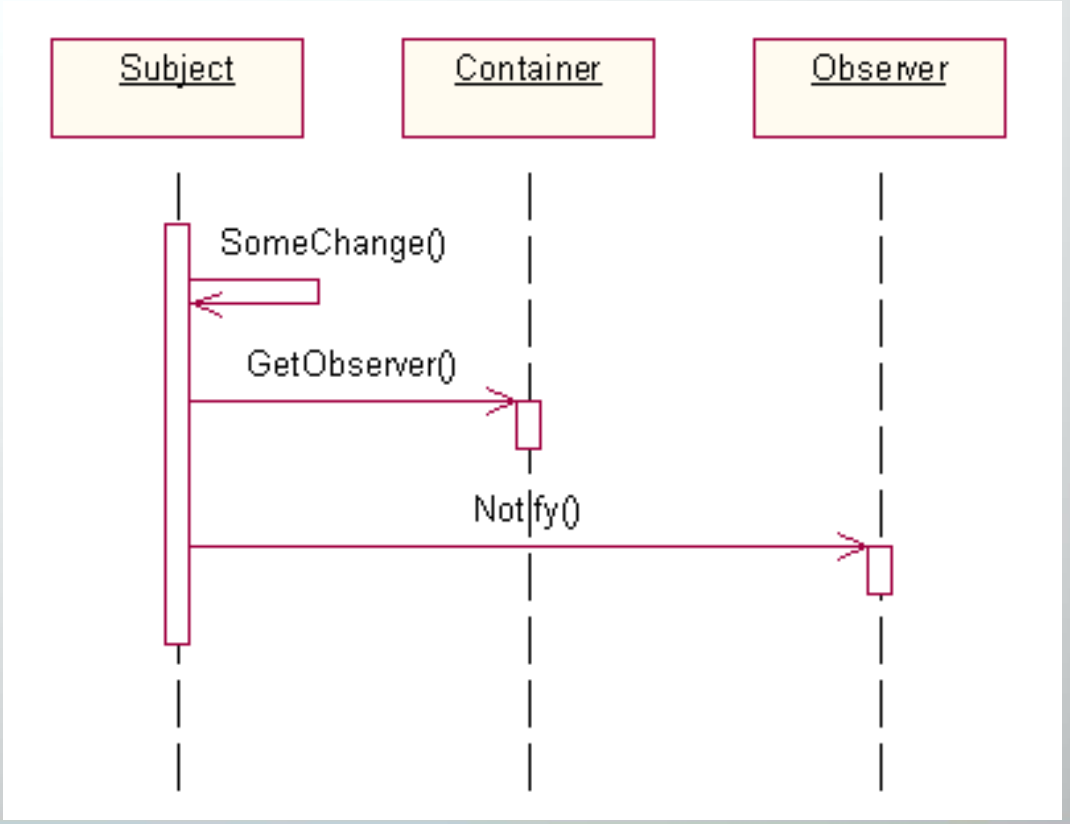
观察者模式实现步骤一

- 观察者将自己注册到被观察对象中，被观察对象将观察者存放在一个容器里



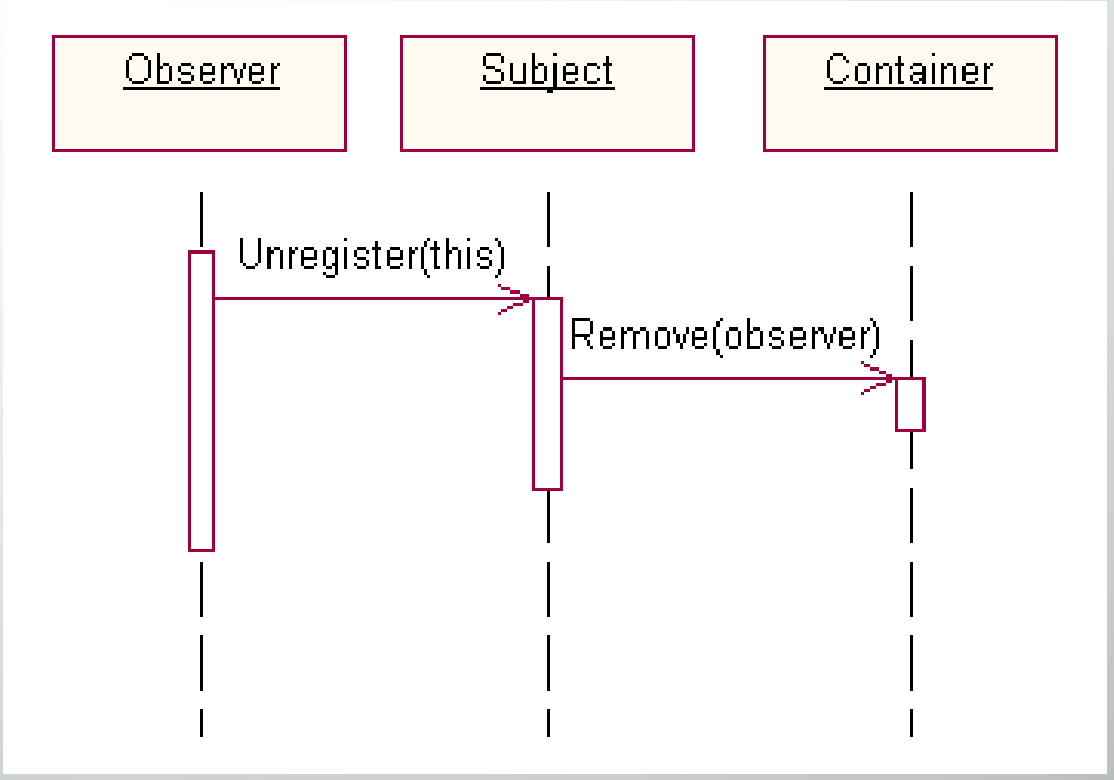
观察者模式实现步骤二

- 被观察对象发生了某种变化，从容器中得到所有注册过的观察者，将变化通知观察者。

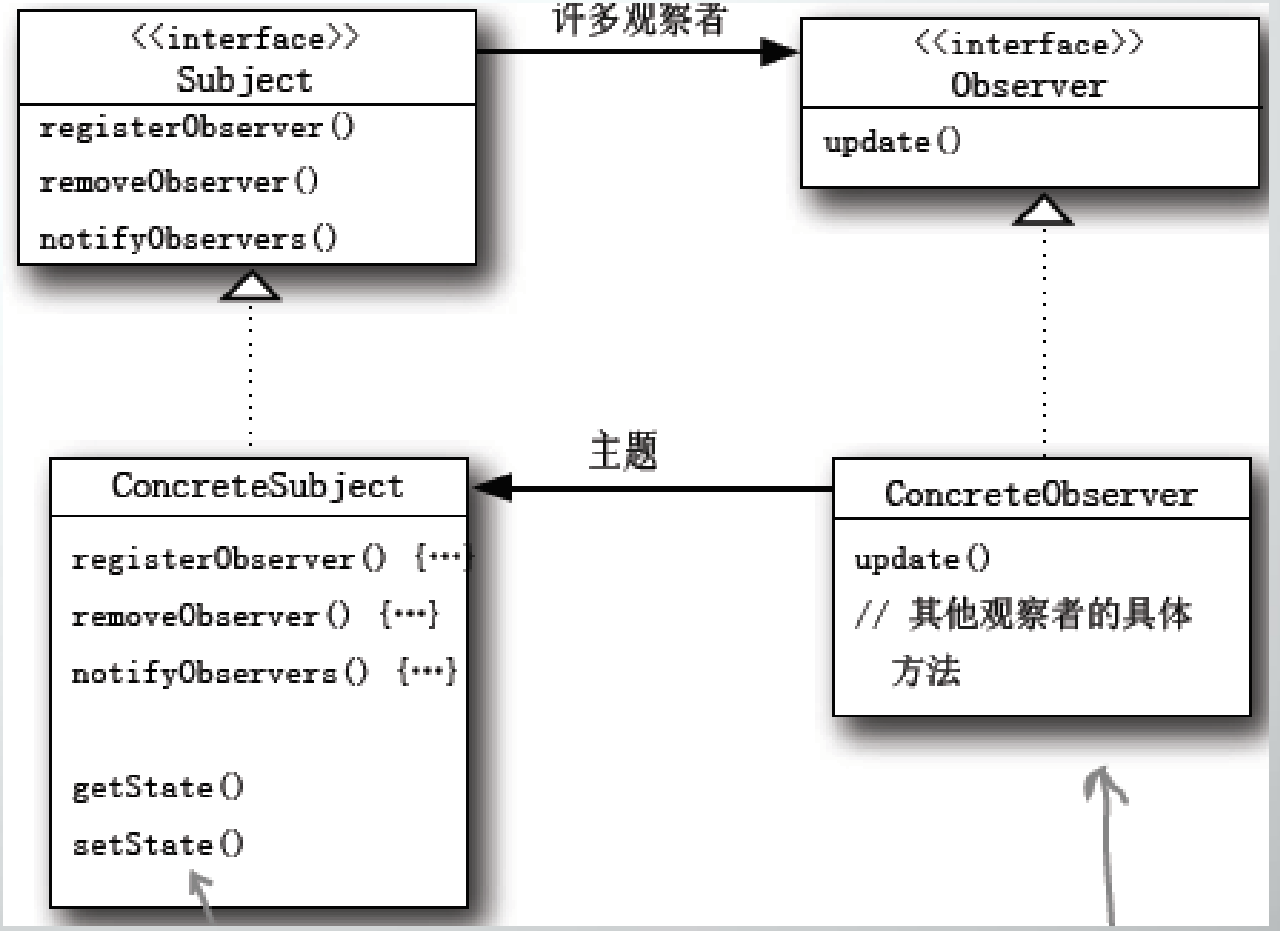


观察者模式实现步骤三

- 观察者告诉被观察者要撤销观察，被观察者从容器中将观察者去除。



观察者模式设计类图


许多观察者
<<interface>>
Subject
registerObserver()
removeObserver()
notifyObservers()
<<interface>>
Observer
update()
主题
ConcreteSubject
registerObserver() {…}
removeObserver() {…}
notifyObservers() {…}
getState()
setState()
ConcreteObserver
update()
// 其他观察者的具体方法

课程内容

- 环境及问题
- 观察者模式详解
- 观察者模式实现
- 扩展练习

观察者模式实现代码

```
interface Subject
{
    void registerObserver(Observer o);
    void removeObserver(Observer o);
    void notifyObserver();
}
```

```
interface Observer
{
    void update(object o);
}
```

```
class Teacher:Subject
{
    private string _phone;
    public string Phone
    {
        get { return _phone; }
        set
        {
            _phone = value;
            notifyObserver();
        }
    }

    public Teacher()
    {
        stuList = new ArrayList();
    }

    private ArrayList stuList;

    public void registerObserver(Observer o)
    {
        stuList.Add(o);
    }
    public void removeObserver(Observer o)
    {
        stuList.Remove(o);
    }
    public void notifyObserver()
    {
        for (int i = 0; i < stuList.Count; i++)
            ((Observer)stuList[i]).update(Phone);
    }
}
```

观察者模式实现代码

```
class Student:Observer
{
    private string _name;

    public string Name
    {
        get { return _name; }
        set { _name = value; }
    }
    private string _tPhone;

    public string TPhone
    {
        get { return _tPhone; }
        set { _tPhone = value; }
    }

    public Student(string name)
    {
        this.Name = name;
    }

    public void update(object o)
    {
        this.TPhone = (string)o;
    }

    public void show()
    {
        Console.WriteLine("Name:" + Name + "\nTeacher's Phone:" + TPhone);
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        Teacher zhangWeizhi = new Teacher();
        Student linLei = new Student("linLei");
        Student lianJie = new Student("lianJie");
        Student wangCongZhao = new Student("wangCongZhao");

        zhangWeizhi.registerObserver(linLei);
        zhangWeizhi.registerObserver(lianJie);
        zhangWeizhi.registerObserver(wangCongZhao);

        zhangWeizhi.Phone = "12345";
        linLei.show();
        lianJie.show();
        wangCongZhao.show();

        //电话号码变更
        Console.WriteLine("====================");
        zhangWeizhi.Phone = "67890";

        zhangWeizhi.removeObserver(wangCongZhao);

        linLei.show();
        lianJie.show();
        wangCongZhao.show();
    }
}
```

课程内容

- 环境及问题
- 观察者模式详解
- 观察者模式实现
- 扩展练习

扩展说明

- 在.NET框架中，使用代理以及事件，可以更好的实现观察者模式。
- 在事件的模式下，声明事件的类就是被观察者。
- IObserver和ISubject接口的方法可以减少观察者和观察对象之间的耦合，而代理和事件几乎消除了这两个模块之间的耦合。

案例练习

- 订阅杂志
- 订阅天气预报

小结

- 当被观察者发生改变的时候，观察者就会观察到这样的变化，并且做出相应的响应
- 实现观察者模式有很多形式，比较直观的一种是使用一种“注册——通知——撤销注册”的形式。
- 减少观察者和观察对象之间的耦合。

Thank You，谢谢！