5. 观察者模式

观察者（Observer）模式的定义

指多个对象间存在一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并被自动更新。这种模式有时又称作发布-订阅模式、模型-视图模式，它是对象行为型模式。

天气预报项目需求,具体要求如下：

1） 气象站可以将每天测量到的温度，湿度，气压等等以公告的形式发布出去(比如发布到自己的网站或第三方)。

2） 需要设计开放型 API，便于其他第三方也能接入气象站获取数据。

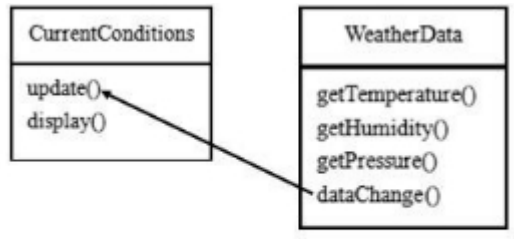
3） 提供温度、气压和湿度的接口

4） 测量数据更新时，要能实时的通知给第三方

天气预报设计方案 1-普通方案

WeatherData 类

传统的设计方案



CurrentConditions(当前的天气情况)
可以理解成是我们气象局的网站 //推送

问题分析

1) 其他第三方接入气象站获取数据的问题

2) 无法在运行时动态的添加第三方（新浪网站）

3) 违反 ocp 原则=>观察者模式

//在 WeatherData 中，当增加一个第三方，都需要创建一个对应的第三方的公告板对象，并加入到 dataChange，不利于维护，也不是动态加入

```
public void dataChange() {
currentConditions.update(getTemperature(), getPressure(),
getHumidity());
}
```

观察者模式原理

1) 观察者模式类似订牛奶业务

2) 奶站/气象局：Subject

3) 用户/第三方网站：Observer

Subject：登记注册、移除和通知

1) registerObserver 注 册

2) removeObserver 移 除

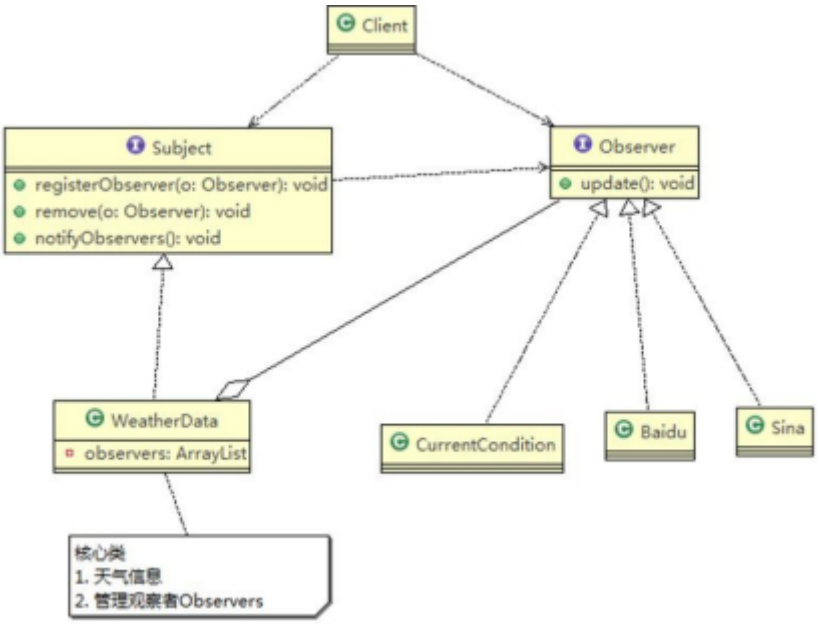
3) notifyObservers() 通知所有的注册的用户，根据不同需求，可以是更新数据，让用户来取，也可能是实施推送， 看具体需求定

Observer：接收输入

观察者模式：对象之间多对一依赖的一种设计方案，并且当一个对象的状态发生变化时，所有依赖于他的对象都想得到通知并更新。被依赖的对象为 Subject，依赖的对象为 Observer，Subject通知 Observer 变化,比如这里的奶站是 Subject，是 1 的一方。用户时 Observer，是多的一方。

观察者模式解决天气预报需求

类图说明




```
//接口, 让WeatherData 来实现
public interface Subject {
   public void registerObserver(Observer o);
   public void removeObserver(Observer o);
   public void notifyObservers();
}


/**
 * 类是核心
 * 1. 包含最新的天气情况信息
 * 2. 含有 观察者集合，使用ArrayList管理
 * 3. 当数据有更新时，就主动的调用  ArrayList, 通知所有的（接入方）就看到最新的信息
 * @author Administrator
 *
 */
public class WeatherData implements Subject {
   private float temperatrue;
   private float pressure;
   private float humidity;
   //观察者集合
   private ArrayList<Observer> observers;

   //加入新的第三方

   public WeatherData() {
      observers = new ArrayList<Observer>();
   }

   public float getTemperature() {
      return temperatrue;
   }
```

```
public float getPressure() {
  return pressure;
}

public float getHumidity() {
  return humidity;
}

public void dataChange() {
  //调用 接入方的 update

  notifyObservers();
}

//当数据有更新时，就调用 setData
public void setData(float temperature, float pressure, float humidity) {
  this.temperatrue = temperature;
  this.pressure = pressure;
  this.humidity = humidity;
  //调用dataChange， 将最新的信息 推送给 接入方 currentConditions
  dataChange();
}

//注册一个观察者
@Override
public void registerObserver(Observer o) {
  // TODO Auto-generated method stub
  observers.add(o);
}

//移除一个观察者
@Override
public void removeObserver(Observer o) {
  // TODO Auto-generated method stub
  if(observers.contains(o)) {
    observers.remove(o);
  }
}

//遍历所有的观察者，并通知
@Override
public void notifyObservers() {
  // TODO Auto-generated method stub
  for(int i = 0; i < observers.size(); i++) {
    observers.get(i).update(this.temperatrue, this.pressure, this.humidity);
  }
```

```
  }
}
//观察者接口，有观察者来实现
public interface Observer {
  public void update(float temperature, float pressure, float humidity);
}


public class CurrentConditions implements Observer {

  // 温度，气压，湿度
  private float temperature;
  private float pressure;
  private float humidity;

  // 更新 天气情况，是由 WeatherData 来调用，我使用推送模式
  public void update(float temperature, float pressure, float humidity) {
    this.temperature = temperature;
    this.pressure = pressure;
    this.humidity = humidity;
    display();
  }

  // 显示
  public void display() {
    System.out.println("***Today mTemperature: " + temperature + "***");
    System.out.println("***Today mPressure: " + pressure + "***");
    System.out.println("***Today mHumidity: " + humidity + "***");
  }
}
```

观察者模式的好处

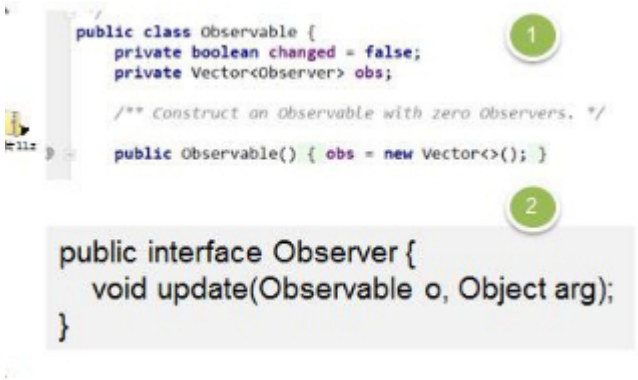
1） 观察者模式设计后，会以集合的方式来管理用户(Observer)，包括注册，移除和通知。

2） 这样，我们增加观察者(这里可以理解成一个新的公告板)，就不需要去修改核心类 WeatherData 不会修改代码， 遵守了 ocp 原则。

观察者模式在 Jdk 应用的源码分析

1） Jdk 的 Observable 类就使用了观察者模式

2） 代码分析+模式角色分析



1） 模式角色分析

Observable 的作用和地位等价于 我们前面讲过 Subject

Observable 是类，不是接口，类中已经实现了核心的方法 ,即管理 Observer 的方 法 add.. delete .. notify...

Observer 的作用和地位等价于我们前面讲过的 Observer, 有 update

Observable 和 Observer 的使用方法和前面讲过的一样，只是 Observable 是类，通过继承来实现观察者模式

观察者模式是一种对象行为型模式，其主要优点如下：

1. 降低了目标与观察者之间的耦合关系，两者之间是抽象耦合关系。
2. 目标与观察者之间建立了一套触发机制。

它的主要缺点如下：

1. 目标与观察者之间的依赖关系并没有完全解除，而且有可能出现循环引用。
2. 当观察者对象很多时，通知的发布会花费很多时间，影响程序的效率。