链滴

设计模式 |02 观察者模式

作者：qq692310342
原文链接：https://ld246.com/article/1567432696188
来源网站：链滴

开头说几句

博主的博客地址： https://www.jeffcc.top/

博主学习设计模式用的书是Head First的《设计模式》，强烈推荐配套使用！

什么是观察者模式

1. 观察者模式定义了对象之间的一对多依赖，这样一来，当一个对象改变状态的时候，它的所有依赖会接受到通知并且自动更新。

2. 该模式类似于 报纸杂志的出版和订阅，需要订阅者先订阅杂志，才能够继续进行后续的杂志更新时推送给订阅者，这里的出版社就是对应着观察者模式的主题，订阅者对应着观察者。

3. 观察者模式在我们的生活中随处可见，并且是JDK 中使用最多的设计模式之一了，同时在消息队列也有相关的应用！

设计原则

1. 为了交互对象这件的松耦合设计而努力。当两个对象之间松耦合，他们依然可以交互，但是不太清彼此的细节。

2. 关于观察者的一切，主题只需要知道观察者实现了某个接口，主题不需要知道观察者的具体类是谁做了什么。

3. 当有新的观察者出现的时候，主题不需要修改代码，主题并不在乎这个问题，它只会发送通知给所实现了观察者接口的对象。

4. 找出程序中会发生变化的方面，然后将其他和固定不变的方面分离。在观察者模式中，会改变的是题的状态以及观察者的数量和类型，所以这些方面可以单独分离出来作为一个接口。

5. 针对接口编程，主题与观察者都使用了接口，观察者利用主题的接口向主题进行注册，而主题利用察者的接口进行发送消息。

6. 多用组合少用继承，观察者模式中使用了组合来将许多的观察者组合进入主题中，并不是使用继承实现的。

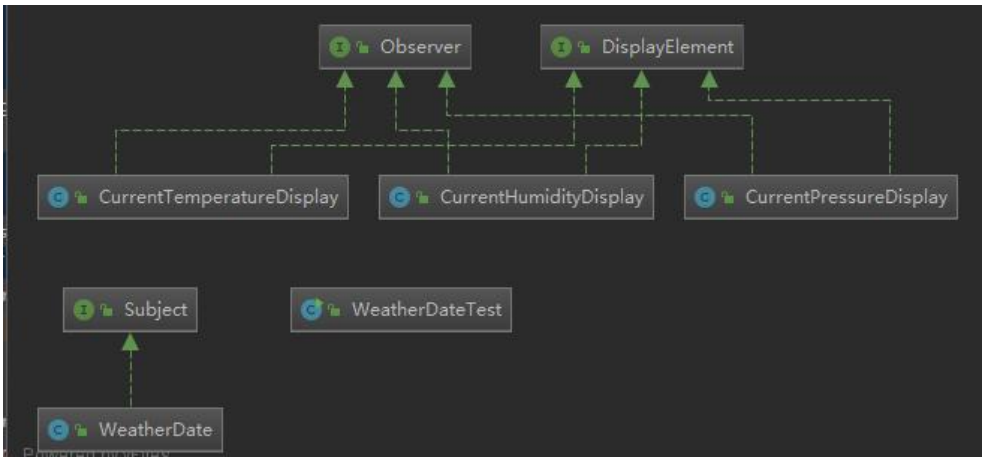
模式实例

设计背景

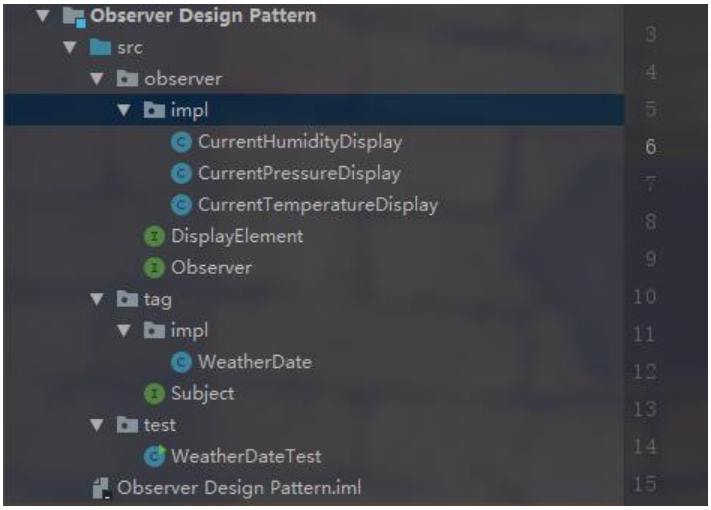
1. 有一个气象站WeatherDate 专门负责收集气象数据（温度，湿度，气压），并且实时向三块布告来传输数据显示（温度，湿度，气压），并且要求能够随时增加删除布告板。

设计代码

项目类图



项目结构



气象站超类

```
package tag;

import observer.Observer;
```

```
/**
 * 主题需要实现的接口规范
 */
public interface Subject {
//    注册观察者
    public void registerObserver(Observer observer);
//    注销观察者
    public void removeObserver(Observer observer);
//    主题发生变化的时候 通知在注册的观察者
    public void notifyObserver();
}
```

气象站实现类

```
package tag.impl;

import observer.Observer;
import tag.Subject;

import java.util.ArrayList;

/**
 * 主题的实现类
 */
public class WeatherDate implements Subject {
//    用于存储观察者的集合
    private ArrayList<Observer> observers;
    private float temperature;
    private float humidity;
    private float pressure;

    /**
     * 初始化集合！ 很重要!
     */
    public WeatherDate() {
        this.observers = new ArrayList<Observer>();
    }

    /**
     * 注册观察者
     * @param observer
     */
    @Override
    public void registerObserver(Observer observer) {
        observers.add(observer);
    }

    /**
     *注销观察者
     * @param observer
     */
    @Override
```

```
    public void removeObserver(Observer observer) {
//        注意防止非空
        if(observers.indexOf(observer)>0){
            observers.remove(observer);
        }
    }

    /**
     * 唤醒观察者
     */
    @Override
    public void notifyObserver() {
        observers.forEach(observer -> {
            observer.update(temperature,humidity,pressure);
        });
    }

    /**
     * 通知观察者
     */
    public void measurementsChanged(){
        notifyObserver();
    }

    /**
     * 模拟气象站抓取数据
     * @param temperature
     * @param humidity
     * @param pressure
     */
    public void setMeasurements(float temperature, float humidity, float pressure){
        this.temperature = temperature;
        this.humidity = humidity;
        this.pressure = pressure;
        measurementsChanged();
    }
}
```

观察者超类

```
package observer;

/**
 * 观察者需要实现的接口
 */
public interface Observer {

    /**
     * 实时更新的方法
     * @param temp 温度
     * @param humidity 湿度
     * @param pressure 气压
     */
```

```
    public void update(float temp, float humidity,float pressure);
}
```

观察者布告板超类

```
package observer;

/**
 * 布告板需要实现的类
 */
public interface DisplayElement {
    /**
     * 显示数据的方法
     */
    public void display();
}
```

三个观察者

```
package observer.impl;

import observer.DisplayElement;
import observer.Observer;
import tag.Subject;

/**
 * 第一个观察者
 * 实时更新温度情况
 */
public class CurrentTemperatureDisplay implements Observer, DisplayElement {

    private float temperature;
    private Subject weatherDate;

    /**
     * 需要一个主题用来注册
     * @param weatherDate
     */
    public CurrentTemperatureDisplay(Subject weatherDate) {
        this.weatherDate = weatherDate;
        weatherDate.registerObserver(this);
    }

    /**
     * 实时发布信息
     */
    @Override
    public void display() {
        System.out.print("一号布告板：");
        System.out.println("temperature:"+temperature);
    }
```

```
    /**
     * 布告板更新
     * @param temp     温度
     * @param humidity 湿度
     * @param pressure 气压
     */
    @Override
    public void update(float temp, float humidity, float pressure) {
        this.temperature = temp;
        display();
    }
}


package observer.impl;

import observer.DisplayElement;
import observer.Observer;
import tag.Subject;

/**
 * 第二个观察者
 * 实时更新湿度情况
 */
public class CurrentHumidityDisplay implements Observer, DisplayElement {

    private float humidity;
    private Subject weatherDate;

    /**
     * 需要一个主题用来注册
     * @param weatherDate
     */
    public CurrentHumidityDisplay(Subject weatherDate) {
        this.weatherDate = weatherDate;
        weatherDate.registerObserver(this);
    }

    /**
     * 实时发布信息
     */
    @Override
    public void display() {
        System.out.print("二号布告板：");
        System.out.println("humidity:"+humidity);
    }

    /**
     * 布告板更新
     * @param temp     温度
     * @param humidity 湿度
     * @param pressure 气压
     */
```

```
    @Override
    public void update(float temp, float humidity, float pressure) {
        this.humidity = humidity;
        display();
    }
}


package observer.impl;

import observer.DisplayElement;
import observer.Observer;
import tag.Subject;

/**
 * 第三个观察者
 * 实时更新气压情况
 */
public class CurrentPressureDisplay implements Observer, DisplayElement {

    private float pressure;
    private Subject weatherDate;

    /**
     * 需要一个主题用来注册
     * @param weatherDate
     */
    public CurrentPressureDisplay(Subject weatherDate) {
        this.weatherDate = weatherDate;
        weatherDate.registerObserver(this);
    }

    /**
     * 实时发布信息
     */
    @Override
    public void display() {
        System.out.print("三号布告板：");
        System.out.println("pressure:"+pressure);
    }

    /**
     * 布告板更新
     * @param temp     温度
     * @param humidity 湿度
     * @param pressure 气压
     */
    @Override
    public void update(float temp, float humidity, float pressure) {
        this.pressure = pressure;
        display();
    }
}
```

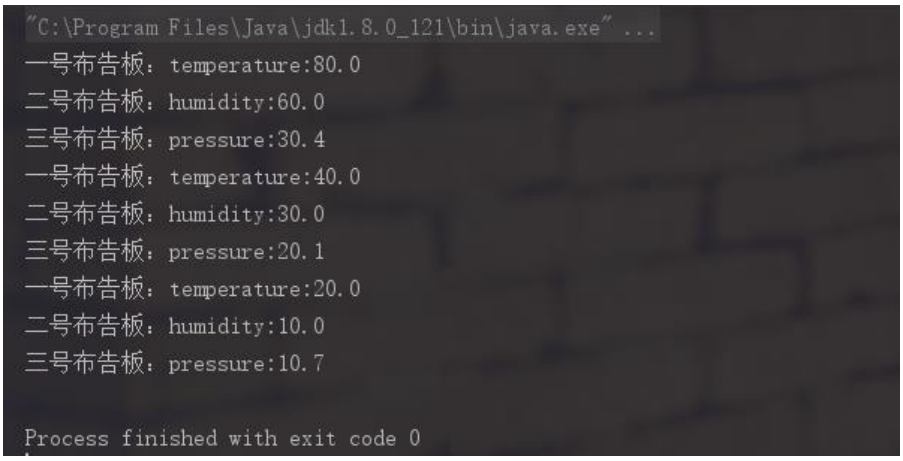
气象站测设类

```
package test;

import observer.impl.CurrentHumidityDisplay;
import observer.impl.CurrentPressureDisplay;
import observer.impl.CurrentTemperatureDisplay;
import tag.impl.WeatherDate;

/**
 * 气象站测试
 */
public class WeatherDateTest {
    public static void main(String[] args) {
//        新建一个气象站
        WeatherDate weatherDate = new WeatherDate();
//        注册三个布告板
        new CurrentTemperatureDisplay(weatherDate);
        new CurrentHumidityDisplay(weatherDate);
        new CurrentPressureDisplay(weatherDate);
//        模拟气象站发布新气象
        weatherDate.setMeasurements(80,60,30.4f);
        weatherDate.setMeasurements(40,30,20.1f);
        weatherDate.setMeasurements(20,10,10.7f);
    }
}
```

输出结果



---END

2019年9月2日21:56:39~~~~