# 观察者模式：一对多的依赖

观察者模式定义了对象之间一对多的依赖，当主题改变状态时，它的所有的依赖——观察者都会收到通知并自动更新。

出版者 + 订阅者 = 观察者模式

主题是具有状态的对象，并且可以控制这些状态。观察者使用这些状态，虽然这些状态不属于它们。有许多的观察者，依赖主题来告诉它们状态何时改变了。

主题是真正拥有数据的人，观察者是主题的依赖者，在数据变化时更新，比起让许多对象控制同一份数据来说，可以得到更干净的 OO 设计。

此系统必须可扩展，我们希望用户可以随心所欲地添加或删除任何布告板。

观察者为了取得信息，必须向主题对象注册。

## 用到的 OO 原则

* 封装变化
* 多用组合，少用继承
* 针对接口编程，不针对实现编程
* 为交互对象之间的松耦合设计而努力

## 目录

• 错误示范

• 松耦合

• 主题接口

• 观察者接口

• 布告板接口

• 实现一个布告板

• 其他布告板

• 客户端程序

• 增加一个布告板

## 错误示范

public class WeatherData {

// 实例变量声明

// 当新的数据备妥时，measurementsChanged() 方法就会被调用

public void measurementsChanged(){

float temp = getTemprature();

float humidity = getHumidity();

float pressure = getPressure();

**// 更新三个布告板；针对具体实现编程，会导致我们以后在增加或删除布告板时必须修改程序**

currentConditionsDisplay.update(temp, humidity, pressure);

statisticsDisplay.update(temp, humidity, pressure);

forecastDisplay.update(temp, humidity, pressure);

// 其他布告板更新数据

}

// 这里是其他 WeatherData 方法

}

更新三个布告板的代码中，三个 update 方法参数相同，看起来更像一个统一的接口，所以不如使三个布告板都实现一个相同的接口，而后用遍历的方式调用每个布告板的 update 方法：

for (Observer observer : observers) {

observer.update(temperature, humidity, pressure);

}

这样做的好处是，做到了针对接口编程，如果针对具体实现编程，将来在增加或删除布告板时就必须修改代码。

## 松耦合

当两个对象之间松耦合，它们依然可以交互，但是不太清楚彼此的细节。

观察者模式让主题和观察者之间松耦合：对于观察者的一切，主题只知其实现了某个接口（即 Observer 接口），主题不需要知道观察者的其他任何细节。主题依赖一个实现了 Observer 接口的对象列表，使我们可以随时增加观察者，当有新类型的观察者出现时，主题代码不需要修改以兼容它，所要做的就是新的观察者在自己的代码中将自己注册到观察者列表中。主题不在乎别的，它只会发送通知给所有实现了观察者接口的对象。

我们可以独立地复用主题或观察者，在其他地方随意使用它们，因为二者并非紧耦合。

改变主题或观察者，并不会影响另一方，因为二者是松耦合，只要它们之间的接口仍被遵守，就可以自由地改变它们。

## 气象站

observer 
observers 
impls 
u:] CurrentConditionsDisplayjava 
ForecastDisplay.java 
u:] HeatlndexDisplayjava 
StatisticsDisplayjava 
DisplayElementjava 
Observerjava 
su bject 
impls 
u:] WeatherDatajava 
Subject.java 
u:] SwingObserverExamplejava 
> u] WeatherStationjava 
L] WeatherStationHeatIndexjava 

主题接口

import com.feng.dp.observer.observers.Observer;

public interface Subject {

void registerObserver(Observer o); // 注册一个观察者

void removeObserver(Observer o); // 删除一个观察者

void notifyObservers(); // 主题改变时，调用该方法以通知所有观察者

}

观察者接口

public interface Observer {

void update(float temperature, float humidity, float pressure);

}

布告板接口

public interface DisplayElement {

void display();

}

WeatherData 实现主题接口

package com.feng.dp.observer.subject.impls;

import java.util.ArrayList;

import java.util.List;

import com.feng.dp.observer.observers.Observer;

import com.feng.dp.observer.subject.Subject;

public class WeatherData implements Subject {

private List<Observer> observers; // 观察者列表

private float temperature; // 温度

private float humidity; // 湿度

private float pressure; // 气压

public WeatherData() {

observers = new ArrayList<Observer>();

}

@Override

public void registerObserver(Observer o) {

observers.add(o);

}

@Override

public void removeObserver(Observer o) {

int i = observers.indexOf(o);

if (i > 0) {

observers.remove(i);

}

}

@Override

public void notifyObservers() {

// 所有观察者具有相同的接口，所以可以遍历、以同样的方式通知它们

for (Observer observer : observers) {

observer.update(temperature, humidity, pressure);

}

}

/\*\*

\* 保存新的测量结果，并通知所有观察者

\* @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();

}

/\*\*

\* 当新的数据备妥时，measurementsChanged() 方法就会被调用

\*/

private void measurementsChanged() {

notifyObservers(); // 通知所有观察者

}

}

实现一个布告板

布告板为了取得信息，必须向 WeatherData 对象注册。

CurrentConditionsDisplay

package com.feng.dp.observer.observers.impls;

import com.feng.dp.observer.observers.DisplayElement;

import com.feng.dp.observer.observers.Observer;

import com.feng.dp.observer.subject.Subject;

public class CurrentConditionsDisplay implements Observer, DisplayElement {

private float temperature;

private float humidity;

private Subject weatherData; // 构造器需要 Subject 对象作为注册之用

public CurrentConditionsDisplay(Subject weatherData) {

this.weatherData = weatherData;

weatherData.registerObserver(this);

}

@Override

public void update(float temperature, float humidity, float pressure) {

// 先把 temperature 和 humidity 保存起来，然后调用 display()

this.temperature = temperature;

this.humidity = humidity;

display();

}

@Override

public void display() {

System.out.println("Current conditions: " + temperature + "F degrees and " + humidity + "% humidity");

}

}

其他布告板

ForecastDisplay

package com.feng.dp.observer.observers.impls;

import com.feng.dp.observer.observers.DisplayElement;

import com.feng.dp.observer.observers.Observer;

import com.feng.dp.observer.subject.Subject;

import com.feng.dp.observer.subject.impls.WeatherData;

public class ForecastDisplay implements Observer, DisplayElement {

private float currentPressure = 29.92f;

private float lastPressure;

private Subject weatherData;

public ForecastDisplay(WeatherData weatherData) {

this.weatherData = weatherData;

weatherData.registerObserver(this);

}

@Override

public void update(float temp, float humidity, float pressure) {

lastPressure = currentPressure;

currentPressure = pressure;

display();

}

@Override

public void display() {

System.out.print("Forecast: ");

if (currentPressure > lastPressure) {

System.out.println("Improving weather on the way!");

} else if (currentPressure == lastPressure) {

System.out.println("More of the same");

} else if (currentPressure < lastPressure) {

System.out.println("Watch out for cooler, rainy weather");

}

}

}

StatisticsDisplay

package com.feng.dp.observer.observers.impls;

import com.feng.dp.observer.observers.DisplayElement;

import com.feng.dp.observer.observers.Observer;

import com.feng.dp.observer.subject.Subject;

import com.feng.dp.observer.subject.impls.WeatherData;

public class StatisticsDisplay implements Observer, DisplayElement {

private float maxTemp = 0.0f;

private float minTemp = 200;

private float tempSum = 0.0f;

private int numReadings;

private Subject weatherData;

public StatisticsDisplay(WeatherData weatherData) {

this.weatherData = weatherData;

weatherData.registerObserver(this);

}

@Override

public void update(float temp, float humidity, float pressure) {

tempSum += temp;

numReadings++;

if (temp > maxTemp) {

maxTemp = temp;

}

if (temp < minTemp) {

minTemp = temp;

}

display();

}

@Override

public void display() {

System.out.println("Avg/Max/Min temperature = " + (tempSum / numReadings) + "/" + maxTemp + "/" + minTemp);

}

}

客户端程序

package com.feng.dp.observer;

import com.feng.dp.observer.observers.impls.CurrentConditionsDisplay;

import com.feng.dp.observer.observers.impls.ForecastDisplay;

import com.feng.dp.observer.observers.impls.StatisticsDisplay;

import com.feng.dp.observer.subject.impls.WeatherData;

public class WeatherStation {

public static void main(String[] args) {

WeatherData weatherData = new WeatherData();

CurrentConditionsDisplay currentDisplay = new CurrentConditionsDisplay(weatherData);

StatisticsDisplay statisticsDisplay = new StatisticsDisplay(weatherData);

ForecastDisplay forecastDisplay = new ForecastDisplay(weatherData);

weatherData.setMeasurements(80, 65, 30.4f);

weatherData.setMeasurements(82, 70, 29.2f);

weatherData.setMeasurements(78, 90, 29.2f);

}

}

试想：把观测值直接传入观察者中是最直接的方法。你认为这样做明智吗？

暗示：这些观测值的种类和个数在未来可能改变吗？如果以后改变，这些变化是否被很好的封装？或者是需要修改的多少代码才能办到？

输出：

Current conditions: 80.0F degrees and 65.0% humidity

Avg/Max/Min temperature = 80.0/80.0/80.0

Forecast: Improving weather on the way!

Current conditions: 82.0F degrees and 70.0% humidity

Avg/Max/Min temperature = 81.0/82.0/80.0

Forecast: Watch out for cooler, rainy weather

Current conditions: 78.0F degrees and 90.0% humidity

Avg/Max/Min temperature = 80.0/82.0/78.0

Forecast: More of the same

新的布告板

HeatIndexDisplay

package com.feng.dp.observer.observers.impls;

import com.feng.dp.observer.observers.DisplayElement;

import com.feng.dp.observer.observers.Observer;

import com.feng.dp.observer.subject.Subject;

import com.feng.dp.observer.subject.impls.WeatherData;

public class HeatIndexDisplay implements Observer, DisplayElement {

float heatIndex = 0.0f;

private Subject weatherData;

public HeatIndexDisplay(WeatherData weatherData) {

this.weatherData = weatherData;

weatherData.registerObserver(this);

}

@Override

public void update(float t, float rh, float pressure) {

heatIndex = computeHeatIndex(t, rh);

display();

}

private float computeHeatIndex(float t, float rh) {

float index = (float) ((16.923 + (0.185212 \* t) + (5.37941 \* rh) - (0.100254 \* t \* rh) + (0.00941695 \* (t \* t)) + (0.00728898 \* (rh \* rh)) + (0.000345372 \* (t \* t \* rh)) - (0.000814971 \* (t \* rh \* rh)) + (0.0000102102 \* (t \* t \* rh \* rh)) - 0.000038646 \* (t \* t \* t)) + (0.0000291583 \* (rh \* rh \* rh)) + (0.00000142721 \* (t \* t \* t \* rh)) + (0.000000197483 \* (t \* rh \* rh \* rh)) - (0.0000000218429 \* (t \* t \* t \* rh \* rh)) + 0.000000000843296 \* (t \* t \* rh \* rh \* rh)) - (0.0000000000481975 \* (t \* t \* t \* rh \* rh \* rh)));

return index;

}

@Override

public void display() {

System.out.println("Heat index is " + heatIndex);

}

}

对于客户端程序，只需增加一条语句：

HeatIndexDisplay heatIndexDisplay = new HeatIndexDisplay(weatherData);

输出：

Current conditions: 80.0F degrees and 65.0% humidity

Avg/Max/Min temperature = 80.0/80.0/80.0

Forecast: Improving weather on the way!

Heat index is 82.95535

Current conditions: 82.0F degrees and 70.0% humidity

Avg/Max/Min temperature = 81.0/82.0/80.0

Forecast: Watch out for cooler, rainy weather

Heat index is 86.90124

Current conditions: 78.0F degrees and 90.0% humidity

Avg/Max/Min temperature = 80.0/82.0/78.0

Forecast: More of the same

Heat index is 83.64967

## JDK 中的观察者模式

建立一个 JButton 对象，把它加到 Jframe，然后设置好倾听者就行了。

import java.awt.\*;

import javax.swing.\*;

public class SwingObserverExample {

JFrame frame;

public static void main(String[] args) {

SwingObserverExample example = new SwingObserverExample();

example.go();

}

public void go() {

frame = new JFrame();

JButton button = new JButton("Should I do it?");

// Without lambdas

// button.addActionListener(new AngelListener());

// button.addActionListener(new DevilListener());

// With lambdas

button.addActionListener(

event -> System.*out*.println("Don't do it, you might regret it!"));

button.addActionListener(event -> System.*out*.println("Come on, do it!"));

frame.getContentPane().add(BorderLayout.*CENTER*, button);

// Set frame properties

frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);

frame.getContentPane().add(BorderLayout.*CENTER*, button);

frame.setSize(300, 300);

frame.setVisible(true);

}

/\*

\* Remove these two inner classes to use lambda expressions instead.

\*

class AngelListener implements ActionListener {

public void actionPerformed(ActionEvent event) {

System.out.println("Don't do it, you might regret it!");

}

}

class DevilListener implements ActionListener {

public void actionPerformed(ActionEvent event) {

System.out.println("Come on, do it!");

}

}

\*/

}