观察者模式

前言:

> 作者：韩数
>
> 代码下载(欢迎star)：Github：https://github.com/hanshuaikang
>
> 时间:2019-01-29
>
> JDK版本：1.8
>
> 文章版本：1.0

定义:

> 定义对象间的一种一对多的依赖关系,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。

适用范围:

> 一个对象状态改变给其他对象通知的问题，而且要考虑到易用和低耦合，保证高度的协作。（例:像我前段时间看的一本书《羊的门》，每当村支书做出什么决定时，就通过村子里的那个大喇叭广播给所有村民，让他们去执行。拍卖会上，拍卖师观察下面谁出的价格更高，然后通知现场所有参与拍卖的人）

优/缺点:

> **优点:**
>
> 1.因为广播者和观察者代码之间是低耦合的，比较灵活，弹性可扩展性比较高(后续 可以动态的添加新的观察者，而广播者几乎不受影响)。
>
> 2.可以建立一套触发机制，比如上例中村支书广播的时候，可以灵活地设定通知阈值，比如只有大事的时候才通知等等。
>
> **缺点:**
>
> 1.比如需要通知的观察者太多，完整的一个流程下来就可能效率比较低，比较浪费时间
>
> 2.如果在观察者和广播者之间有循环依赖的话，观察目标会触发它们之间进行循环调用，可能导致系统崩溃。
>
> 3.因为广播者对观察者数据改变的过程是隐藏的，所有观察者并不知道广播者是如何工作的。

前提引入:

经历过上一次被自己弟弟二呆抢尽风头之后，阿呆一直怀恨在心，但公司财务状况堪忧，之前老板巨资投入研发的中美合作大型古装玄幻爱情伦理3D手游《两开花》距离上线遥遥无期，更不要说盈利了，前几天还听说老板准备开猿节流呢，阿呆想着，这段时间可一定得好好表现，公司就自己一个程序猿，开猿节流不就是明显针对自己吗。

山重水复疑无路,柳暗花明又一村

前几天老板回来，一改往日颓势，欣喜之情溢于言表，刚到公司，就把阿呆叫到了自己办公室。

老板:阿呆啊，你来公司一年多了吧，这一年多，公司业绩没上去，你王者段位上去了不少，当然这也不怪你，公司本来也就没什么业绩，不过！现在不一样了，我刚拿到一个大客户，气象台的，这活儿要是做成了，你放心，上一年的工资我肯定给你发了！你可不要辜负我对你的期望啊。

阿呆：(内心戏：没想到老板拿到项目还记得给我发上一年的工资，真是感动，这么好的老板我上哪找去，大哭，跟老板到天涯海角，老板跑路我陪你跑)，保证完成任务，老板那需求是什么能给我说说吗?

老板：需求是这样的，人家气象台呢，觉得按照之前那么通知太费电话费了，现在不是兴打造互联网平台吗，人家也想搞一个，就是人家源源不断的从地面监测站传感器读数据，然后呢，当数据有改变时，通知到气象局开发的APP，网站，小程序上实时更新天气信息，同时呢，人家是做平台，得有一组API对外提供，方便其他的合作伙伴接进来，需求就是这么个需求，我看不就写个程序吗，简单，你好好给人家做做！

阿呆：(内心戏：这个，这个，哪里简单了？你做个试试)，没问题老板，保证完成任务！

阿呆的思路是这样的，每个客户端(APP,网站,小程序等)，我都给他们声明一个updata方法，然后呢，我服务端声明一个measurementsChanged方法，当数据改变时，我就一个个调用客户端对象的updata方法不就行了，哈哈哈，说干就干，当天晚上阿呆通宵撸码，把第一个版本写了出来，精华版如下：

```
public void measurementsChanged(float temperature, float humidity, float pressure)
{
        this.temperature = temperature;//温度
        this.humidity = humidity;//湿度
        this.pressure = pressure//气压;

        App app = new App();
        app.updata(temperature,humidity,pressure);
        WX wx = new WX();
        wx.updata(temperature,humidity,pressure);
        Web web = new Web();
        wb.updata(temperature,humidity,pressure);


    }
```

简直完美，越看自己代码越发现简直Nice，毫无Bug，但是阿呆这次不敢贸然交给老板，有了前车之鉴，阿呆决定先把第一版程序给自己弟弟阿二呆看一下。

阿呆一脸得意地把程序给二呆，没想到二呆轻轻冷笑了一声，你知道为什么**吃脑怪**不会找你吗?

阿呆:为什么?

二呆:因为你没有脑子！比如有其他的客户端加入进来，你怎么办？不用你说，我知道肯定在measurementsChanged里面new一个新的客户端，然后再调用人家的updata方法，忘了你上一集是怎么死的了吗？！

写程序不是谈恋爱，如胶似漆是大忌，就应该像搞暧昧一样，即使有一方出了问题，另一方也不会有太致命的影响

阿呆听了茅塞顿开，妙呀，实在是妙呀，那，到底该怎么做?

真相只有一个!

代码实战:

本着面向接口编程的思想，我们先建立一个Subject 类，作为我们广播者的超类。

```
/***
 *
 * @author 韩数
 * 定义一个主题的超类，规定方法行为
 *
 */
public interface Subject {
    //注册一个观察者(被通知者)，这里Observer也是所有类的超类
    public void registerObserver(Observer o);
    //移除一个观察者(被通知者)，
    public void removeObserver(Observer o);
    //通知所有观察者(被通知者)，
    public void notifyObservers();
}
```

然后我们声明一个天气广播者的类，**WeatherData**，实现并扩展**Subject** :

```
/***
 *
 * @author 韩数
 * 天气广播者，实现并扩展Subject类
 *
 */
public class WeatherData implements Subject {
    //定义一个ArrayList对象，用于存储注册的观察者类
    private ArrayList observers;
    private float temperature;//温度
    private float humidity;//湿度
    private float pressure;//气压

    //在构造器中初始化observers对象
    public WeatherData() {
        observers = new ArrayList();
    }

    //注册一个观察者
    @Override
    public void registerObserver(Observer o) {
        observers.add(o);
    }
```

```
    //移除一个观察者
    @Override
    public void removeObserver(Observer o) {
        int i = observers.indexOf(o);
        if (i >= 0) {
            observers.remove(i);
        }
    }

    //通知所有观察者
    @Override
    public void notifyObservers() {
        for (int i = 0; i < observers.size(); i++) {
            //Observer是所以观察者的父类，此处运用了多态
            Observer observer = (Observer)observers.get(i);
            observer.update(temperature, humidity, pressure);//通知更新
        }
    }

    public void measurementsChanged() {
        //触发通知
        notifyObservers();
    }

    public void setMeasurements(float temperature, float humidity, float pressure)
{
        this.temperature = temperature;
        this.humidity = humidity;
        this.pressure = pressure;
        //数据改变触发通知
        measurementsChanged();
    }

    //get方法，用于拉数据，后面会讲到
    public float getTemperature() {
        return temperature;
    }

    public float getHumidity() {
        return humidity;
    }

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

同理，对于观察者而言，我们同样也需要定义一个通用的接口Observer 和一个展示数据的接口DisplayElement

```
/***
 *
```

```
 * @author 韩数
 * 声明通知者通用的接口，定义updata方法
 *
 */
public interface Observer {
    public void update(float temp, float humidity, float pressure);
}

public interface DisplayElement {
    public void display();
}
```

定义一个客户端用于显示CurrentConditionsDisplay 当前的实时天气数据：

```
/***
 *
 * @author 韩数
 * 定义一个客户端显示当前数据
 *
 */
public class CurrentConditionsDisplay implements Observer, DisplayElement {

    private float temperature;//温度
    private float humidity;//湿度
    private float pressure;//气压
    private Subject weatherData;

    public CurrentConditionsDisplay(Subject weatherData) {
        this.weatherData = weatherData;//保存 weatherData对象，方便后续移除该观察者
        weatherData.registerObserver(this);//注册观察者
    }

    //通知方法
    @Override
    public void update(float temperature, float humidity, float pressure) {
        this.temperature = temperature;
        this.humidity = humidity;
        this.pressure=pressure;
        display();//调用显示方法显示数据
    }

    //显示方法
    @Override
    public void display() {
        System.out.println("温度: "+temperature+"\n湿度:"+humidity+"\n气
压:"+pressure);
    }
}
```

测试Text:

```
/***
 *
 * @author 测试类
 * 模拟天气改变时通知操作
 *
 */
public class WeatherStation {

    public static void main(String[] args) {

        WeatherData weatherData = new WeatherData();
        CurrentConditionsDisplay conditionsDisplay = new
CurrentConditionsDisplay(weatherData);
        weatherData.setMeasurements(80, 90, 100l);
        weatherData.setMeasurements(90, 90, 100l);
    }
}
```

OUT:

温度：80.0 湿度:90.0 气压:100.0 温度：90.0 湿度:90.0 气压:100.0

代码总结:

通过简单的一个小Demo我们大概已经初步了解观察者模式了，也更加清楚地理解上文所提到的观察者模式的优缺点是如何体现的了，实际上，观察者模式是使用地较为广泛的设计模式，所以Java内置了对观察者模式的支持，接下来我们通过java内置API的方式来说明观察者模式的另外一种通知方式。

中提引入：

生活中大家可能会经常遇到这样的情况，就是什么的，像我寒假回到家，父母总喜欢给我饭碗里夹很多的肉类，但我往往想说的是我想吃肉的时候自己去夹就好了啊，在观察者模式中仍然有时候可能会考虑到这类问题，之前的实现方式就是类似于推(push)的方式把所有数据都通知给每一个观察者，那么有没有另外一种方式，拉(pull)的方式，来让观察者按需调用广播者的get方法获取自己需要的数据呢？答案是有的。

FreeStyle时间到:

我每次都不辞劳苦通知你

可是你却显得有些不乐意

好吧，yeah

下次需要什么你就自己来取

想去哪里全靠你自己

yeah，skr

代码实战：

首先在另外一个包observable中新建一个WeatherData类并继承java.util.Observable类

```
/***
 *
 * @author 新建一个WeatherData类，需要继承java.util.Observable来实现观察者模式。
 *
 */

public class WeatherData extends Observable {
    private float temperature;
    private float humidity;
    private float pressure;

    public WeatherData() { }

    public void measurementsChanged() {
        setChanged();//激活改变状态，使changed值为true
        notifyObservers();//调用通知
    }

    public void setMeasurements(float temperature, float humidity, float pressure)
{
        this.temperature = temperature;
        this.humidity = humidity;
        this.pressure = pressure;
        measurementsChanged();
    }
    //方便观察者取数据
    public float getTemperature() {
        return temperature;
    }

    public float getHumidity() {
        return humidity;
    }

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

首先在另外一个包中新建一个CurrentConditionsDisplay类并继承 java.util.Observer;类

```
public class CurrentConditionsDisplay implements Observer, DisplayElement {

    Observable observable;//定义一个Observable
    private float temperature;
    private float humidity;

    public CurrentConditionsDisplay(Observable observable) {
        this.observable = observable;
        observable.addObserver(this);//注册该观察者
    }
```

```
    @Override
    public void update(Observable obs, Object arg) {
        if (obs instanceof WeatherData) {//如果obs是WeatherData类的一个实例
            WeatherData weatherData = (WeatherData)obs;
            //获取观察者需要的数据
            this.temperature = weatherData.getTemperature();
            this.humidity = weatherData.getHumidity();
            display();//通知显示
        }
    }

    //显示方法
        @Override
        public void display() {
            System.out.println("温度: "+temperature+"\n湿度:"+humidity);
        }
}
```

编写测试类:

```
/***
 *
 * 测试类
 * 模拟天气改变时通知操作
 *
 */

public class WeatherStation {

    public static void main(String[] args) {
        WeatherData weatherData = new WeatherData();
        CurrentConditionsDisplay currentConditions = new
CurrentConditionsDisplay(weatherData);
        weatherData.setMeasurements(80, 65, 30.4f);

    }
}
```

OUT:

温度：80.0 湿度:65.0

源码分析:

首先我们来看我精简过的java.util.Observable类的源码，如下:

```
public class Observable {
    //当对象状态改变时，调用setChanged方法设置changed值为true
    private boolean changed = false;
    private Vector<Observer> obs;
```

```
public Observable() {
    obs = new Vector<>();
}

//注册观察者
public synchronized void addObserver(Observer o) {
    if (o == null)
        throw new NullPointerException();
    if (!obs.contains(o)) {
        obs.addElement(o);
    }
}

//删除观察者
public synchronized void deleteObserver(Observer o) {
    obs.removeElement(o);
}

//通知所有观察者
public void notifyObservers() {
    notifyObservers(null);
}

//通知所有观察者，携带额外参数
public void notifyObservers(Object arg) {

    Object[] arrLocal;

    synchronized (this) {

        if (!changed)
            return;
        arrLocal = obs.toArray();
        clearChanged();
    }

    for (int i = arrLocal.length-1; i>=0; i--)
        //注意这一句：当进行通知行为时，notifyObservers会遍历，并且依次调用
        //观察者对象的update方法，第一个参数为当前通知的广播者对象，本例中也就是
        //WeatherData对象，第二个是其他参数
        ((Observer)arrLocal[i]).update(this, arg);
}


public synchronized void deleteObservers() {
    obs.removeAllElements();
}


protected synchronized void setChanged() {
    changed = true;
}
```

```
}
```

同时来看java.util.Observer类的源码：如下

```
public interface Observer {

    void update(Observable o, Object arg);
}
```

通过查看源码你会发现，本质上java对于观察者的实现和我们自己实现的方式是相差无几的，当数据改变时，调用setChanged方法，将changed值的状态设置为true之后，调用 notifyObservers方法循环遍历通知每一个观察者。

但是，需要值得大家注意的是，由于java.util.Observable本身是一个实体类，并非是一个接口，我们的WeatherData类是继承并且扩展Observable类的，而且Observable的实现有诸多问题，限制了它的使用和复用，同时，由于java不支持多重继承，所以当需要继承另外一个超类的某些行为时，就容易陷入两难的境地，再者，我们可以看到setChanged方法是被protected关键字保护的，这意味着：除非你继承自Observable，否则你无法 创建Observable实例并组合到你自己的对象中来。这个设计违反了第二个设计原则：“多用组合，少用继承”

所以根据自己的实际情况来选择使用java api实现还是自己实现观察者模式才是最佳的选择。

最后阿呆把修改过的程序交给了老板，老板交给了客户，客户很满意，当下就结了尾款，老板回到公司，抱住阿呆嚎啕大哭，说人家突然不想做了，我对不起你啊阿呆，像你这么好的员工。

阿呆安慰老板说，那下次有项目了工资再发也不词嘛，内心无限感动，现在这么好的老板去哪里找啊？

写在最后：

因为寒假事情比较多，可能无法比较规律地按时写文章什么的，同时呢，因为个人水平的问题，如果文章中技术上有所疏漏和错误，希望大家批评，我会努力做的更好。

欢迎大家踊跃给star，您的支持是我写下去的不竭动力！

代码部分请移步本人Github下载：

Github地址：

Github：https://github.com/hanshuaikang/design-pattern-java

知乎：https://www.zhihu.com/people/da-chuan-92-10/activities

参考资料：

菜鸟教程:http://www.runoob.com/design-pattern/strategy-pattern.html

Head frist of 设计模式