设计模式学习----观察者模式（事件监听实现）

octobershiner 于 星期五, 2011-12-16 11:57 提交

今天分享一个设计模式----观察者模式，其实这种设计模式，我们早就接触过，做过 GUI 编程的同学肯定会知道，事件的注册监听等机制，其实这个机制的实现就是利用了观察者模式。
可能在程序中，我们需要在某些数据变化时，其他的类做出一些响应，我们总不能开一个线程，每隔一段时间就去检测数据有没有发生变化吧，那样效率很低，我们 更希望那些具有变化属性的类能够主动将自己的变化推送给，希望检测这些变化的其他类。正如我们去医院排队，不能隔两分钟就去问大夫“轮到我了吗”，而是坐 在那里等大夫的通知，大家都接收到一个名字信号，叫到谁谁就去~ 而观察者就是这样一种模式，大夫的名单就是被观察者，也叫做一个 主题。每一个病人就是一个观察者。因而，观察者模式也叫做 发布订阅模式（相信使用 google reader 的朋友更容易理解）。
下面 是一副来自维基百科的结构图

可以看出来，观察者模式，是一种一对多的关系，即多个观察者监听一个主题。下面用一段代码给大家解释 观察者模式，通过这个例子，大家也可以看到一个缩水版的 事件监听机制的实现~我自己编写了一个交通灯的例子，应该比较形象
首先创建一个主题接口，他声明了一个被观察者应该具有的基本方法，包括添加观察者，删除观察者，通知

```
package observerpattern;

/**
*
* @author liyang
* 2011 08 2
* 主题的接口 定义标准的方法
*/
public interface Subject {
public void addListener(Listener listener);
public void deleteListener(Listener listener);
public void notifyListener();
```

}
接下来我们用一个类实现这个接口，所创建的类也就是我们具体要观察的对象了

```
package observerpattern;

import java.util.ArrayList;

/**
*
* @author liyang
* 2011 08 2
* SE.HIT
* 实现主题的类，相当于图中的ConcretSubject A
*/
public class Lights implements Subject{

private ArrayList lights; //collection 用于保存 观察者的集合
private String whichLight;
private String currentLight = "green"; //记录当前 交通灯的状态
private int time; //记录交通灯持续的时间

public Lights(){
lights = new ArrayList();
}
//实现 添加一个观察者的操作
@Override
public void addListener(Listener listener){
lights.add(listener);
}
//实现删除一个观察者的操作
@Override
public void deleteListener(Listener listener){
int index = lights.indexOf(listener);
if( index != -1){
lights.remove(index);
}
}
//实现通知的机制 ，通知每一个观察者
public void notifyListener(){
int size = lights.size();
for(int i = 0; i < size; i++){
Listener listener = (Listener)lights.get(i);
```

```
listener.updateSignal(whichLight,time);
}
}
//更新主题状态数据的方法
public void setLight(String whichlight,int time){
this.whichLight = whichlight;
this.time = time;
//检查状态是否发生了变化
check();
this.currentLight = whichLight;
}
//实现检查状态的函数
private void check(){
if(! this.currentLight.equals(this.whichLight)){
notifyListener();
}
}
}
```

里面涉及了一些arrayList的东西,不清楚的同学可以好好了解一下arraylist，他里面有个Fail Fast机制，从中你可以学习到更为精彩的装饰器模式，在这里不多解释了，我博客中有一篇相关的总结
下面我们编写一个 观察者的接口，他声明观察者对通知做出的反应的方法

```
package observerpattern;

/**
* @author liyang
* 2011 08 2
* 定义观察者接口 声明更新方法
*/
public interface Listener {
public void updateSignal(String whichLight,int time);
}
```

最后就是编写 观察者了，本例子中我们拥有三个观察者，分别观察红黄绿等是否被打开，然后各自做出相应的通知

```
package observerpattern;
/**
*
* @author liyang
* 2011 08 02
* SE.HIT
* 实现红色交通灯的观察者类，相当于图中国的ConcretObserver类
*/
```

```
public class RedLightListener implements Listener{
@Override
//模拟接收通知，更新状态
public void updateSignal(String whichLight, int time){
if(whichLight.equals("red")){
System.out.println("红灯亮了，禁止通行");
System.out.println("持续时间： "+time);
}
}

}

package observerpattern;
/**
*
* @author liyang
* 2011 08 02
* SE.HIT
* 实现黄色交通灯的观察者类
*/
public class YellowLightListener implements Listener{
@Override
//模拟接收通知，更新状态
public void updateSignal(String whichLight,int time){
if(whichLight.equals("yellow")){
System.out.println("黄灯亮了，请稍等");
System.out.println("持续时间： "+ time);
}
}

}

package observerpattern;

/**
*
* @author liyang
* 2011 08 02
* SE.HIT
* 实现绿色交通灯的观察者类
*/
public class GreenLightListener implements Listener{
@Override
//模拟接收通知，更新状态
```

```
public void updateSignal(String whichLight,int time){
if(whichLight.equals("green")){
System.out.println("绿灯亮了，请通过");
System.out.println("持续时间： "+time);
}
}

}
```

最后就是一个演示的 main 了

```
package observerpattern;

/**
*
* @author liyang
* 2011 08 02
* SE.HIT
* 演示设计模式---观察者模式的 demo
*/
public class ObserverPattern {

public static void main(String[] args) {
// TODO code application logic here
//创建主题，被监听对象，通常是一个自身状态属性可变的类
Lights lights = new Lights();

//创建三个观察者实例
YellowLightListener yellow = new YellowLightListener();
GreenLightListener green = new GreenLightListener();
RedLightListener red = new RedLightListener();
//向主题注册三个观察者，监听变化
lights.addListener(green);
lights.addListener(red);
lights.addListener(yellow);
//手动的改变交通灯的状态
lights.setLight("red",30);
lights.setLight("yellow", 10);
lights.setLight("green", 20);
}
}
```

最后的输出结果为

```
run:
红灯亮了，禁止通行
持续时间： 30
```

黄灯亮了，请稍等
持续时间： 10
绿灯亮了，请通过
持续时间： 20
成功生成（总时间：0 秒）

这就是一个比较完整的观察者模式的例子，大家应该很熟悉，我们平时 addListener 的时候就是在添加 观察者的~