Observer 观察者模式

作者：jsy
原文链接：https://ld246.com/article/1481211960641
来源网站：链滴

Observer模式的Java实现：

- Java的API中已经为我们提供了Observer模式的实现。
- 具体由java.util.Observable类和java.util.Observer接口完成。
- 前者有两个重要的方法：
    1. setChanged：设置内部状态为已改变
    2. notifyObservers(Object obj)：通知观察者所发生的改变，参数obj是一些改变的信息
- 后者有一个核心方法：
    1. update(Object obj)：相应被观察者的改变，其中obj就是被观察者传递过来的信息，该方法会notifyObservers被调用时自动调用。
- 下面是Observer模式的实现过程：
    1. 创建一个被观察者，继承java.util.Observable
    2. 创建一个观察者，实现java.util.Observer接口
    3. 注册观察着，调用addObserver(Observer observer)
    4. 在被观察者改变对象内部状态的地方，调用setChanged()方法，然后调用notifyObservers(Object)方法，通知被观察者
    5. 在观察者的update(Object)方法中，对改变做出响应。

Observer模式的好处：

1. 被观察者只需要知道谁在观察它，无需知道具体的观察细节
2. 被观察者一旦发生变化，只需要通过广播的方式告知观察者，至于消息如何到达则不需知道。这样话无疑消除了被观察者和观察者之间通信的硬编码
3. 当一个被观察者同时被多个观察着观察时，观察者可以只选择自己感兴趣的事件，而忽略其它的事件
4. 多个观察者组合起来可以形成一个观察链，如果一旦需要回滚多个操作，此时观察链可以发挥作用
5. 观察者可以实时对被观察对象的变化做出响应，例如自动告警、中断运行等

参见链接

如何实现简单的观察者模式

- 接下来我们实现壹個简单的被观察者类 ExampleObservable, 代码如下：

```
import java.util.Observable;

public class ExampleObservable extends Observable {
    int data = 0;

    public void setData(int data){
        this.data = data;
```

```
        this.setChanged();//标记此Observable对象为已改变的对象
    this.notifyObservers();//通知所有的观察者
    }
}
```

- 再实现壹個观察者类 ExampleObserver，代码如下：

```
import java.util.Observable;
import java.util.Observer;

public class ExampleObserver implements Observer {
    //有被观察者发生变化，自动调用对应观察者的update方法
    @Override
    public void update(Observable object, Object argument){
    //通过强制类型转换获取被观察者对象
    ExampleObservable example =
    (ExampleObservable)object;

    System.out.println("example.data changed, the new value of data is " + example.data);
    }
}
```

- 我们再写壹個简单的测试类来测试下它是否运行正确，代码如下：

```
public class Main {

    public static void main(String[] args) {
        ExampleObservable example = new ExampleObservable();
        example.addObserver(new ExampleObserver());//给example这个被观察者添加观察者，允
添加多個观察者
        example.setData(2);
        example.setData(-5);
        example.setData(9999);
    }
}
```

- 运行之后在控制台输出如下结果：

```
example.data changed, the new value of data is 2
example.data changed, the new value of data is -5
example.data changed, the new value of data is 9999
```

既是观察者又是被观察者

```
//ObserverA.java
import java.util.Observable;
import java.util.Observer;

public class ObserverA extends Observable implements Observer {

    @Override
    public void update(Observable object, Object arg) {
        ObserverB observerB = (ObserverB)object;
        System.out.println("observerB changed, the new value of observerB.data is " + observerB
```

```
data);
        this.setChanged();
        this.notifyObservers();
    }
}

//ObserverB.java
import java.util.Observable;
import java.util.Observer;

public class ObserverB extends Observable implements Observer {

    int data = 0;
    @Override
    public void update(Observable object, Object arg)
    {
        System.out.println("ObserverB found that ObserverA changed...");
    }

    public void setData(int data){
        this.data = data;
        this.setChanged();
        this.notifyObservers();
    }
}

//Main.java
import net.oschina.bairrfhoinn.multiply.ObserverA;
import net.oschina.bairrfhoinn.multiply.ObserverB;

public class Main {

    public static void main(String[] args) {
        ObserverA a = new ObserverA();
        ObserverB b = new ObserverB();

        a.addObserver(b);
        b.addObserver(a);

        b.setData(2);
    }
}
```

运行之后的结果为：

```
observerB changed, the new value of observerB.data is 2
ObserverB found that ObserverA changed...
```

● 之所以会出现上述运行结果，最初 ObserverA 和 ObserverB 相互之间作为观察者与被观察者，但 ObserverB 的实例 b 先调用的 setData() 方法，然后 ObserverA 的实例 a 观察到了这個变化，于调用了本类的 update 方法打印出了第壹行，紧接着 ObserverA 的 update() 方法在方法体中声明了己发生了变化，于是 ObserverB 观察了这個情况，也调用了自身的 update() 方法并打印了第二句话。