设计模式-观察者模式

> 观察者模式：观察者模式（有时又被称为发布（publish ）-订阅（Subscribe）模式、模型-视图（View）模式、源-收听者(Listener)模式或从属者模式）是软件设计模式的一种。在此种模式中，一个目标物件管理所有相依于它的观察者物件，并且在它本身的状态改变时主动发出通知。这通常透过呼叫各观察者所提供的方法来实现。此种模式通常被用来实现事件处理系统。

实现方式：观察者模式（Observer）完美的将观察者和被观察的对象分离开。举个例子，用户界面可以作为一个观察者，业务数据是被观察者，用户界面观察业务数据的变化，发现数据变化后，就显示在界面上。面向对象设计的一个原则是：系统中的每个类将重点放在某一个功能上，而不是其他方面。一个对象只做一件事情，并且将他做好。观察者模式在模块之间划定了清晰的界限，提高了应用程序的可维护性和重用性。 观察者设计模式定义了对象间的一种一对多的依赖关系，以便一个对象的状态发生变化时，所有依赖于它的对象都得到通知并自动刷新。

过程：比较直观的方式就是当你注册了我们的服务的时候，就会收到通知，当你撤销注册的时候就不会再收到通知。

实例1：（我们以一个微信公众号和一个qq的订阅号为订阅者提供消息，为实例的代码让大家看一下）（自己纯手写了一个观察者模式）：

ObjectForWeiXin.java:

```java
/*
 * @author linSir;
 * 2016-08-04
 * 这是一个微信的公众号
 */

public class ObjectForWeiXin implements Subject {
    /*
     * 一个观察者的集合
     */
    private List<Observer> observers = new ArrayList<Observer>();

    private String msg;// 微信提示的消息

    @Override
    public void registerObserber(Observer obsever) {

        observers.add(obsever);//向用户集合中添加用户
    }

    @Override
    public void removeObserver(Observer obserber) {//移除观察者
        int index = observers.indexOf(obserber);
        if (index >= 0) {
            observers.remove(index);
        }

    }

    @Override
    public void notifyObservers() {//遍历,让每一个用户都更新
        for (Observer observer : observers) {
            observer.update(msg);
        }

    }

    public void setMsg(String msg) {//设置推送信息
        this.msg = msg;
        notifyObservers();
    }

}
```

Observer.java:

```java
/*
 * @author linSir;
 * 2016-08-04
 * 所有用户的基类，有一个更新消息的方法
 */

public interface Observer {

    public void update(String msg);

}
```

Observer1.java:

```java
/*
 * @author linSir;
 * 2016-08-04
 * 模拟的用户1
 */

public class Observer1 implements Observer {


    public Observer1(Subject subject) {
        subject.registerObserber(this);
    }

    public Observer1(Subject subject,Subject subject2) {
        subject.registerObserber(this);
        subject2.registerObserber(this);
    }

    @Override
    public void update(String msg) {
        System.out.println("我(O1)收到消息是--->" + msg + ",我要记下来");
    }

}
```

Observer2.java:

```java
/*
 * @author linSir;
 * 2016-08-04
 * //模拟的用户2
 */

public class Observer2 implements Observer{

    private Subject subject;

    public Observer2(Subject subject) {
        this.subject=subject;
        subject.registerObserber(this);
    }

    @Override
    public void update(String msg) {
        System.out.println("我(O2)收到消息是--->" + msg + ",我要记下来");

    }
}
```

Subject.java:

```java
/*
 * @author lin_sir;
 * 2016-08-04
 */

public interface Subject {
    /*
     * 注册一个观察者
     */
    public void registerObserber(Observer obsever);

    /*
     * 移除一个观察者
     */
    public void removeObserver(Observer obserber);

    /*
     * 通知所有观察者
     */
    public void notifyObservers();

}
```

```
Test.java:
    /*
     * 测试类
     * 用户1订阅了两个公众号，用户2订阅了一个微信的公众号
     */
public class Test {

    public static void main(String[] args) {

        ObjectForWeiXin weiXin = new ObjectForWeiXin();
        ObjectForQQ qq = new ObjectForQQ();

        Observer observer1 = new Observer1(weiXin, qq);
        Observer observer2 = new Observer2(weiXin);

        qq.setMsg("qq : 祝大家开心快乐！");
        weiXin.setMsg("微信 : 祝大家财源滚滚！");
    }
}
```

```
输出结果：
我(O1)收到消息是--->qq : 祝大家开心快乐！,我要记下来
我(O1)收到消息是--->微信 : 祝大家财源滚滚！,我要记下来
我(O2)收到消息是--->微信 : 祝大家财源滚滚！,我要记下来
```

以上便是我们的手写的观察者模式了；

实例2：（我们以一个微信公众号和QQ公众号为实例的代码让大家看一下）（利用java内置的观察者模式来完成）：

下面我们使用java内置的类实现观察者模式：

```
Observer1.java:
/*
 * @author linSir;
 *  2016-08-04
 */
public class Observer1 implements Observer {//模拟的用户

    public void registerSubject(Observable observable) {
        observable.addObserver((Observer) this);
    }

    public void update(Observable o, Object arg) {
        if (o instanceof SubjectForWeiSXin) {
            SubjectForWeiSXin subjectFor3d = (SubjectForWeiSXin) o;
            System.out.println("subjectForWeiXin's msg -- >" + subjectFor3d.getMsg());
        }

        if (o instanceof SubjectForQQ) {
            SubjectForQQ subjectForSSQ = (SubjectForQQ) o;
            System.out.println("subjectForQQ's msg -- >" + subjectForSSQ.getMsg());
        }
    }

}
```

SubjectForQQ.java:

```
/*
 * @author linSir;
 *  2016-08-04
 */

public class SubjectForQQ extends Observable {

    private String msg;

    public String getMsg() {
        return msg;
    }

    public void setMsg(String msg) {
        this.msg = msg;
        setChanged();
        notifyObservers();
    }

}
```

SubjectForWeiSXin.java:

```
/*
 * @author lin_sir;
 *  2016-08-04
 */

public class SubjectForWeiSXin extends Observable {

    private String msg;

    public String getMsg() {
        return msg;
    }

    public void setMsg(String msg) {
        this.msg = msg;
        setChanged();
        notifyObservers();
    }

}
```

Test.java:

```
/*
 * @author linSir;
 *  2016-08-04
 */

public class Test {

    public static void main(String[] args) {

        SubjectForQQ subjectForQQ=new SubjectForQQ();
        SubjectForWeiSXin subjectForWeiSXin=new SubjectForWeiSXin();

        Observer1 observer1=new Observer1();
        observer1.registerSubject(subjectForQQ);
        observer1.registerSubject(subjectForWeiSXin);

        subjectForQQ.setMsg("QQ让聊天更生动");
        subjectForWeiSXin.setMsg("微信让聊天更简洁");



    }
}
```

输出结果:

```
subjectForQQ's msg -- >QQ让聊天更生动
subjectForWeiXin's msg -- >微信让聊天更简洁
```

> 以上就是我们利用java内置的观察者模式，写出来的一段示例代码了，这样的好处是代码非常的简洁，但是并没有使用接口模式，这也是一个不足之处经常为人所诟病，当然我就是抱着学习的态度去看它的，在这里就不加以评价了。