**观察者模式**

**个人github地址：**

https://github.com/ZhanZhiXiang

**开源代码连接：**

https://github.com/youlookwhat/DesignPattern/tree/master/app/src/main/java/com/example/jingbin/designpattern/observer

**任务：**

模拟一个微信3D彩票服务号和一些订阅者

**开发者介绍：**

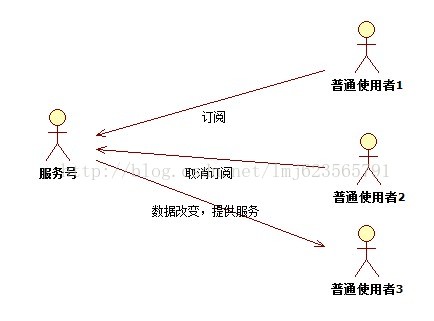
**Jingbin\_**youlookwhat

Android

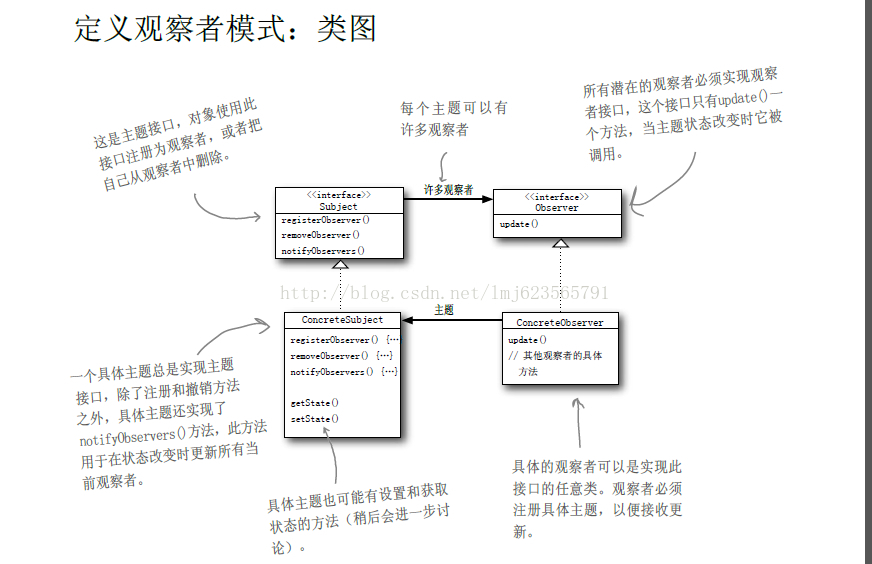
Wuhan China

http://jingbin.me

**模块关系：**







**符合观察者模式的代码：**

首先开始写我们的主题接口和观察者接口：

**[java]** [view plain](http://blog.csdn.net/lmj623565791/article/details/24179699) [copy](http://blog.csdn.net/lmj623565791/article/details/24179699)

1. **package** com.zhy.pattern.observer;
3. /\*\*
4. \* 主题接口，所有的主题必须实现此接口
5. \*
6. \* @author zhy
7. \*
8. \*/
9. **public** **interface** Subject
10. {
11. /\*\*
12. \* 注册一个观察着
13. \*
14. \* @param observer
15. \*/
16. **public** **void** registerObserver(Observer observer);
18. /\*\*
19. \* 移除一个观察者
20. \*
21. \* @param observer
22. \*/
23. **public** **void** removeObserver(Observer observer);
25. /\*\*
26. \* 通知所有的观察着
27. \*/
28. **public** **void** notifyObservers();
30. }

**[java]** [view plain](http://blog.csdn.net/lmj623565791/article/details/24179699) [copy](http://blog.csdn.net/lmj623565791/article/details/24179699)

1. **package** com.zhy.pattern.observer;
3. /\*\*
4. \* @author zhy 所有的观察者需要实现此接口
5. \*/
6. **public** **interface** Observer
7. {
8. **public** **void** update(String msg);
10. }

接下来3D服务号的实现类：

**[java]** [view plain](http://blog.csdn.net/lmj623565791/article/details/24179699) [copy](http://blog.csdn.net/lmj623565791/article/details/24179699)

1. **package** com.zhy.pattern.observer;
3. **import** java.util.ArrayList;
4. **import** java.util.List;
6. **public** **class** ObjectFor3D **implements** Subject
7. {
8. **private** List<Observer> observers = **new** ArrayList<Observer>();
9. /\*\*
10. \* 3D彩票的号码
11. \*/
12. **private** String msg;
14. @Override
15. **public** **void** registerObserver(Observer observer)
16. {
17. observers.add(observer);
18. }
20. @Override
21. **public** **void** removeObserver(Observer observer)
22. {
23. **int** index = observers.indexOf(observer);
24. **if** (index >= 0)
25. {
26. observers.remove(index);
27. }
28. }
30. @Override
31. **public** **void** notifyObservers()
32. {
33. **for** (Observer observer : observers)
34. {
35. observer.update(msg);
36. }
37. }
39. /\*\*
40. \* 主题更新消息
41. \*
42. \* @param msg
43. \*/
44. **public** **void** setMsg(String msg)
45. {
46. **this**.msg = msg;
48. notifyObservers();
49. }
51. }

模拟两个使用者：

**[java]** [view plain](http://blog.csdn.net/lmj623565791/article/details/24179699) [copy](http://blog.csdn.net/lmj623565791/article/details/24179699)

1. **package** com.zhy.pattern.observer;
3. **public** **class** Observer1 **implements** Observer
4. {
6. **private** Subject subject;
8. **public** Observer1(Subject subject)
9. {
10. **this**.subject = subject;
11. subject.registerObserver(**this**);
12. }
14. @Override
15. **public** **void** update(String msg)
16. {
17. System.out.println("observer1 得到 3D 号码  -->" + msg + ", 我要记下来。");
18. }
20. }

**[java]** [view plain](http://blog.csdn.net/lmj623565791/article/details/24179699) [copy](http://blog.csdn.net/lmj623565791/article/details/24179699)

1. **package** com.zhy.pattern.observer;
3. **public** **class** Observer2 **implements** Observer
4. {
5. **private** Subject subject ;
7. **public** Observer2(Subject subject)
8. {
9. **this**.subject = subject  ;
10. subject.registerObserver(**this**);
11. }
13. @Override
14. **public** **void** update(String msg)
15. {
16. System.out.println("observer2 得到 3D 号码 -->" + msg + "我要告诉舍友们。");
17. }


21. }

可以看出：服务号中维护了所有向它订阅消息的使用者，当服务号有新消息时，通知所有的使用者。整个架构是一种松耦合，主题的实现不依赖与使用者，当增加新的使用者时，主题的代码不需要改变；使用者如何处理得到的数据与主题无关；

最后看下[**测试**](http://lib.csdn.net/base/softwaretest)代码：

**[java]** [view plain](http://blog.csdn.net/lmj623565791/article/details/24179699) [copy](http://blog.csdn.net/lmj623565791/article/details/24179699)

1. **package** com.zhy.pattern.observer.test;
3. **import** com.zhy.pattern.observer.ObjectFor3D;
4. **import** com.zhy.pattern.observer.Observer;
5. **import** com.zhy.pattern.observer.Observer1;
6. **import** com.zhy.pattern.observer.Observer2;
7. **import** com.zhy.pattern.observer.Subject;
9. **public** **class** Test
10. {
11. **public** **static** **void** main(String[] args)
12. {
13. //模拟一个3D的服务号
14. ObjectFor3D subjectFor3d = **new** ObjectFor3D();
15. //客户1
16. Observer observer1 = **new** Observer1(subjectFor3d);
17. Observer observer2 = **new** Observer2(subjectFor3d);
19. subjectFor3d.setMsg("20140420的3D号码是：127" );
20. subjectFor3d.setMsg("20140421的3D号码是：333" );
22. }
23. }

输出结果：

**[html]** [view plain](http://blog.csdn.net/lmj623565791/article/details/24179699) [copy](http://blog.csdn.net/lmj623565791/article/details/24179699)

1. observer1 得到 3D 号码  --**>**20140420的3D号码是：127, 我要记下来。
2. observer2 得到 3D 号码 --**>**20140420的3D号码是：127我要告诉舍友们。
3. observer1 得到 3D 号码  --**>**20140421的3D号码是：333, 我要记下来。
4. observer2 得到 3D 号码 --**>**20140421的3D号码是：333我要告诉舍友们。

对于JDK或者Andorid中都有很多地方实现了观察者模式，比如XXXView.addXXXListenter ， 当然了 XXXView.setOnXXXListener不一定是观察者模式，因为观察者模式是一种一对多的关系，对于setXXXListener是1对1的关系，应该叫回调。

上面的观察者模式使我们从无到有的写出，当然了[**Java**](http://lib.csdn.net/base/java)中已经帮我们实现了观察者模式，借助于java.util.Observable和java.util.Observer。

下面我们使用Java内置的类实现观察者模式：

首先是一个3D彩票服务号主题：

**[java]** [view plain](http://blog.csdn.net/lmj623565791/article/details/24179699) [copy](http://blog.csdn.net/lmj623565791/article/details/24179699)

1. **package** com.zhy.pattern.observer.java;
3. **import** java.util.Observable;
5. **public** **class** SubjectFor3d **extends** Observable
6. {
7. **private** String msg ;

10. **public** String getMsg()
11. {
12. **return** msg;
13. }

16. /\*\*
17. \* 主题更新消息
18. \*
19. \* @param msg
20. \*/
21. **public** **void** setMsg(String msg)
22. {
23. **this**.msg = msg  ;
24. setChanged();
25. notifyObservers();
26. }
27. }

下面是一个双色球的服务号主题：

**[java]** [view plain](http://blog.csdn.net/lmj623565791/article/details/24179699) [copy](http://blog.csdn.net/lmj623565791/article/details/24179699)

1. **package** com.zhy.pattern.observer.java;
3. **import** java.util.Observable;
5. **public** **class** SubjectForSSQ **extends** Observable
6. {
7. **private** String msg ;

10. **public** String getMsg()
11. {
12. **return** msg;
13. }

16. /\*\*
17. \* 主题更新消息
18. \*
19. \* @param msg
20. \*/
21. **public** **void** setMsg(String msg)
22. {
23. **this**.msg = msg  ;
24. setChanged();
25. notifyObservers();
26. }
27. }

最后是我们的使用者：

**[java]** [view plain](http://blog.csdn.net/lmj623565791/article/details/24179699) [copy](http://blog.csdn.net/lmj623565791/article/details/24179699)

1. **package** com.zhy.pattern.observer.java;
3. **import** java.util.Observable;
4. **import** java.util.Observer;
6. **public** **class** Observer1 **implements** Observer
7. {
9. **public** **void** registerSubject(Observable observable)
10. {
11. observable.addObserver(**this**);
12. }
14. @Override
15. **public** **void** update(Observable o, Object arg)
16. {
17. **if** (o **instanceof** SubjectFor3d)
18. {
19. SubjectFor3d subjectFor3d = (SubjectFor3d) o;
20. System.out.println("subjectFor3d's msg -- >" + subjectFor3d.getMsg());
21. }
23. **if** (o **instanceof** SubjectForSSQ)
24. {
25. SubjectForSSQ subjectForSSQ = (SubjectForSSQ) o;
26. System.out.println("subjectForSSQ's msg -- >" + subjectForSSQ.getMsg());
27. }
28. }
29. }

看一个测试代码：

**[java]** [view plain](http://blog.csdn.net/lmj623565791/article/details/24179699) [copy](http://blog.csdn.net/lmj623565791/article/details/24179699)

1. **package** com.zhy.pattern.observer.java;
3. **public** **class** Test
4. {
5. **public** **static** **void** main(String[] args)
6. {
7. SubjectFor3d subjectFor3d = **new** SubjectFor3d() ;
8. SubjectForSSQ subjectForSSQ = **new** SubjectForSSQ() ;
10. Observer1 observer1 = **new** Observer1();
11. observer1.registerSubject(subjectFor3d);
12. observer1.registerSubject(subjectForSSQ);

15. subjectFor3d.setMsg("hello 3d'nums : 110 ");
16. subjectForSSQ.setMsg("ssq'nums : 12,13,31,5,4,3 15");
18. }
19. }

测试结果：

**[java]** [view plain](http://blog.csdn.net/lmj623565791/article/details/24179699) [copy](http://blog.csdn.net/lmj623565791/article/details/24179699)

1. subjectFor3d's msg -- >hello 3d'nums : 110
2. subjectForSSQ's msg -- >ssq'nums : 12,13,31,5,4,3 15

可以看出，使用Java内置的类实现观察者模式，代码非常简洁，对了addObserver,removeObserver,notifyObservers都已经为我们实现了，所有可以看出Observable（主题）是一个类，而不是一个接口，基本上书上都对于Java的如此设计抱有反面的态度，觉得Java内置的观察者模式，违法了面向接口编程这个原则，但是如果转念想一想，的确你拿一个主题在这写观察者模式（我们自己的实现），接口的思想很好，但是如果现在继续添加很多个主题，每个主题的ddObserver,removeObserver,notifyObservers代码基本都是相同的吧，接口是无法实现代码复用的，而且也没有办法使用组合的模式实现这三个方法的复用，所以我觉得这里把这三个方法在类中实现是合理的。