概念

一个对象（目标对象）的状态发生改变，所有的依赖对象（观察者对象）都将得到通知，进行广播通知。 典型的应用就是订阅/推送消息

代码

珠宝商运送一批钻石，有黄金强盗准备抢劫，珠宝商雇佣了私人保镖，警察局也派人护送，于是当运输车上路的时候，强盗保镖警察都要观

抽象的观察者

```
public interface Watcher {

    /**
     * 收到通知
     */
    void receiveNotification();
}
```

具体的观察者 - 警察

```
public class Police implements Watcher {
    @Override
    public void receiveNotification() {
        System.out.println("宝物上路了，警察跟上护航");
    }
}
```

具体的观察者 - 保镖

```
public class Security implements Watcher {

    @Override
    public void receiveNotification() {
        System.out.println("运输车上路了，保镖跟上");
    }
}
```

具体的观察者 - 强盗

```java
public class Thief implements Watcher {
    @Override
    public void receiveNotification() {
        System.out.println("运输车上路了，强盗伺机下手");
    }
}
```

抽象的被观察者

```java
/**
 * 抽象的被观察者
 */
public interface Watched {
    void addWatcher(Watcher watcher);

    void removeWatcher(Watcher watcher);

    void notifyWatchers();
}
```

具体的被观察者 - 宝物

```java
/**
 * 具体的被观察者 - 宝物
 */
public class Transporter implements Watched {

    private List<Watcher> watcherList = new LinkedList<>();

    @Override
    public void addWatcher(Watcher watcher) {
        watcherList.add(watcher);
    }

    @Override
    public void removeWatcher(Watcher watcher) {
        watcherList.remove(watcher);
    }

    @Override
    public void notifyWatchers() {
        watcherList.forEach(Watcher::receiveNotification);
    }
}
```

测试类

```java
public class Main {
    public static void main(String[] args) {
        Watched transporter = new Transporter();

        Watcher police = new Police();
        Watcher security = new Security();
        Watcher thief = new Thief();

        transporter.addWatcher(police);
        transporter.addWatcher(security);
        transporter.addWatcher(thief);
```

```
12
13          transporter.notifyWatchers();
14      }
15  }
```

代码2（java中自带的实现机制）

学生监听老师布置作业的事件

Teacher.java

```
1  import java.util.ArrayList;
2  import java.util.List;
3
4  public class Teacher extends java.util.Observable {
5
6      private String name;
7      private List<String> books;
8
9      public String getName() {
10         return this.name;
11     }
12
13     public Teacher(String name) {
14         this.name = name;
15         books = new ArrayList<String>();
16     }
17
18     public void setHomework(String homework) {
19         System.out.printf("%s布置了作业%s \n", this.name, homework);
20         books.add(homework);
21         setChanged();
22         notifyObservers(homework);
23
24     }
25 }
```

Student.java

```
1  import java.util.Observable;
2
3  public class Student implements java.util.Observer {
4
5      private String name;
6
7      public Student(String name) {
8          this.name = name;
9      }
10
11     @Override
12     public void update(Observable o, Object arg) {
13         Teacher teacher = (Teacher) o;
14         System.out.printf("学生%s观察到（实际是被通知）%s布置了作业《%s》 \n", this.name, teacher.getName
15     }
```

```
16
17 }
```

Client.java

```
1  public class Client {
2
3      public static void main(String[] args) {
4          Student student1 = new Student("张三");
5          Student student2 = new Student("李四");
6          Teacher teacher1 = new Teacher("zuikc");
7          teacher1.addObserver(student1);
8          teacher1.addObserver(student2);
9          teacher1.setHomework("事件机制第一天作业");
10     }
11 }
```