观察者模式

2020年2月19日 17:03

1. 概念

定义了对象之间的一对多依赖，让多个观察者对象同时监听某个主题对象，当主题对象发生变化时，它的所有依赖者（观察者）都会收到通知并更新。
京东的降价提醒/到货通知就是观察者模式的典型应用。

2. UML结构图

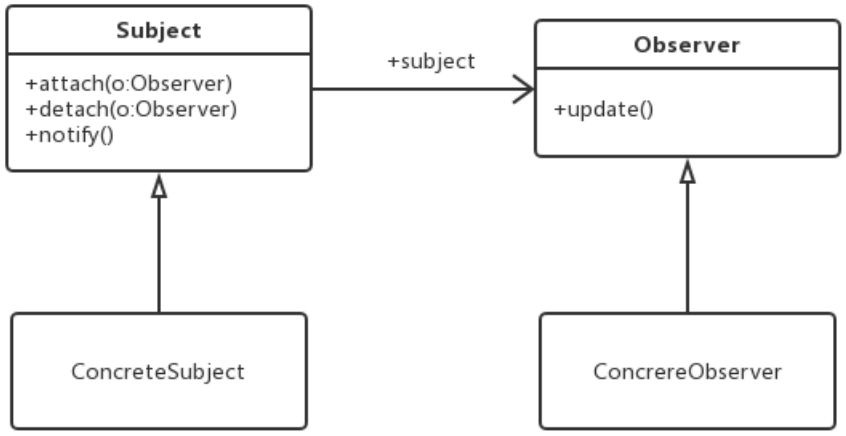

图2.1 观察者模式UML结构图

1. Subject：抽象主题（抽象被观察者），抽象主题角色把所有观察者对象保存在一个集合里，每个主题都可以有任意数量的观察者，抽象主题提供一个接口，可以增加和删除观察者对象。Java.util包中提供了Observable类作为抽象被观察者。
2. ConcreteSubject：具体主题（具体被观察者），该角色将有关状态存入具体观察者对象，在具体主题的内部状态发生改变时，给所有注册过的观察者发送通知。可以继承Observable类作为具体被观察者。
3. Observer：抽象观察者，是观察者者的抽象类，它定义了一个更新接口，使得在得到主题更改通知时更新自己。Java.util包中提供了Observer接口作为抽象观察者。
4. ConcrereObserver：具体观察者，实现抽象观察者定义的更新接口，以便在得到主题更改通知时更新自身的状态。实现Observer接口中的update方法。

3. 观察者模式实例

当有学生对某个老师所上传的课程提出问题时，网站应该提醒老师对问题进行解答。利用观察者模式进行实现，其中老师是观察者，课程是被观察者。
其中抽象主题类和抽象观察者类采用Java.util包下的Observable类和Observer接口。

1. 问题类

```
public class Question {
    private String userName;
    private String questionContent;
    public String getUserName() {
        return userName;
    }
    public void setUserName(String userName) {
        this.userName = userName;
```

```
    }
    public String getQuestionContent() {
        return questionContent;
    }
    public void setQuestionContent(String questionContent) {
        this.questionContent = questionContent;
    }
}
```

2. 具体主题 - 被观察者

```
public class Course extends Observable {
    private String courseName;
    public Course(String courseName) {
        this.courseName = courseName;
    }

    public String getCourseName() {
        return courseName;
    }

    public void produceQuestion(Course course, Question question){
        System.out.println(question.getUserName() + "在" + course.getCourseName() +"提出了问题: " +
question.getQuestionContent());
        // Observable内置方法,表示被观察者状态发生改变
        setChanged();
        // Observable内置方法,通知观察者, 传入的参数为Object对象
        notifyObservers(question);
    }
}
```

3. 具体观察者

```
public class Teacher implements Observer {
    private String teacherName;

    public Teacher(String teacherName) {
        this.teacherName = teacherName;
    }

    //  Observable - 被观察对象 arg - 通知内容
    @Override
    public void update(Observable o, Object arg) {
        Course course = (Course)o;
        Question question = (Question)arg;
        System.out.println(teacherName + "老师的" + course.getCourseName() +"课程接受到一个" +
question.getUserName() + "提交的问题, 内容为: " + question.getQuestionContent());
    }
}
```

4. 客户端

```
 @Test
    public void observerTest(){
        Teacher teacher1 = new Teacher("MR.WANG");
        Teacher teacher2 = new Teacher("MRS.QIU");
        Course course = new Course("设计模式课程");
        course.addObserver(teacher1);
        course.addObserver(teacher2);

        Question question = new Question();
        question.setUserName("HB");
        question.setQuestionContent("老师, 我学不会! ");
        // 相应课程产生问题, 通知观察者
        course.produceQuestion(course,question);
    }
```

5. 结果

```
HB在设计模式课程提出了问题: 老师, 我学不会!
MRS.QIU老师的设计模式课程课程接受到一个HB提交的问题, 内容为: 老师, 我学不会!
MR.WANG老师的设计模式课程课程接受到一个HB提交的问题, 内容为: 老师, 我学不会!
```

5. 优/缺点

1. 优点

- 观察者和被观察者之间建立一个抽象的耦合, 两者都易于扩展。
- 观察者模式支持广播通信。

2. 缺点

- 如果一个被观察者对象有很多的直接和间接的观察者的话，将所有的观察者都通知到会花费很多时间，以及提升程序复杂度。
- 如果在观察者和观察目标之间有循环依赖的话，观察目标会触发它们之间进行循环调用，可能导致系统崩溃
- 观察者模式没有相应的机制让观察者知道所观察的目标对象是怎么发生变化的，而仅仅只是知道观察目标发生了变化。

6. 适用场景

7. 关联行为场景，建立一套触发机制。其中关联行为是可拆分的，而不是“组合”关系

7. 参考资料

[1] https://blog.csdn.net/itachi85/article/details/50773358