此文章已于 21:17:05 2018/3/15 重新发布到 qq\_34850038的博客

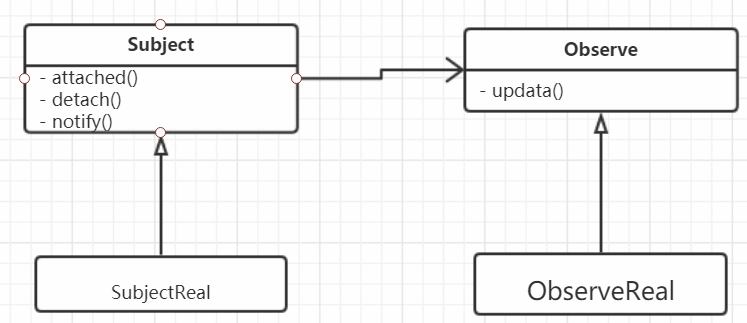
观察者模式

类别 Java ; 设计模式 ; Android

**1.观察者模式模式简介**

观察者模式（又被称为发布-订阅（Publish/Subscribe）模式，属于行为型模式的一种，它定义了一种一对多的依赖关系，让多个观察者对象同时监听某一个主题对象。这个主题对象在状态变化时，会通知所有的观察者对象，使他们能够自动更新自己。

**观察者模式结构图**



**2.观察者模式简单实现**

观察者模式这种发布-订阅的形式我们可以拿微信公众号来举例，假设微信用户就是观察者，微信公众号是被观察者，有多个的微信用户关注了程序猿这个公众号，当这个公众号更新时就会通知这些订阅的微信用户。好了我们来看看用代码如何实现：

**抽象观察者（Observer）**

里面定义了一个更新的方法：

1. **public** **interface** Observer {
2. **public** **void** update(String message);
3. }

**具体观察者（ConcrereObserver）**

微信用户是观察者，里面实现了更新的方法：

1. **public** **class** WeixinUser **implements** Observer {
2. // 微信用户名
3. **private** String name;
4. **public** WeixinUser(String name) {
5. **this**.name = name;
6. }
7. @Override
8. **public** **void** update(String message) {
9. System.out.println(name + "-" + message);
10. }
11. }

**抽象被观察者（Subject）**

抽象主题，提供了attach、detach、notify三个方法：

1. **public** **interface** Subject {
2. /\*\*
3. \* 增加订阅者
4. \* @param observer
5. \*/
6. **public** **void** attach(Observer observer);
7. /\*\*
8. \* 删除订阅者
9. \* @param observer
10. \*/
11. **public** **void** detach(Observer observer);
12. /\*\*
13. \* 通知订阅者更新消息
14. \*/
15. **public** **void** notify(String message);
16. }

**具体被观察者（ConcreteSubject）**

微信公众号是具体主题（具体被观察者），里面存储了订阅该公众号的微信用户，并实现了抽象主题中的方法：

1. **public** **class** SubscriptionSubject **implements** Subject {
2. //储存订阅公众号的微信用户
3. **private** List<Observer> weixinUserlist = **new** ArrayList<Observer>();
4. @Override
5. **public** **void** attach(Observer observer) {
6. weixinUserlist.add(observer);
7. }
8. @Override
9. **public** **void** detach(Observer observer) {
10. weixinUserlist.remove(observer);
11. }
12. @Override
13. **public** **void** notify(String message) {
14. **for** (Observer observer : weixinUserlist) {
15. observer.update(message);
16. }
17. }
18. }

**客户端调用**

1. **public** **class** Client {
2. **public** **static** **void** main(String[] args) {
3. SubscriptionSubject mSubscriptionSubject=**new** SubscriptionSubject();
4. //创建微信用户
5. WeixinUser user1=**new** WeixinUser("杨影枫");
6. WeixinUser user2=**new** WeixinUser("月眉儿");
7. WeixinUser user3=**new** WeixinUser("紫轩");
8. //订阅公众号
9. mSubscriptionSubject.attach(user1);
10. mSubscriptionSubject.attach(user2);
11. mSubscriptionSubject.attach(user3);
12. //公众号更新发出消息给订阅的微信用户
13. mSubscriptionSubject.notify("刘望舒的专栏更新了");
14. }
15. }

**3.使用观察者模式的场景和优缺点**

**使用场景**

* 关联行为场景，需要注意的是，关联行为是可拆分的，而不是“组合”关系。
* 事件多级触发场景。
* 跨系统的消息交换场景，如消息队列、事件总线的处理机制。

**优点**

解除耦合，让耦合的双方都依赖于抽象，从而使得各自的变换都不会影响另一边的变换。

**缺点**

在应用观察者模式时需要考虑一下开发效率和运行效率的问题，程序中包括一个被观察者、多个观察者，开发、调试等内容会比较复杂，而且在Java中消息的通知一般是顺序执行，那么一个观察者卡顿，会影响整体的执行效率，在这种情况下，一般会采用异步实现。