设计模式系列| 观察者模式

程序猿阿星 4 days ago

大家好，我是狼王，一个爱打球的程序员

> 这篇让我们来认识一下**观察者模式**

1、概述

观察者模式 是一种行为设计模式， 允许你定义一种订阅机制， 可在对象事件发生时通知多个 “观察” 该对象的其他对象。

2、适用场景

1）当一个对象状态的改变需要改变其他对象， 或实际对象是事先未知的或动态变化时， 可使用 观察者模式 。

2） 当应用中的一些对象必须观察其他对象时，可使用该模式。但仅能在有限时间内或特定情况下使用。订阅者可随时加入或离开该列表。

3、实例

有以下场景：

有一个小区，需要进行核酸检测。
假设每个人通过关注公众号获取核酸检测结果。

发布者接口：

```
/**
 * 发布接口
 */
public interface IPublisher {

    /**
     * 发布事件
     * @param event
     */
    void publish(IEvent event);
}
```

订阅者接口：

```
/**
 * 通用订阅接口
 */
public interface ISubscriber {

    /**
     * 查看结果
     */
```

```
    void look();
}
```

事件接口：

```
/**
 * 通用事件接口
 */
public interface IEvent {

    /**
     * 打印事件信息
     */
    void print();

}
```

消息发送者

```
/**
 * 消息发送者
 */
public class Publisher implements IPublisher{

    private IEvent event;

    private List<ISubscriber> subscribers;

    public Publisher(IEvent event, List<ISubscriber> subscribers) {
        this.event = event;
        this.subscribers = subscribers;
    }

    /**
     * 发布消息
     * @param event
```

```
     */
    @Override
    public void publish(IEvent event){
        event.print();
    }

    public IEvent getEvent() {
        return event;
    }

    public void setEvent(IEvent event) {
        this.event = event;
    }

    public List<ISubscriber> getSubscribers() {
        return subscribers;
    }

    public void setSubscribers(List<ISubscriber> subscribers) {
        this.subscribers = subscribers;
    }

}
```

事件：

```
/**
 * 检测事件
 */
public class CheckEvent implements IEvent{

    private String name;

    private String result;

    private ISubscriber subscriber;

    public ISubscriber getSubscriber() {
        return subscriber;
    }
```

```
    public void setSubscriber(ISubscriber subscriber) {
        this.subscriber = subscriber;
    }

    public CheckEvent(String name) {
        this.name = name;
    }

    @Override
    public void print() {
        subscriber.look();
        System.out.println("事件名称: " + name);
        System.out.println("事件结果: " + result);
    }


    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getResult() {
        return result;
    }

    public void setResult(String result) {
        this.result = result;
    }
}
```

订阅者:

```
/**
 * 订阅者
 */
public class User implements ISubscriber{
```

```
    private String name;

    public User(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    @Override
    public void look() {
        System.out.println("检测姓名: " + name);
    }
}
```

客户端：

```
/**
 * 测试类
 */
public class TestDemo {

    public static void main(String[] args) {
        //定义两种结果
        String[] doc = {"阴性", "阳性"};
        //初始化检测事件
        CheckEvent check = new CheckEvent("核酸检测");
        //初始化消息发布者
        Publisher publisher = new Publisher(check,new ArrayList<>());
        //实例化接受检测的用户
        List<ISubscriber> users = new ArrayList<>();
        for (int i = 0; i < 10; i++) {
```

```
            //初始化用户
            User user = new User("狼王" + i);
            users.add(user);
        }
        //用户订阅事件
        publisher.setSubscribers(users);
        int index;
        //发布检测结果
        for (int i = 0; i < 10; i++) {
            System.out.println("---------------");
            //随机检测结果
            index = (int) (Math.random() * doc.length);
            check.setSubscriber(users.get(i));
            check.setResult(doc[index]);
            //发布
            publisher.publish(check);
        }

    }
}
```

结果：

```
---------------
检测姓名：狼王0
事件名称：核酸检测
事件结果：阴性
---------------
检测姓名：狼王1
事件名称：核酸检测
事件结果：阴性
---------------
检测姓名：狼王2
事件名称：核酸检测
事件结果：阳性
---------------
检测姓名：狼王3
事件名称：核酸检测
事件结果：阴性
---------------
检测姓名：狼王4
事件名称：核酸检测
```

事件结果：阳性

检测姓名：狼王5
事件名称：核酸检测
事件结果：阳性

检测姓名：狼王6
事件名称：核酸检测
事件结果：阳性

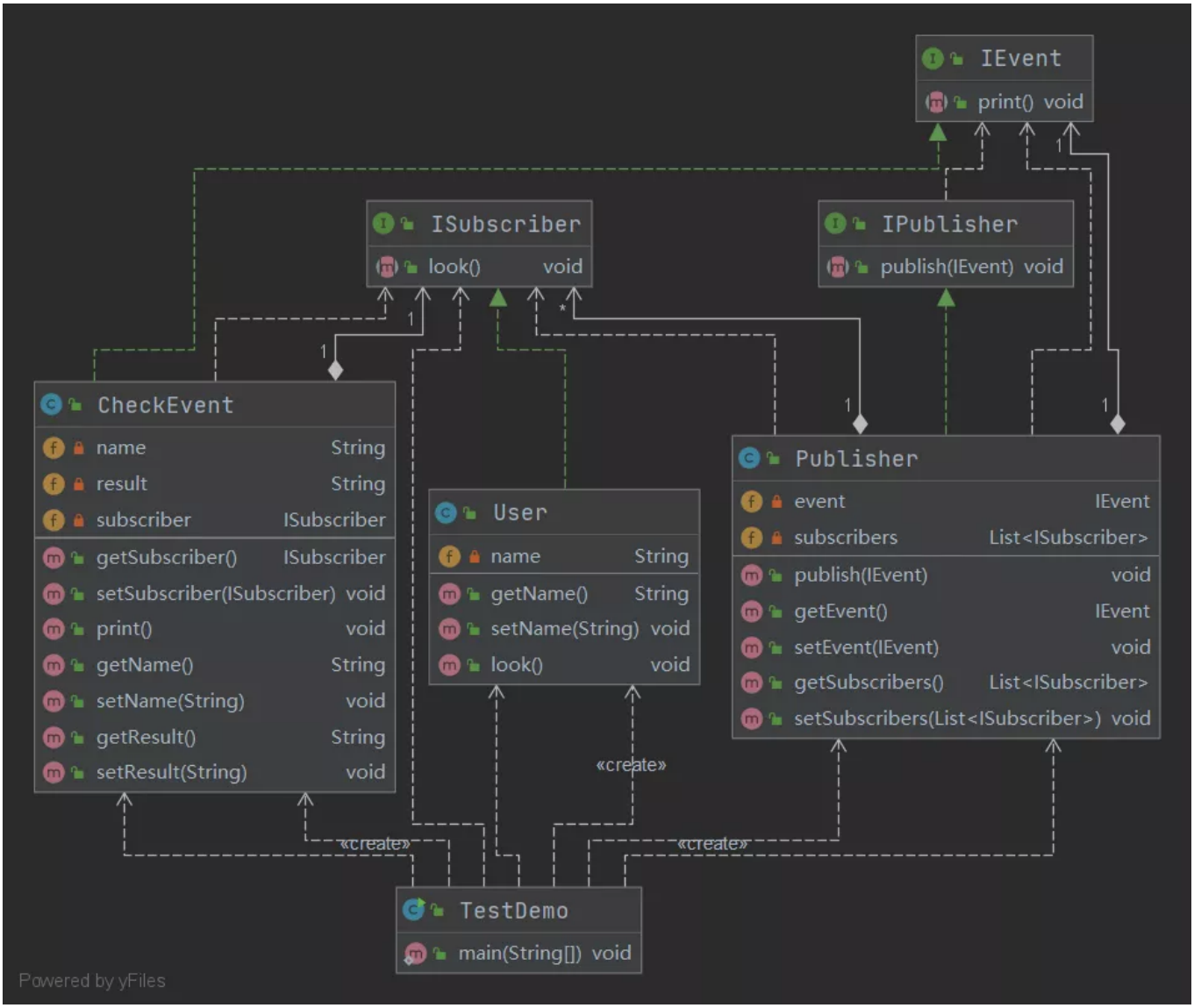
检测姓名：狼王7
事件名称：核酸检测
事件结果：阴性

检测姓名：狼王8
事件名称：核酸检测
事件结果：阴性

检测姓名：狼王9
事件名称：核酸检测
事件结果：阴性

4、分析

代码依赖关系如下图所示：



分别定义了三个接口： 事件接口，消息发布者接口，消息订阅者接口 每个接口有其对应的实现。

这样设计利于后续的扩展，在不同的事件和不同的订阅者以及消息发布者，都可以进行扩展而不影响其他。

5、总结

优点：

1）开闭原则

2）业务代码解耦，具体消息订阅者和发布者没有直接关联。

缺点：

1）多个消费者存在的时候，可能会无法控制顺序和时间较长。

好了。今天就说到这了，我还会不断分享自己的所学所想，希望我们一起走在成功的道路上！

> 乐于输出干货的Java技术公众号：**狼王编程**。公众号内有大量的技术文章、海量视频资源、精美脑图，不妨来关注一下！回复**资料**领取大量学习资源和免费书籍！

QGIS绘制山体阴影图

生态与遥感应用