## 第21章 物业通知很重要，一个住户不能漏——观察者模式

## 21.1

熊小猫：今天你是不是没有洗脸？一定是睡懒觉没顾上吧！上次迟到你说地铁安检太严格，这次我看看你还有什么理由。

兔小白：哎，今天物业修理自来水管道，早上停水了，我也没有办法呀！物业给房主发了短信通知，但没通知我，否则我肯定会提前准备点生活用水。。

熊小猫：因为你没在物业那登记，物业哪里知道你的存在。如果你是租的是我说的那种公寓式的房子，中介会负责通知每一位租住人和房主。

你还记得吧？咱们上次学习中介者模式的练习中，有个场景就是中介转发物业通知给租住人和房主。

兔小白：我记得这个场景，中介者对象会调用租住人对象和房主对象的接受消息接口，将物业的通知转发出去。

熊小猫：在之前的练习中，假设只有一名房主和一名租房人。但真实情况，中介提供服务的房主和租住人肯定不会只有一、两个。当中介管理多个租户和房主时，中介如何处理消息转发就变得复杂起来。

兔小白：你说的这个场景符合实际情况，程序确实需要优化。但是中介者模式中好像没有提到这种场景呀！

熊小猫：设计模式往往需要多个结合起来使用。比如这个地方，可以结合使用观察者模式。咱们把之前的程序拿出来，看看如何使用观察者。

## 21.2

熊小猫：咱们先把之前练习的需求做减法，只保留中介接收通知，转发给房主和租住人的功能。租住人收到停水的消息后，需要提前准备生活用水。房东收到消息后不需要做任何事情。然后再对需求做加法，中介可以维护多位房主和租房人，并且确保通知送达每一位他所维护的房主和租房人。

兔小白：想做到中介维护多位房主和租房人，可以让中介类声明房主和租房人的List。其他的变化不大。

10分钟后，兔小白写完了代码。

兔小白：中介类EstateMediator 声明租住人和房主两个List，并提供2个方法用于向List中添加相应对象。acceptNotification方法中迭代两个List，转发通知。

//代码

房主类HouseOwner在收到通知后，不需要做任何处理。

//代码

租住人类Tenant收到通知后，需要准备生活用水。

//代码

这版代码其实只是将中介类维护单一对象改成了维护对象的List，变化并不大，不过功能完整实现。

熊小猫：功能实现没错，但是优化空间还很大。最大的问题出在两个List的迭代，租住人和房主接收通知的方法已经被你抽象为一模一样，那么为什么不抽象出接口呢？目前中介类依赖具体的租住人和房东类，这不符合依赖倒置原则。这是首先要改造的地方干。

另外，Teant和HouseOwner接收消息的acceptNotification方法，现在不具备扩展性，只能处理中介的消息。假如房主还要接收居委会的消息，现在的程序很难处理。

兔小白：我也看着迭代两个List觉得别扭，听你说完，我知道问题出在哪里了。另外我确实没有考虑消息发送者的扩展问题。我尝试修改一下。

30分钟后，兔小白对熊小猫提出的两个问题进行了优化。

兔小白：接收并处理消息的行为，我抽象成Observer接口。

//代码

可以看到update方法接收一个Subject类型的参数。Subject是对消息发送行为进行抽象后形成的抽象类。它内部维护一个Observer类型的列表，提供了增加和移除元素的方法。notifyObservers方法迭代列表中的每个Observer对象，调用其update方法处理消息。

//代码

EstateMediator类继承Subject类，因此它拥有了维护Observer列表以及通知每一个Observer对象的能力。除此之外，EstateMediator只负责提供接口保存外部传递给他的通知。

//代码

Observer的两个子类HouseOwner和Tenant，实现update方法的形式类似。以HouseOwner为例，update方法接收Subject类型参数，为消息发送者留有了扩展性。通过判断Subject的具体类型，能够区分消息的类型，做相应的处理。

//代码

Tenant类的代码如下。

//代码

客户端代码分别生成2个房主和租住人对象，通过attach方法保存到中介对象的列表中，然后向中介对象发送通知，中介对象会将通知发送给每一个房主和租住人对象。房主和租住人会对此消息进行各自的处理。。

//代码

输出结果和之前一样，符合预期。

//代码

熊小猫：这版代码改造的不错，通知发送者和接收者之间的关系做到了依赖倒置。现在我加入一个新的通知发送者——居委会。居委会发布活动的通知，只有房主会关注居委会的消息，并报名参加。你看看如何实现？

兔小白：这个简单，我增加居委会类Committee，同样继承自Subject类。因此它也拥有了维护Observer列表，发送通知的功能。Committee还可以创建活动，并发送活动通知。

//代码

房主HouseOwner类的update方法需要增加Committee类型的消息处理。

//代码

客户端代码中，创建两个HouseOwner对象，维护进Committee的Observer列表中。调用Committee对象的举办活动方法holdActivity，触发活动通知的发送。

//代码

订阅此类通知的HouseOwner对象执行报名逻辑。程序输出如下。

//代码。

熊小猫：这就是面向接口编程的优势，不会把两个具体的类绑定死。类之间的依赖只认“类型”，扩展性来自于此。

这版代码基本上已经实现了观察者模式，下面我再讲观察者模式，你一定很容易就能理解。

## 20.3

熊小猫：我们来看看观察者模式的结构图。

//图

观察者模式也被称为发布/订阅模式。它的结构很简单，分为Subject和Observer两个继承体系。Subject意为主题，是被订阅的主体。Observer是观察者，它会关注Subject。一个Subject会有多个Observer对象关注它，当Subject状态变化时，会通过notify方法通知每一个观察它的Observer对象。

一个Observer对象也可以关注多个Subject。当Observer对象的update方法被触发时，它需要判断是哪一个主题触发，然后做出与之对应的处理动作。

你程序中的EstateMediator和Committee属于Subject。HouseOwner和Tenant则属于Observer。

观察者模式定义了一种一对多的对象依赖关系。当Subject对象的状态发生改变时，所有依赖它的对象都将得到通知。

观察者模式让主题和观察者只依赖对方干的接口，拥有如下优点。

1. 观察者易于编写。某个对象只要属于Observer接口的实现类，就可以成为主题的观察者。Observer的接口也非常简单，只含有一个已经被高度抽象的update方法。这使得已经存在的类可以被方便地改造为观察者。
2. 主题和观察者可以独立发展。二者依赖接口， 这使得在符合接口定义的前提下，既可以发展出一系列丰富多彩的主题，又可以扩展出功能各异的观察者。

兔小白：观察者之所以这么容易被扩展，关键在于高度抽象的update方法。

熊小猫：确实如此。在实际开发中，update可以不接收任何参数，只作为一个动作触发点。也可以接受Subject参数，将Subject的自身状态传递给Observer。如果Observer需要更多的数据，也可以扩展update方法的参数列表，将与Subject状态无关，但是与Observer计算有关的信息分离出来，单独作为参数传递给update方法

兔小白：我觉得观察者模式有一个不太好的地方。由于update方法的抽象程度过高，当Subject传递了一些不恰当的数据给Observer时，造成接口可以正常调用，但执行过程中过可能会出问题。此类问题在编译期无法被暴露出来。

熊小猫：接口抽象程度的提高，可以带来更好的通用性，但也会造成接口定义过于宽泛，造成问题的暴露延后。建议不要使用update接口传递复杂的数据结构，而应该让每个Observer都具备根据触发update的Subject类型，获取相关数据进行处理的能力。

接下来我们看看观察者模式的适用场景。

1. 一对多的对象关系中，一方对象的状态改变会联动多方响应。例如数据可视化，同一份数据可能绑定多种展示图形，数据的变化会触发所有依赖它的图形发生变化。
2. 广播消息的场景。观察者模式也称之为发布订阅。发布者将消息广播给每一位订阅者，是其核心应用场景。例如在窗体界面中存在多个窗口，某个窗口状态的改变，会引起其窗口的改变。此时可以使用中介者+观察者模式，变化的窗口将通知发布给中介者，中介者再使用观察者模式，将消息广播给所有感兴趣的窗体控件对象。
3. Subject不关心Observer的处理结果。如果Subject需要使用Observer的处理结果来执行后面的操作，并不适合使用观察者模式。原因在于不同类型Observer的update执行结果可能存在较大差异，难以统一返回类型。

观察者模式除了应用在解耦对象关系，还被广泛应用于系统间的解耦。如果两个系统通过API直接调用，会造成高度耦合。我们可以借助消息中间件，如Kafka，以发布/订阅消息的方式进行解耦。

兔小白：看来学习好设计模式，对系统架构也非常有帮助！

熊小猫：还是需要理解底层的原理，否则学习是学不过来的。这也是为什么Java内置了观察者模式的实现，我还要让你自己思考，写了一遍代码。

兔小白：啊？合计我又在重复造轮子？

## 20.4

熊小猫：自己动动手才能深刻理解，否则你的水平只会停留在使用层面。Java从JDK 1.0开始，内置了Observable抽象类，以及Observer接口。Observable抽象类对应观察者模式中的Subject。使用Java内置接口实现观察者模式非常简单，下面我对你的代码进行改造。

首先自己定义的Subject抽象类和Observer接口不再需要，可以直接移除。EstateMediator类继承Observable。在acceptNotification中，需要先调用Observable的setChanged方法设置Observable对象的changed状态为true，这样在调用notifyObservers方法时，才会触发发布消息给Observer。

//代码

HouserOwner和Tenant的修改类似，实现Observer接口，update方法的入参增加了一个Object类型的对象，用来传递不包含在Observable之中的必要数据。下面代码以Tenant为例。

//代码

客户端代码几乎没有改动，只是将添加Observer的方法改为addObserver。

//代码

一般情况，我们使用Java提供的观察者接口就可以满足。但是由于Java不支持多继承，如果作为Subject的类存在父类，那么需要让Subject的父类继承Observable抽象类。

兔小白：毕竟自己已经写了一遍，很容易理解Java实现的观察者墨水。项目中没有特殊情况，肯定首选Java的实现，尽量不去自己造轮子。