|17.1 observer模式

【发送状态变化通知】

在本章中，我们将要学习Observer模式。

Observer的意思是“进行观察的人”，也就是“观察者”的意思。

在Observer模式中，当观察对象的状态发生变化时，会通知给观察者。Observer模式适用于根据对象状态进行相应处理的场景。

|17.2示例程序

下面我们来看一段使用了Observer模式的示例程序。这是一段简单的示例程序，观察者将观察一个会生成数值的对象，并将它生成的数值结果显示出来。不过，不同的观察者的显示方式不一样。Digitobserver会以数字形式显示数值，而Graphobserver则会以简单的图示形式来显示数值。



|17.3 observer模式中的登场角色

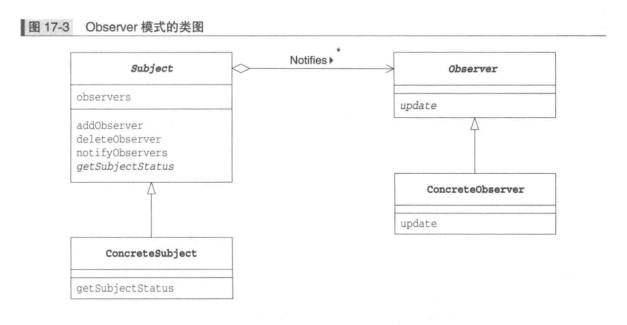
在Observer模式中有以下登场角色。

·Subject（观察对象）Subject角色表示观察对象。Subject角色定义了注册观察者和删除观察者的方法。此外，它还声明了“获取现在的状态”的方法。在示例程序中，由NumberGenerator 类扮演此角色。

·ConcreteSubject（具体的观察对象）ConcreteSubject角色表示具体的被观察对象。当自身状态发生变化后，它会通知所有已经注册的Observer角色。在示例程序中，由RandomNumberGenerator类扮演此角色。

·Observer（观察者）Observer角色负责接收来自Subject角色的状态变化的通知。为此，它声明了update方法。

在示例程序中，由observer接口扮演此角色。·ConcreteObserver（具体的观察者）ConcreteObserver角色表示具体的Observer。当它的update方法被调用后，会去获取要观察的对象的最新状态。在示例程序中，由Digitobserver类和Graphobserver类扮演此角色。



|17.4拓展思路的要点

这里也出现了可替换性

使用设计模式的目的之一就是使类成为可复用的组件。

在Observer模式中，有带状态的ConcreteSubject角色和接收状态变化通知的ConcreteObserver角色。连接这两个角色的就是它们的接口（API）Subject角色和Observer角色。

一方面RandomNumberGenerator类并不知道，也无需在意正在观察自己的（自己需要通知的对象）到底是Digitobserver类的实例还是Graphobserver类的实例。不过它知道在它的observers字段中所保存的观察者们都实现了observer接口。因为这些实例都是通过addobserver方法注册的，这就确保了它们一定都实现了observer接口，一定可以调用它们的update方法。

另一方面，Digitobserver类也无需在意自己正在观察的究竟是RandomNumberGenerator类的实例还是其他xxXXNumberGenerator类的实例。不过，Digitobserver类知道它们是NumberGenerator类的子类的实例，并持有getNumber方法。按照章节顺序阅读本书的读者一定注意到了在本书中已经多次出现了这种可替换性的设计思想。

·利用抽象类和接口从具体类中抽出抽象方法

·在将实例作为参数传递至类中，或者在类的字段中保存实例时，不使用具体类型，而是使用抽象类型和接口

这样的实现方式可以帮助我们轻松替换具体类。

obsever的顺序

Subject角色中注册有多个Observer角色。在示例程序的notifyobservers方法中，先注册的Observer的update方法会先被调用。

通常，在设计ConcreteObserver角色的类时，需要注意这些Observer的update方法的调用顺序，不能因为update方法的调用顺序发生改变而产生问题。例如，在示例程序中，绝不能因为先调用Digitobserver的update 方法后调用Graphobserver的update方法而导致应用程序不能正常工作。当然，通常，只要保持各个类的独立性，就不会发生上面这种类的依赖关系混乱的问题。

不过，我们还需要注意下面将要提到的情况。

|当obsever的行为会对Subject产生影响时

在本节的示例程序中，RandomNumberGenerator类会在自身内部生成数值，调用update方法。不过，在通常的Observer模式中，也可能是其他类触发Subject角色调用update方法。例如，在GUI应用程序中，多数情况下是用户按下按钮后会触发update方法被调用。

当然，Observer角色也有可能会触发Subject角色调用update方法。这时，如果稍不留神，就可能会导致方法被循环调用。



传递更新信息的方式

NumberGenerator利用update方法告诉observer自己的状态发生了更新。传递给update方法的参数只有一个，就是调用update方法的NumberGenerator的实例自身。

observer会在update方法中调用该实例的 getNumber来获取足够的数据。

不过在示例程序中，update方法接收到的参数中并没有被更新的数值。也就是说，update方法的定义可能不是如下（1）中这样，而是如下（2）中这样，或者更简单的（3）这样的。

void update（NumberGenerator generator）；……（1）

void update（NumberGenerator generator，int number）；……（2）

void update（int numbex）；…（3）

（1）只传递了Subject角色作为参数。Observer角色可以从Subject角色中获取数据。

（2）除了传递Subject角色以外，还传递了Observer所需的数据（这里指的是所有的更新信息）。这样就省去了Observer自己获取数据的麻烦。不过，这样做的话，Subject角色就知道了Observer所要进行的处理的内容了。

在很复杂的程序中，让Subject角色知道Observer角色所要进行的处理会让程序变得缺少灵活性。例如，假设现在我们需要传递上次传递的数值和当前的数值之间的差值，那么我们就必须在Subject角色中先计算出这个差值。因此，我们需要综合考虑程序的复杂度来设计update方法的参数的最优方案。

（3）比（2）简单，省略了Subject角色。示例程序同样也适用这种实现方式。不过，如果一个Observer角色需要观察多个Subject角色的时候，此方式就不适用了。这是因为Observer角色不知道传递给update方法的参数究竟是其中哪个Subject角色的数值。

|从“观察”变为“通知”

Observer本来的意思是“观察者”，但实际上Observer角色并非主动地去观察，而是被动地接受来自Subject角色的通知。因此，Observer模式也被称为Publish-Subscribe（发布一订阅）模式。

笔者认为Publish（发布）和Subscribe（订阅）这个名字可能更加合适。

|Model/View/Controler（Mvc）

大家听说过Model/View/Controller（MVC）吗？MVC中的Model和View的关系与Subject角色和Observer角色的关系相对应。Model是指操作“不依赖于显示形式的内部模型”的部分，View则是管理Model“怎样显示”的部分。通常情况下，一个Model对应多个View。

|17.5延伸阅读：java.util.Observer接口

Java类库中的java.util.observer接口和java.util.observable类就是一种Observer模式。

java.util.observer接口中定义了以下方法。

publio void update（Observable obj，Object arg）

面update方法的参数则接收到了如下内容。

·observable类的实例是被观察的Subject角色

·object类的实例是附加信息这与上文中提到的类型（2）相似。

看到这里，大家可能会有这样的想法：原来Java已经为我们提供了Observer模式了啊，那我们直接用就可以了吧。

话虽如此，但是java.util.observer 接口和java.util.observable类并不好用。理由很简单，传递给java.util.observer 接口的Subject角色必须是java.util.observable类型（或者它的子类型）的。但Java只能单一继承，也就说如果Subject角色已经是某个类的子类了，那么它将无法继承java.util.observable类。

Coad书（请参见附录E[Coad]）讲解了这个问题的解决办法。在该书介绍的Observer模式中，Subject角色和observer接口都被定义为Java的接口，这种Observer模式更容易使用。

|17.6相关的设计模式

◆Mediator模式（第16章）在Mediator模式中，有时会使用Observer模式来实现Mediator 角色与Colleague角色之间的通信。

就“发送状态变化通知”这一点而言，Mediator模式与Observer模式是类似的。不过，两种模式中，通知的目的和视角不同。

在Mediator模式中，虽然也会发送通知，不过那不过是为了对Colleague角色进行仲裁而已。

而在Observer模式中，将Subject角色的状态变化通知给Observer角色的目的则主要是为了使Subject 角色和Observer角色同步。