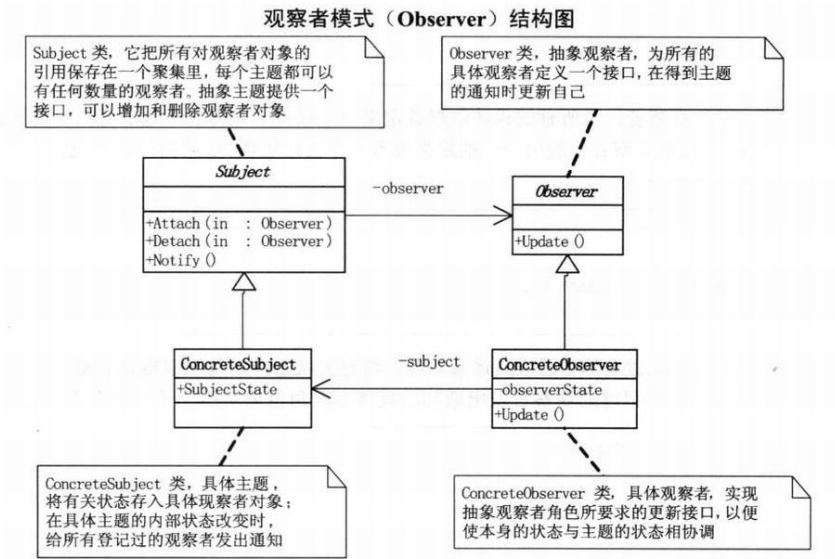
问题：前台通知同事老板回来了，开始正式工作。

分析：

1. 对前台而言，可以有很多同事要通知，在一个通知列表中，也可以从列表中去除某些同事。对同事而言，每个同事要做的事情都是相同的，绑定前台，接受前台的信息并更新自己的工作状态。
2. 观察者模式（publish-subscribe模式）：定义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发生变化时，会通知所有观察者对象，使他们能够自动更新自己。



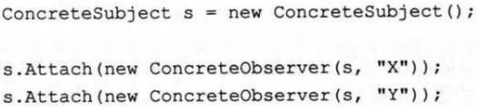
注意的是：在Subject中是有一个List存储所有的Observer，而在ConcreteObserver中是有一个ConcreteSubject作为属性，获取具体主题的状态。

具体的使用过程中，观察者类可能是完全不同的类，所以一般抽象的观察者以接口的形式出现，具体的观察者类实现update的方法。



在这里是将列表放在了抽象的Subject类中

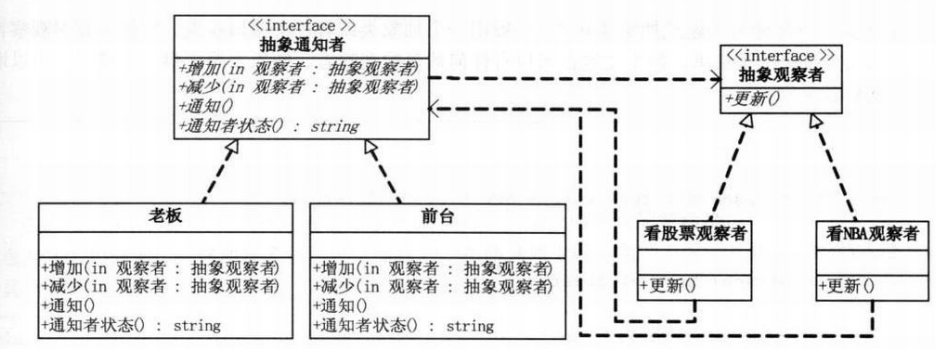
在客户端：





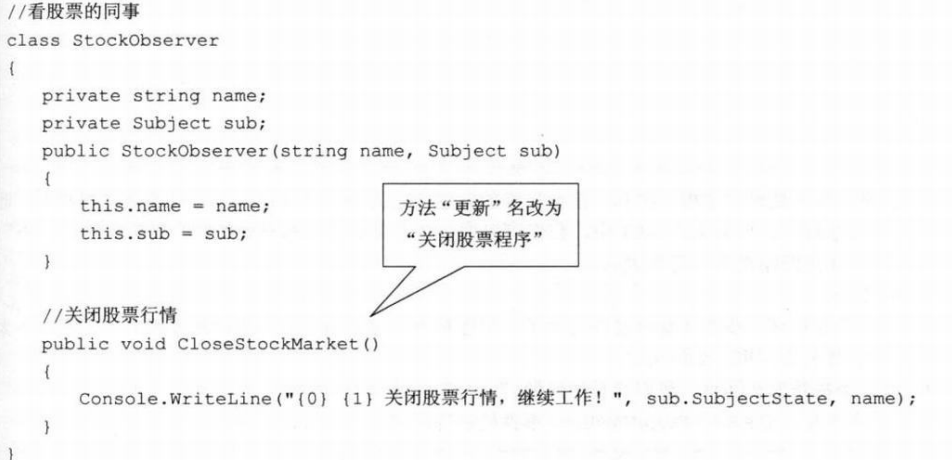
观察者模式是为了解除耦合，让耦合的双方都依赖于抽象，而非具体，从而使得各自的变化都不会影响另一边的变化

1. 根据以上分析，也可以把Subject作为接口，仅包括Attach、Detach和Notify以及一个状态SubjectState，然后在具体的Subject中去写一个列表并实现各种方法：



从图中可以看出，接口抽象通知者依赖于抽象观察者，在具体的观察者中，又依赖于抽象通知者

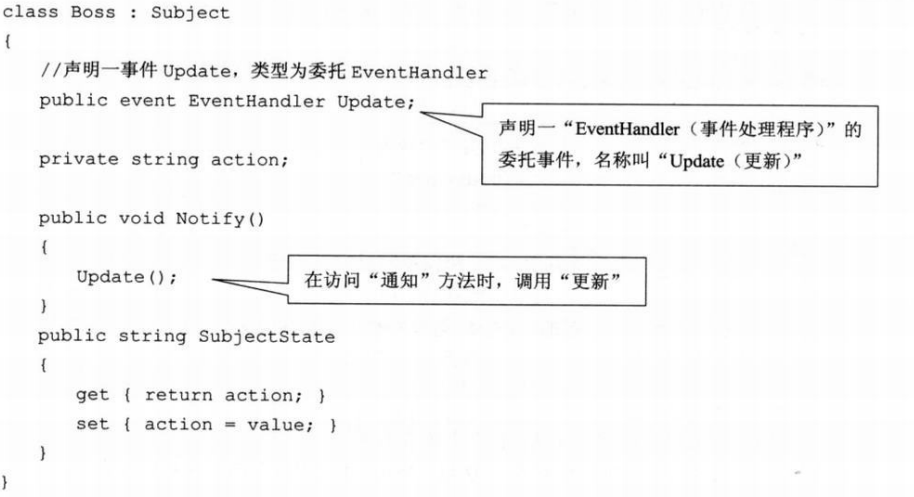
1. 真实情况下，很多类库或控件已经封装好的，如果自己要修改的话，必须再去实现一个Observer的接口，而且并不是每个观察者的方法都叫update。那么在观察者和主题互相不知道时，就需要使用事件委托的方法。比如在java中的button需要为其注册相应的点击事件或浮动事件。
2. 如下 看股票观察者有自己的关闭股票的事件：



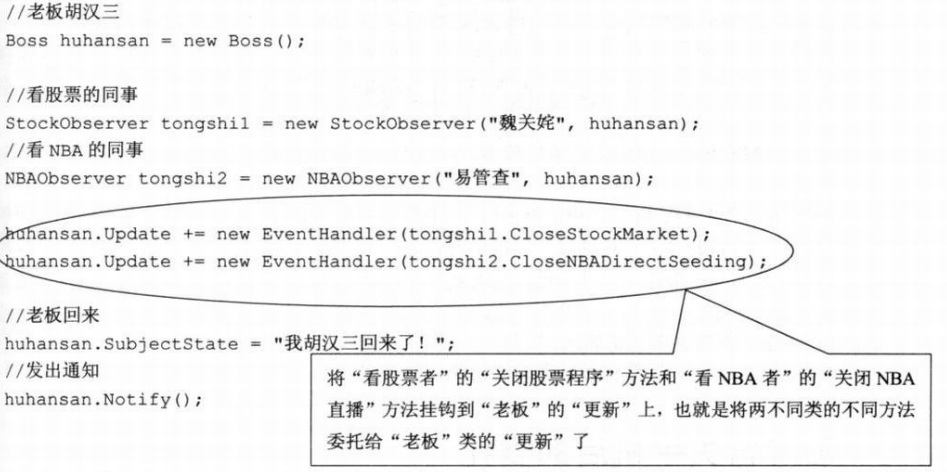
1. 通知者接口

仅包含了Notify和state，避免与抽象观察者的依赖

1. 具体通知者



1. 客户端



可以看到，本来在具体Subject中的Attach和Detach以及对Observer的遍历、抽象观察者都没有了，转到客户端来让委托搭载多个方法，进行事件的注册。

委托（delegate）：一旦为委托分配了方法，那么委托将与该方法具有完全相同的行为。委托方法的使用可以像其他任何方法一样，具有参数和返回值。委托可以看做是对函数的抽象，是函数的类，而委托的实例如上面的Update是一个具体的函数。一个委托可以搭载多个方法，所有方法被依次唤起。它可以使得委托对象所搭载的方法来自不同的类。