18、The Observer Pattern
（观察者模式）

主讲：赵洁琼
成员：葛照君、黄珊珊、朱萍

OBSERVER—想知道咱们公司最新MM情报吗？加入公司的MM情报邮件组就行了，tom负责搜集情报，他发现的新情报不用一个一个通知我们，直接发布给邮件组，我们作为订阅者（观察者）就可以及时收到情报啦

观察者模式：观察者模式定义了一种一队多的依赖关系，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上发生变化时，会通知所有观察者对象，使他们能够自动更新自己。

设计原则

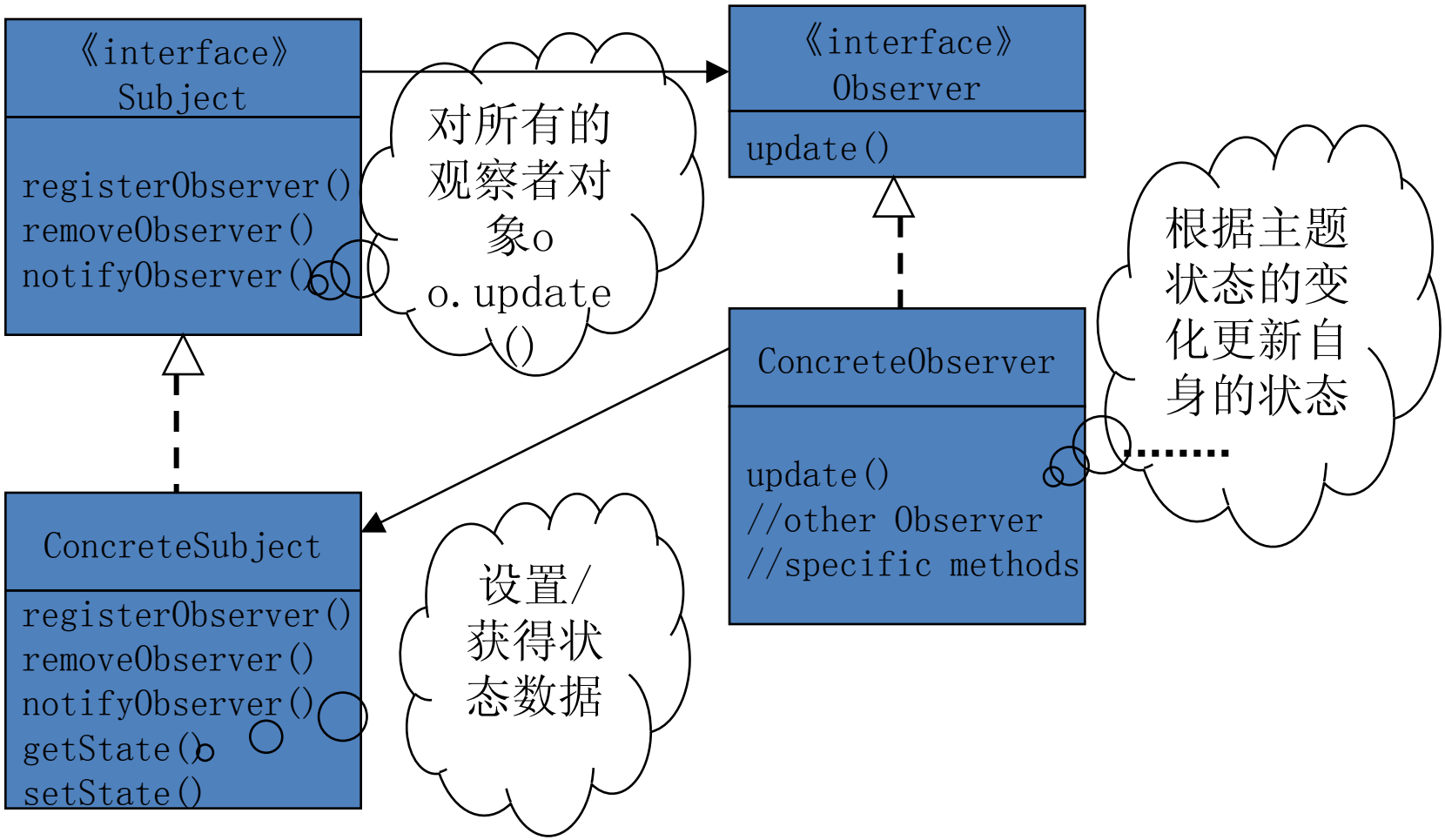
- 将变化部分与固定不变的部分相分离。
- 对该原则的另一种理解是：将变化的部分拿出来进行封装，以便以后你可以修改它而不会影响那些不变的部分。
- 这一原则几乎是所有设计模式的基础，所有设计模式都提供了这样一种机制：让系统的某些部分独立于其他部分变化。

- 对接口编程，而不是对实现编程。

观察者模式的定义

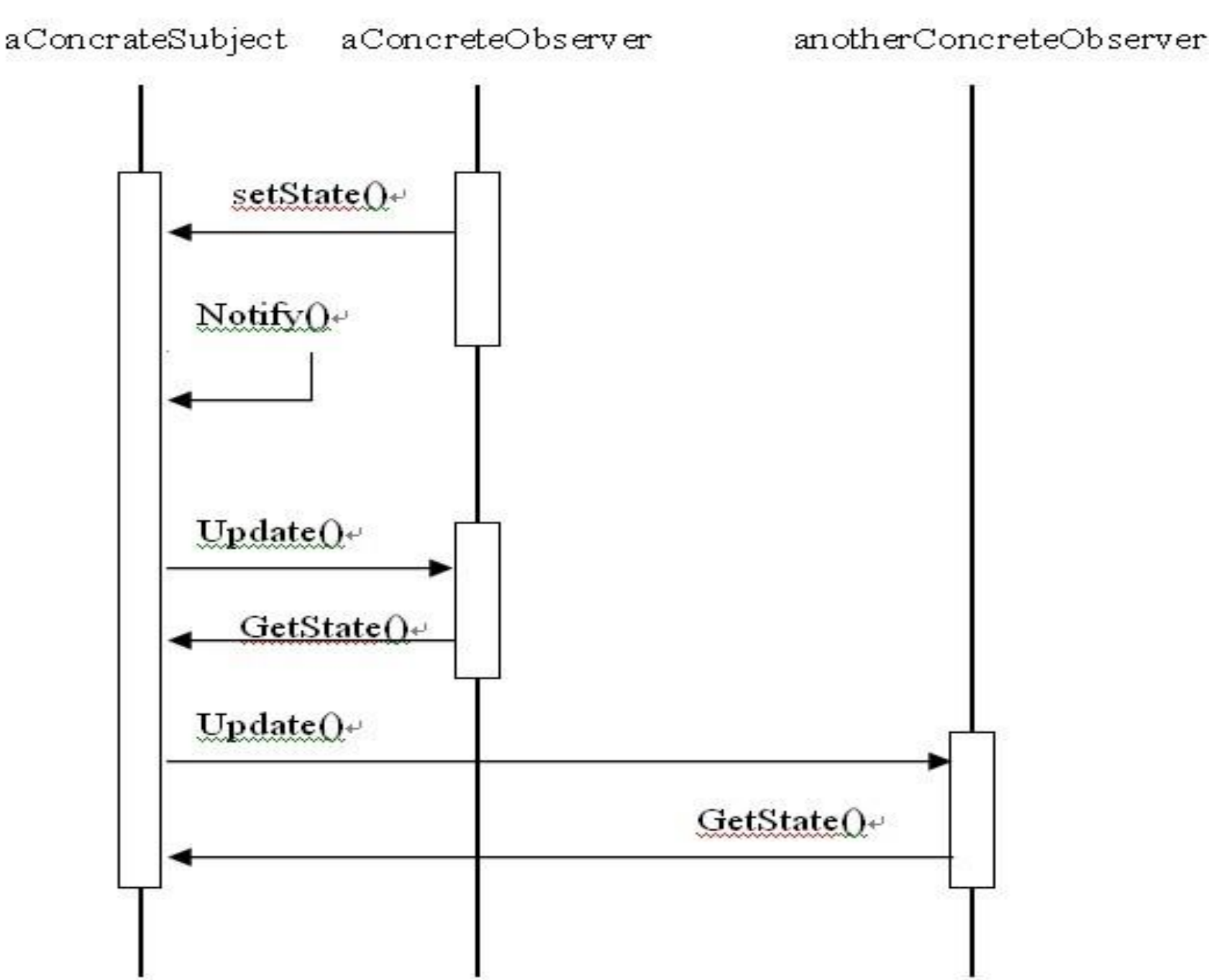
- 观察者模式定义了对象间的一对多依赖关系。当一方的对象改变状态时，所有的依赖者都会被通知并自动被更新。
- 在观察者模式中，被依赖的一方叫做目标或主题（Subject),依赖方叫做观察者(Observers)。

观察者模式的基本类图


《interface》
Subject
registerObserver()
removeObserver()
notifyObserver()
对所有的观察者对象o
o.update()
《interface》
Observer
update()
根据主题状态的变化更新自身的状态
ConcreteObserver
update()
//other Observer
//specific methods
ConcreteSubject
registerObserver()
removeObserver()
notifyObserver()
getState()
setState()
设置/获得状态数据

Observer模式—参与者

- Subject（主题）
  - 知道它的观察者（观察者必须实现了一定的接口），可以有任意多个观察者。
  - 提供注册和注销观察者的接口
- Observer（观察者）
  - 为那些在主题发生变化时需要获得通知的对象定义一个更新（update）接口。
- ConcreteSubject（具体主题）
  - 保持实际状态数据，当状态发生变化时通知各观察者
- ConcreteObserver （具体观察者）
  - 维持一个指向具体主题对象的引用
  - 存储有关状态
  - 实现Observer的更新接口，使自身状态与主题状态保持一致


aConcrateSubject
aConcreteObserver
anotherConcreteObserver
setState()
Notify()
Update()
GetState()
Update()
GetState()

高质量设计的原则---松耦合(loose Coupling)

- 如果两个对象是松耦合的，则他们可以相互作用，但彼此的依赖性很小。
- 观察者模式符合松耦合的原则。因为：
  - 主题(subject)只需要知道其观察者(Observer)实现了某个接口。
  - 可以随时加入观察者。
  - 不需要修改主题就可以加入新的类型的观察者
  - 主题和观察者都可以独立地被复用
  - 修改主题或观察者都不会影响另一方。
  - 观察者之间互不相干。

观察者模式的优点

- 观察者模式在被观察者和观察者之间建立一个抽象的耦合。被观察者角色所知道的只是一个具体观察者聚集，每一个具体观察者都符合一个抽象观察者的接口。被观察者并不认识任何一个具体的观察者，它只知道都有一个共同的接口。由于被观察者和观察者没有紧密地耦合在一起，因此它们可以属于不同的抽象化层次。如果被观察者和观察者都被扔在一起，那么这个对象必然跨越抽象化和具体化层次
- 观察者模式支持广播通信。被观察者会向所有的登记过的观察者发出通知。

观察者模式的缺点

- 如果一个被观察者对象有很多直接和间接的观察者的话，将所有的观察者都通知到会花费很多时间
- 如果在被观察者之间有循环依赖的话，被观察者会触发它们之间进行循环调用，导致系统崩溃。
- 如果对观察者的通知是通过另外的线程进行异步投递的话，系统必须保证投递是以自恰的方式进行的。
- 虽然观察者模式可以随时的使观察者知道所观察的对象发生的变化，但是观察者模式没有相应的机制使观察者知道所观察的对象是怎么发生变化的。

总结：Observer模式

- 意图
  - 定义对象间的一种一对多的依赖关系。当一方的对象改变状态时，所有的依赖者都会得到通知并被自动更新。
- 别名
  - 依赖（Dependents）
  - 发布-订阅(Publish-Subscribe)

总结：Observer模式--动机

- 将一个系统分割成一系列相互协作的类有一个常见的副作用：需要维护相关对象间的一致性。我们不希望为了维持一致性而使得各个类紧密耦合，导致可重用性的降低。
- 观察者模式使得任意数目的观察者不必知道彼此的存在，且主题发生变化时都可以得到主题的通知，而同步改变状态。是一种很松的耦合，具有很好的可重用性。

总结：Observer模式--适用性

- 当一个抽象模型有两个方面，其中一个方面依赖于另一个方面时，将这两者封装在独立的对象中使他们可以独立的改变和复用。
- 当一个对象的改变需要同时改变其他对象，而不知道具体有多少对象需要改变。
- 当一个对象必须通知其他对象，而他又不能假定其它对象是谁。