返回目录:全网各大厂 iOS 面试题-题集大全

通知，代理，KVO 的区别，以及通知的多线程问题

1. delegate

当我们第一次编写 ios 应用时，我们注意到不断的在使用“delegate”，并且贯穿于整个 SDK。delegation 模式不是 IOS 特有的模式，而是依赖与你过去拥有的编程背景。针对它的优势以及为什么经常使用到，这种模式可能不是很明显的。

delegation 的基本特征是:一个 controller 定义了一个协议（即一系列的方法定义）。该协议描述了一个 delegate 对象为了能够响应一个 controller 的事件而必须做的事情。协议就是 delegator 说，“如果你想作为我的 delegate，那么你就必须实现这些方法”。实现这些方法就是允许 controller 在它的 delegate 能够调用这些方法，而它的 delegate 知道什么时候调用哪种方法。delegate 可以是任何一种对象类型，因此 controller 不会与某种对象进行耦合，但是当该对象尝试告诉委托事情时，该对象能确定 delegate 将响应。

delegate 的优势：

- 1.非常严格的语法。所有将听到的事件必须是在 delegate 协议中有清晰的定义。
- 2.如果 delegate 中的一个方法没有实现那么就会出现编译警告/错误
- 3.协议必须在 controller 的作用域范围内定义
- 4.在一个应用中的控制流程是可跟踪的并且是可识别的；
- 5.在一个控制器中可以定义定义多个不同的协议，每个协议有不同的 delegates
- 6.没有第三方对象要求保持/监视通信过程。
- 7.能够接收调用的协议方法的返回值。这意味着 delegate 能够提供反馈信息给 controller

缺点：

- 1.需要定义很多代码：1.协议定义；2.controller 的 delegate 属性；3.在 delegate 本身中实现 delegate 方法定义
- 2.在释放代理对象时，需要小心的将 delegate 改为 nil。一旦设定失败，那么调用释放对象的方法将会出现内存 crash
- 3.在一个 controller 中有多个 delegate 对象，并且 delegate 是遵守同一个协议，但还是很难告诉多个对象同一个事件，不过有可能。

2. notification

在 iOS 应用开发中有一个”Notification Center“的概念。它是一个单例对象，允许当事件发生时通知一些对象。它允许我们在低程度耦合的情况下，满足控制器与一个

任意的对象进行通信的目的。这种模式的基本特征是为了让其他的对象能够接收到在该 controller 中发生某种事件而产生的消息，controller 用一个 key（通知名称）。这样对于 controller 来说是匿名的，其他的使用同样的 key 来注册了该通知的对象（即观察者）能够对通知的事件作出反应。

通知优势：

- 1.不需要编写多少代码，实现比较简单；
- 2.对于一个发出的通知，多个对象能够做出反应，即 1 对多的方式实现简单
- 3.controller 能够传递 context 对象（dictionary），context 对象携带了关于发送通知的自定义的信息

缺点：

- 1.在编译期不会检查通知是否能够被观察者正确的处理；
- 2.在释放注册的对象时，需要在通知中心取消注册；
- 3.在调试的时候应用的工作以及控制过程难跟踪；
- 4.需要第三方对喜爱那个来管理 controller 与观察者对象之间的联系；
- 5.controller 和观察者需要提前知道通知名称、UserInfo dictionary keys。如果这些没有在工作区间定义，那么会出现不同步的情况；
- 6.通知发出后，controller 不能从观察者获得任何的反馈信息

3. KVO

KVO 是一个对象能够观察另外一个对象的属性的值，并且能够发现值的变化。前面两种模式更加适合一个 controller 与任何其他的对象进行通信，而 KVO 更加适合任何类型的对象侦听另外一个任意对象的改变（这里也可以是 controller，但一般不是 controller）。这是一个对象与另外一个对象保持同步的一种方法，即当另外一种对象的状态发生改变时，观察对象马上作出反应。它只能用来对属性作出反应，而不会用来对方法或者动作作出反应。

优点：

- 1.能够提供一种简单的方法实现两个对象间的同步。例如：model 和 view 之间同步；
- 2.能够对非我们创建的对象，即内部对象的状态改变作出响应，而且不需要改变内部对象（SKD 对象）的实现；
- 3.能够提供观察的属性的最新值以及先前值；
- 4.用 key paths 来观察属性，因此也可以观察嵌套对象；
- 5.完成了对观察对象的抽象，因为不需要额外的代码来允许观察值能够被观察

缺点：

- 1.我们观察的属性必须使用 strings 来定义。因此在编译器不会出现警告以及检查；
- 2.对属性重构将导致我们的观察代码不再可用；
- 3.复杂的“IF”语句要求对象正在观察多个值。这是因为所有的观察代码通过一个方法来指向；
- 4.当释放观察者时不需要移除观察者。

总结:

1. 从上面的分析中可以看出 3 中设计模式都有各自的优点和缺点。其实任何一种事物都是这样，问题是如何在正确的时间正确的环境下选择正确的事物。下面就讲讲如何发挥他们各自的优势，在哪种情况下使用哪种模式。注意使用任何一种模式都没有对和错，只有更适合或者不适合。每一种模式都给对象提供一种方法来通知一个事件给其他对象，而且前者不需要知道侦听者。在这三种模式中，我认为 KVO 有最清晰的使用案例，而且针对某个需求有清晰的实用性。而另外两种模式有比较相似的用处，并且经常用来给 controller 间进行通信。那么我们在什么情况使用其中之一呢？

2. 根据我开发 iOS 应用的经历，我发现有些过分的使用通知模式。我个人不是很喜欢使用通知中心。我发现用通知中心很难把握应用的执行流程。UserInfo dictionaries 的 keys 到处传递导致失去了同步，而且在公共空间需要定义太多的常量。对于一个工作于现有的项目的开发者来说，如果过分的使用通知中心，那么很难理解应用的流程。

3. 我觉得使用命名规则好的协议和协议方法定义对于清晰的理解 controllers 间的通信是很容易的。努力的定义这些协议方法将增强代码的可读性，以及更好的跟踪你的 app。代理协议发生改变以及实现都可通过编译器检查出来，如果没有将会在开发的过程中至少会出现 crash，而不仅仅是让一些事情异常工作。甚至在同一事件通知多控制器的场景中，只要你的应用在 controller 层次有着良好的结构，消息将在该层次上传递。该层次能够向后传递直至让所有需要知道事件的 controllers 都知道。

4. 当然会有 delegation 模式不适合的例外情况出现，而且 notification 可能更加有效。例如：应用中所有的 controller 需要知道一个事件。然而这些类型的场景很少出现。另外一个例子是当你建立了一个架构而且需要通知该事件给正在运行中应用。

5. 根据经验，如果是属性层的时间，不管是在不需要编程的对象还是在紧紧绑定一个 view 对象的 model 对象，我只使用观察。对于其他的事件，我都会使用 delegate 模式。如果因为某种原因我不能使用 delegate，首先我将估计我的 app 架构是否出现了严重的错误。如果没有错误，然后才使用 notification。

返回目录:全网各大厂 iOS 面试题-题集大全

更多精选大厂 · iOS 面试题答案 PDF 文集

*