iOS初学者容易理解错误的几个问题

这是之前写的一篇博客，搬运到这里。

这几天在为公司招聘iOS开发实习生，通过面试，发现很多同学存在普遍误区的几个问题，列举出来希望大家以后在学习的时候少走弯路。

1.深拷贝和浅拷贝

这是这几周的面试以来，理解错误率最高的问题了，不是100%也有90%以上。至于错误的理解，有很多种：~~1.mutableCopy是深拷贝，copy是浅拷贝。2.浅拷贝是指针复制，深拷贝是内存复制（很多同学的原话）。~~下面就来仔细说说深拷贝和浅拷贝的问题。

~~mutableCopy是深拷贝，copy是浅拷贝~~

很多Foundation对象都有可变和不可变两种类型（NSString-NSMutableString、NSArray-NSMutableArray、NSDictionary-NSMutableDictionary等等）。当对这些对象（无论是可变还是不可变）调用copy方法，返回的是不可变类型，如果要返回可变类型，则需要调用mutableCopy。这也是一个可变类型的属性不可以用copy来修饰的原因。copy需要继承NSCopying协议并实现copyWithZone:方法；mutableCopy需要继承NSMutableCopying协议并实现mutableCopyWithZone:方法。这和深浅拷贝并没有关系，仅仅只是拷贝生成的对象是可变 or 不可变的区别。

~~浅拷贝是指针复制，深拷贝是内存复制~~

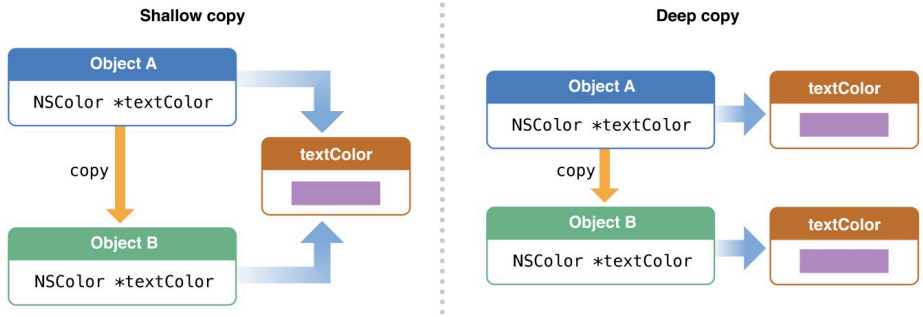
做这种解释的同学数量最多。我如果追问一句：用一个新的指针指向该对象并做一次引用，和浅拷贝有什么区别呢？同学便会先犹豫再一下回答：~~没有区别~~。

深拷贝和浅拷贝的区别

费了这么多话，该说说深拷贝和浅拷贝的真正区别了。

苹果关于对象拷贝的官方文档 这篇文档完美地解释了深浅拷贝的区别。话不多说，直接上图：

> Copies of objects can be shallow or deep. Both shallow- and deep-copy approaches directly duplicate scalar properties but differ on how they handle pointer references, particularly references to objects (for example, `NSString *str`). A deep copy duplicates the objects referenced while a shallow copy duplicates only the references to those objects. So if object A is shallow-copied to object B, object B refers to the same instance variable (or property) that object A refers to. Deep-copying objects is preferred to shallow-copying, especially with value objects.



图中的shallow copy即浅拷贝，deep copy即深拷贝。不难发现，深浅拷贝都会复制Object A到Object B（开辟新的内存空间），但是区别就是他们成员变量所引用的对象处理不同：浅拷贝中，源对象（Object A）和新对象（Object B）的textColor指针指向同一个textColor对象；而深拷贝中，源对象和新对象的textColor指针指向不同的textColor对象，既textColor也被复制了一份。这才是深浅拷贝的区别。

下面翻译一下图上那段话来帮助同学们理解问题：对象的拷贝可以是深拷贝和浅拷贝。深浅拷贝都会直接复制基本类型的属性，而他们的不同之处就在于如何处理指针引用，特别是对象的引用（例如NSString *str）。深拷贝会复制被引用的对象，而浅拷贝只复制那些对象的引用。因此如果对象A被浅拷贝到对象B，对象B关联的变量（或者属性）与对象A关联的是同一个实例。深拷贝的对象最好是浅复制，特别是值对象。

关于拷贝的一些问题

有的同学会问：对一个不可变NSArray对象调用copy方法，返回指针和原指针相等，这又是怎么回事呢？`[array copy]`和`[NSArray arrayWithArray:array]`又有什么区别呢？
这两个问题放到一起说会比较好。对于不可变NSArray，调用copy是会直接返回自身的引用，原因是apple在实现NSArray的copyWithZone:时做了处理，直接`return [self retain]`；而`[NSArray arrayWithArray:array]`会创建一个新的NSArray对象（似乎这才是真正的copy，然而新数组中的元素仍然是引用原先的元素）。 同样的问题在NSString、NSDictionary等类中也存在，同学们使用的时候也应该多加注意。

2.Dispatch Queue的理解

内存管理和多线程，是iOS面试中必问的两个问题。OC多线程的三种实现方式中，GCD是被用得最多，最好用的。但是同学们对GCD的重要成员：Dispatch Queue的理解普遍不那么到位。
首先看一下苹果官方对GCD的说明：
开发者要做的只是定义想执行的任务并追加到适当的Dispatch Queue中。

“Dispatch Queue”是什么呢

如其名称所示，是执行处理的等待队列。程序员通过`dispatch_async`函数等API，在Block语法中记述想执行的任务，并将其追加到Dispatch Queue中。Dispatch Queue按照追加的顺序（FIFO）执行任务。在执行任务的时候存在两种Dispatch Queue，一种是串行的Serial Dispatch Queue，另一种是并行的Concurrent Dispatch Queue。

Serial Dispatch Queue

Serial Dispatch
Queue为串行队列，它需要等待当前任务的处理结束，才会处理队列中下一个任务，如此重复，同时处理的任务数只能有1个，并且一定按照队列顺序进行处理。

Concurrent Dispatch Queue

Concurrent Dispatch
Queue为并行队列，它不用等待当前任务的处理结束，就可以开始处理后面的任务。这样虽然不用等待处理结束，可以并行执行多个任务，但并行执行的任务数量取决于当前系统的状态。即iOS和OS X基于Dispatch
Queue中的处理数、CPU核数以及CPU负荷等当前系统的状态来决定Concurrent Dispatch
Queue中并行处理数。所谓“并行执行”，就是使用多个线程同时执行多个处理。

Dispatch Queue的获取和创建

第一种方法是通过GCD的API生成Dispatch Queue。

通过`dispatch_queue_create`函数可生成Dispatch Queue，通过传入的参数来决定生成的是Serial Dispatch Queue还是Concurrent Dispatch Queue。

如前所述，Concurrent Dispatch Queue并行执行多个任务，而Serial Dispatch Queue同时只能执行一个任务。虽然Serial Dispatch Queue和Concurrent Dispatch Queue收到系统资源的限制，但是`dispatch_queue_create`函数可以生成任意多个Dispatch Queue。

当生成多个Serial Dispatch Queue时，各个Serial Despatch Queue将并行执行。虽然在1个Serial Dispatch
Queue中同时只能处理一个任务，但如果将任务分别追加到4个Serial Dispatch Queue中，各个Serial Dispatch
Queue执行一个，即为同时执行4个任务。一旦生成Serial Dispatch Queue并追加任务，系统对于一个Serial Dispatch
Queue就只生成并使用一个线程。如果生成2000个Serial Dispatch Queue，那么就生成2000个线程。如果过多使用多线程，就会消耗大量内存，引起大量的上下文切换，大幅度降低系统的响应性能。因此，只在为了避免多线程会造成资源竞争时才使用Serial Dispatch Queue，而且Serial Dispatch Queue的生成数量要加以控制，仅生成必须的数量。例如更新数据库时1个表生成1个Serial Dispatch
Queue；更新文件时，1个文件或是可以分割的1个文件块生成1个Serial Dispatch Queue。绝对不能激动之下大量生成Serial Dispatch Queue。

当并发执行不会发生资源竞争的时候，使用Concurrent Dispatch Queue。而且对于Concurrent Dispatch
Queue来说，不管生成多少，由于XNU内核只使用有效管理的线程，因此不会发生Serial Dispatch Queue的那些问题。

第二种方法是获取系统标准提供的Dispatch Queue。

那就是Main Dispatch Queue和Global Dispatch Queue。

Main Dispatch Queue正如其名“Main”一样，是在主线程中执行的Dispatch Queue。因为主线程只有一个，所以Main Dispatch Queue自然就是Serial Dispatch Queue。

另一个Global Dispatch Queue是所有应用程序都能够使用的Concurrent Dispatch Queue。没有必要通过`dispatch_queue_create`函数逐个生成Concurrent Dispatch Queue，只要获取Global Dispatch Queue使用即可。另外，Global Dispatch Queue有4个执行优先级（High Priority、Default Priority、Low Priority、Background Priority），但通过XNU内核用于Global Dispatch Queue的线程并不能保证实时性，因此执行优先级只是大致的判断。

另外，对于Main Dispatch Queue和Global Dispatch Queue执行`dispatch_retain`函数和`dispatch_release`函数不会引起任何变化，也不会有任何问题。