iOS weak 关键字漫谈

weak 关键字的运用在 iOS 当中属于基础知识，在面试的时候问 weak 的用处，就像两个 iOS 程序员见面寒暄问候一样普通了。

weak 的常见场景是在 delegate，block，NSTimer 中使用，以避免循环引用所带来的内存泄漏，这是教科书式的用法。

编程语言是工具，语言特性只是工具的特性，工具怎么用在于使用者。weak 关键字的方便之处绝不局限于避免循环引用，适当脑洞，可以在其他场景下带来一些有趣的应用。

weak 的用处用一句话可归纳为：**弱引用，在对象释放后置为 nil，避免错误的内存访问。**用更通俗的话来表述是：weak 可以在不增加对象的引用计数的同时，又使得指针的访问是安全的。

weak singleton

之前见过一篇文章介绍了一个新 pattern 叫 「weak singleton」，原文出处点我 (http://www.ios-blog.co.uk/tutorials/objective-c-ios-weak-singletons/)。这种特殊的单例有一个有意思的特性：在所有使用该单例的对象都释放后，单例对象本身也会自己释放。我所见过的大部分单例使用场景，被创建都单例最后都会一直存活着，比如注册登录模块所需要共享

状态所创建的 XXLoginManager，即使在用户注册成功进入主界面之后也不会被显式的释放，这在一定程度上会带来内存使用的浪费。所谓的「weak singleton」代码很简单：

```
+ (id)sharedInstance
{
    static __weak ASingletonClass *instance;
    ASingletonClass *strongInstance = instance;
    @synchronized(self) {
        if (strongInstance == nil) {
            strongInstance = [[[self class] alloc] init];
            instance = strongInstance;
        }
    }
    return strongInstance;
}
```

Controller A， B， C 都可以持有 ASingletonClass 的强引用，一旦 A，B，C 都销毁后，ASingletonClass 的单例对象也会随之销毁，略巧妙不是吗?

「weak singleton」这个漂亮名字背后其实只是简单而巧妙的利用了 weak 特性，sharedInstance 中的 weak 就像是一个智能管家，在无人使用 instance 之后就置为 nil 销毁，当 sharedInstance 再次被调用时，instance 又会重新被创建。

weak associated object

当我们需要给已有的功能模块添加新功能特性的时候，比如给所有的 UIViewController 添加一个 dumpViewHierarchy 方法，可以把当前 Controller 的 view 结构完整保存来下并上报服务器，我们有几种思路可供选择：

方案一：定义一个新的父类 DumpViewController，继承该父类的子类可以获得 dumpViewHierarchy 方法。

方案二：定义一个新的 DumpViewObject 类，已有的 Controller 只需要创建一个 DumpViewObject 对象，并调用 dumpViewHierarchy 方法，传入 self 即可。

方案三：给已有的 Controller 类添加一个 Category，XXController + DumpView，并在 Category 中实现 dumpViewController 方法，有时候我们还需要做一些状态保存，所以扩展性更好的办法是使用 associated object 给 Category 添加一个 DumpViewObject property，将 dumpView 相关的逻辑都写入 DumpViewObject 类中。

方案四：使用 AOP 的方式，利用 Objective C 的 rumtime 特性 hook 每个 Controller 的 dumpViewHierarchy 方法，并在当中实现相应逻辑。

方案一，二都对已有代码改动较大，方案四改动最小，神不知鬼不觉，dumpViewHierarchy 方法甚至可以不出现在 Controller 里面，但这也导致代码管理上比较松散。方案三是我个人比较推崇的方式，代码侵入少，同时方法调用逻辑也会出现在合适的地方，不少知名的第三方库都使用过这种方式来添加功能，比如 facebook 开源的 FBKVOController，就通过 associated object 的方式给每个 NSObject 对象添加了一个功能属性。

使用 associated object 的时候，有一些细节需要额外考虑。比如 property 是强引用还是弱引用，这个选择题取决于代码结构的设计。如果是强引用，则对象的生命周期跟随所依附的对象，XXController dealloc 的时候，DumpViewObject 也随之 dealloc。如果是弱引用，则说明 DumpViewObject 对象的创建会销毁由其他对象负责，一般是为了避免存在循环引用，或者由于 DumpViewObject 的职责多于所依附对象的需要，DumpViewObject 有更多的状态需要维护处理。

associated object 本身并不支持添加具备 weak 特性的 property，但我们可以通过一个小技巧来完成：

```
- (void)setContext:(CDDContext*)object {
    id __weak weakObject = object;
    id (^block)() = ^{ return weakObject; };
    objc_setAssociatedObject(self, @selector(context), block, OBJC_
}

- (CDDContext*)context {
    id (^block)() = objc_getAssociatedObject(self, @selector(contex
    id curContext = (block ? block() : nil);
    return curContext;
}
```

添加了一个中间角色 block，再辅以 weak 关键字就实现了具备 weak 属性的 associated object。这种做法也印证了软件工程里一句名言「We can solve any problem by introducing an extra level of indirection」。

类似的用法还有不少，比如 NSArray，NSDictionary 中的元素引用都是强引用，但我们可以通过添加一个中间对象 WeakContainer，WeakContainer 中再通过 weak property 指向目标元素，这样就能简单的实现一个元素弱引用的集合类。

编程语言一直处于进化当中，语言的设计者会站在宏观的角度，结合行业的需要，添加更多的方便特性，如果只是记住官方文档里的几个应用场景，而不去思考背后的设计思路，则很难写出有想象力的代码。

上一篇
懒惰三问 **(/blog/lazy/)**

下一篇
iOS 真机访问 sandbox 目录的各种姿势 **(/blog/ios-sandbox-file/)**