详解苹果的黑魔法 - KVO 的奥秘

2015-12-15 06:17 编辑：lansekuangtu 分类：iOS开发 来源：Sindri的小巢 投稿 0 192

KVO iOS

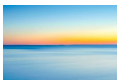
招聘信息：iOS开发高级工程师

作者：Sindri

前言

在iOS开发中，苹果提供了许多机制给我们进行回调。KVO(key-value-observing)是一种十分有趣的回调机制，在某个对象注册监听者后，在被监听的对象发生改变时，对象会发送一个通知给监听者，以便监听者执行回调操作。最常见的KVO运用是监听scrollView的contentOffset属性，来完成用户滚动时动态改变某些控件的属性实现效果，包括渐变导航栏、下拉刷新控件等效果。

热门资讯


如何让iOS 保持界面流畅？这些技巧你知道吗
点击量 13677


【资源集合】94个iOS开发资源推荐，帮你加
点击量 8943


对 Xcode 菜单选项的详细探索（干货）
点击量 6448

苹果Swift编程语言开源 有望进一步普及
点击量 6442


经验之谈！35岁程序员的独家面试经历
点击量 5917


Swift 3 API 设计准则
点击量 5880


iOS 事件处理机制与图像渲染过程
点击量 5716


一款Loading动画的实现思路（一）：拆分复
点击量 5599

Swift中编写单例的正确方式
点击量 5398

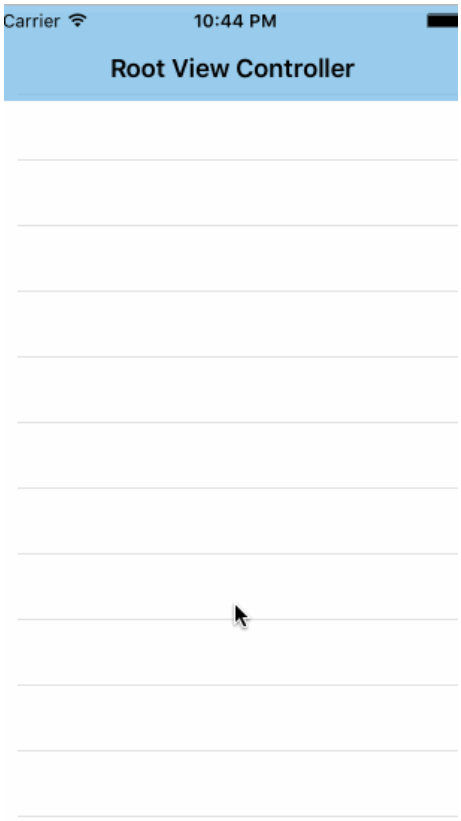

Swift学习: 从Objective-C到Swift
点击量 5031

综合评论

mark
121 评论了 【资源集合】94个iOS开发资源推荐，帮你加速应用...

为什么这么好的帖子却没有什么评论，虽然我是新手，好多看不懂
goldWave 评论了 扒一扒知乎上的帖子——“为什么有些大公司技术弱爆了...

mark
自来也大人 评论了 Objective-C 引用计数：不讲用法，只说原...



渐变导航栏

使用

KVO的使用非常简单，使用KVO的要求是对象必须能支持kvc机制——所有NSObject的子类都支持这个机制。拿上面的渐变导航栏做??，我们为tableView添加了一个监听者controller，在我们滑动列表的时候，会计算当前列表的滚动偏移量，然后改变导航栏的背景色透明度。

```
//添加监听者
[self.tableView addObserver: self forKeyPath: @"contentOffset" options: NSKeyVa
/**
 *  监听属性值发生改变时回调
 */
- (void)observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object change:(
{
    CGFloat offset = self.tableView.contentOffset.y;
    CGFloat delta = offset / 64.f + 1.f;
    delta = MAX(0, delta);
    [self alphaNavController].barAlpha = MIN(1, delta);
}
```

毫无疑问，kvo是一种非常便捷的回调方式，但是编译器是怎么完成监听这个任务的呢？先来看看苹果文档对于KVO的实现描述

Automatic key-value observing is implemented using a technique called isa-swizzling... When an observer is registered for an attribute of an object the isa pointer of the observed object is modified, pointing to an intermediate class rather than at the true class ..

简要的来说，在我们对某个对象完成监听的注册后，编译器会修改监听对象（上文中的tableView）的isa指针，让这个指针指向一个新生成的中间类。从某个意义上来说，这是一场骗局。

```
typedef struct objc_class *Class;
typedef struct objc_object {
    Class isa;
```

```
} *id;
```

这里要说明的是isa这个指针，isa是一个Class类型的指针，用来指向父类。在oc中，规定了只要拥有isa指针的变量，通通都属于对象。上面的objc_object表示的是NSObject这个类的结构体表示，因此oc不允许出现非NSObject子类的对象（block是一个特殊的例外）

当然了，苹果并不想讲述更多的实现细节，但是我们可以通过运行时机制来完成一些有趣的调试。

苹果的黑魔法

根据苹果的说法，在对象完成监听注册后，修改了被监听对象的某些属性，并且改变了isa指针，那么我们可以在监听前后输出被监听对象的相关属性来进一步探索kvo的原理。为了保证能够得到对象的真实类型，我使用了object_getClass方法（class方法本质上是调用这个函数），这个方法在runtime.h头文件中

```
NSLog(@"address: %p", self.tableView);
NSLog(@"class method: %@", self.tableView.class);
NSLog(@"description method: %@", self.tableView);
NSLog(@"use runtime to get class: %@", object_getClass(self.tableView));
[self.tableView addObserver: self forKeyPath: @"contentOffset" options: NSKeyVa
NSLog(@"==========================================================");
NSLog(@"address: %p", self.tableView);
NSLog(@"class method: %@", self.tableView.class);
NSLog(@"description method: %@", self.tableView);
NSLog(@"use runtime to get class %@", object_getClass(self.tableView));
```

在看官们运行这段代码之前，可以先思考一下上面的代码会输出什么。

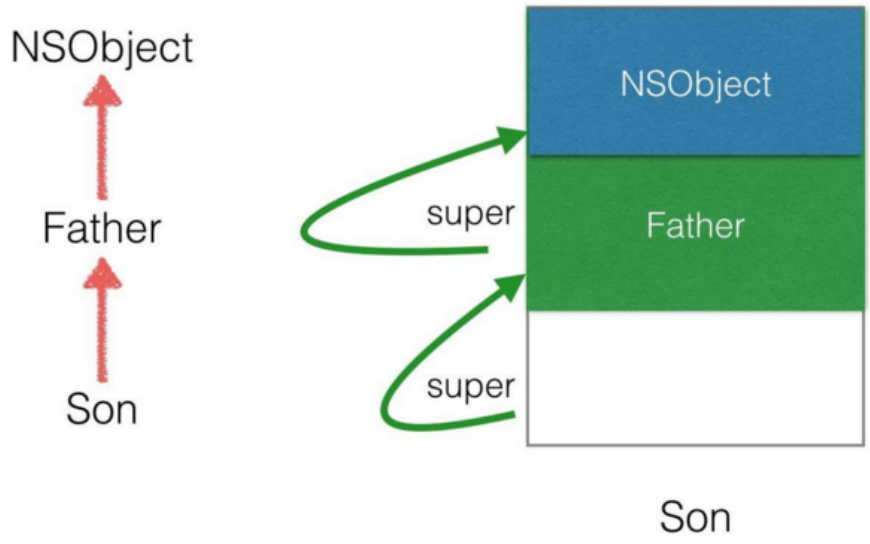
```
2015-12-12 23:02:33.216 LXDAlphaNavigationController[1487:63171] address: 0x7f92
2015-12-12 23:02:33.216 LXDAlphaNavigationController[1487:63171] class method: U
2015-12-12 23:02:33.217 LXDAlphaNavigationController[1487:63171] description met
2015-12-12 23:02:33.217 LXDAlphaNavigationController[1487:63171] ==============
2015-12-12 23:02:33.218 LXDAlphaNavigationController[1487:63171] address: 0x7f92
2015-12-12 23:02:33.218 LXDAlphaNavigationController[1487:63171] class method: U
2015-12-12 23:02:33.218 LXDAlphaNavigationController[1487:63171] description met
```

除了通过object_getClass获取的类型之外，其他的输出没有任何变化。class方法跟description方法可以重写实现上面的效果，但是为什么连地址都是一样的。

这里可以通过一句小代码来说明一下：

```
NSLog(@"%@, %@", self.class, super.class);
```

上面这段代码不管你怎么输出，两个结果都是一样的。这是由于super本质上指向的是父类内存。这话说起来有点绕口，但是我们可以通过对象内存图来表示：



类的内存

每一个对象占用的内存中，一部分是父类属性占用的；在父类占用的内存中，又有一部分是父类的父类占用的。前文已经说过isa指针指向的是父类，因此在这个图中，Son的地址从Father开始，Father的地址从NSObject开始，这三个对象内存的地址都是一样的。通过这个，我们可以猜到苹果文档中所提及的中间类就是被监听对象的子类。并且为了隐藏实现，苹果还重写了这个子类的class方法跟description方法来掩人耳目。另外，我们还看到了新类相对于父类添加了一个NSKVONotifying_前缀，添加这个前缀是为了避免多次创建监听子类，节省资源

怎么实现类似效果

既然知道了苹果的实现过程，那么我们可以自己动手通过运行时机制来实现KVO。runtime允许我们在程序运行时动态的创建新类、拓展方法、method-swizzling、绑定属性等等这些有趣的事情。

在创建新类之前，我们应该学习苹果的做法，判断当前是否存在这个类，如果不存在我们再进行创建，并且重新实现这个新类的class方法来掩盖具体实现。基于这些原则，我们用下面的方法来获取新类

```
- (Class)createKVOClassWithOriginalClassName: (NSString *)className
{
    NSString * kvoClassName = [kLXDkvoClassPrefix stringByAppendingString: clas
    Class observedClass = NSClassFromString(kvoClassName);
    if (observedClass) { return observedClass; }
    //创建新类，并且添加LXDObserver_为类名新前缀
    Class originalClass = object_getClass(self);
    Class kvoClass = objc_allocateClassPair(originalClass, kvoClassName.UTF8Str
    //获取监听对象的class方法实现代码，然后替换新建类的class实现
    Method classMethod = class_getInstanceMethod(originalClass, @selector(class
    const char * types = method_getTypeEncoding(classMethod);
    class_addMethod(kvoClass, @selector(class), (IMP)kvo_Class, types);
    objc_registerClassPair(kvoClass);
    return kvoClass;
}
```

另外，在判断是否需要中间类来完成监听的注册前，我们还要判断监听的属性的有效性。通过获取变量的setter方法名（将首字母大写并加上前缀set），以此来获取setter实现，如果不存在实现代码，则抛出异常使程序崩溃。

```
SEL setterSelector = NSSelectorFromString(setterForGetter(key));
Method setterMethod = class_getInstanceMethod([self class], setterSelector);
if (!setterMethod) {
    @throw [NSException exceptionWithName: NSInvalidArgumentException reason: [
    return;
}
```

```
Class observedClass = object_getClass(self);
NSString * className = NSStringFromClass(observedClass);
//如果被监听者没有LXDObserver_，那么判断是否需要创建新类
if (![className hasPrefix: kLXDkvoClassPrefix]) {
    observedClass = [self createKVOClassWithOriginalClassName: className];
    object_setClass(self, observedClass);
}
//重新实现setter方法，使其完成
const char * types = method_getTypeEncoding(setterMethod);
class_addMethod(observedClass, setterSelector, (IMP)KVO_setter, types);
```

在重新实现setter方法的时候，有两个重要的方法：willChangeValueForKey和didChangeValueForKey，分别在赋值前后进行调用。此外，还要遍历所有的回调监听者，然后通知这些监听者：

```
static void KVO_setter(id self, SEL _cmd, id newValue)
{
    NSString * setterName = NSStringFromSelector(_cmd);
    NSString * getterName = getterForSetter(setterName);
    if (!getterName) {
        @throw [NSException exceptionWithName: NSInvalidArgumentException reaso
        return;
    }
    id oldValue = [self valueForKey: getterName];
    struct objc_super superClass = {
        .receiver = self,
        .super_class = class_getSuperclass(object_getClass(self))
    };
    [self willChangeValueForKey: getterName];
    void (*objc_msgSendSuperKVO)(void *, SEL, id) = (void *)objc_msgSendSuper;
    objc_msgSendSuperKVO(&superClass, _cmd, newValue);
    [self didChangeValueForKey: getterName];
    //获取所有监听回调对象进行回调
    NSMutableArray * observers = objc_getAssociatedObject(self, (__bridge const
    for (LXD_ObserverInfo * info in observers) {
        if ([info.key isEqualToString: getterName]) {
            dispatch_async(dispatch_queue_create(DISPATCH_QUEUE_PRIORITY_DEFAUL
            info.handler(self, getterName, oldValue, newValue);
            });
        }
    }
}
```

所有的监听者通过动态绑定的方式将其存储起来，但这样也会产生强引用，所以我们还需要提供释放监听的方法：

```
- (void)LXD_removeObserver:(NSObject *)object forKey:(NSString *)key
{
    NSMutableArray * observers = objc_getAssociatedObject(self, (__bridge void
    LXD_ObserverInfo * observerRemoved = nil;
    for (LXD_ObserverInfo * observerInfo in observers) {
        if (observerInfo.observer == object && [observerInfo.key isEqualToStrin
            observerRemoved = observerInfo;
            break;
        }
    }
    [observers removeObject: observerRemoved];
}
```

虽然上面已经粗略的实现了kvo，并且我们还能自定义回调方式。使用target-action或者block的方式进行回调会比单一的系统回调要全面的多。但kvo真正的实现并没有这么简单，上述代码目前只能实现对象类型的监听，基本类型无法监听，况且还有keyPath可以监听对象的成员对象的属性这种更强大的功能。

尾言

对于基本类型的监听，苹果可能是通过void *类型对对象进行桥接转换，然后直接获取内存，通过type encoding我们可以获取所有setter对象的具体类型，虽然实现比较麻烦，但是确实能够达成类似的效果。

钻研kvo的实现可以让我们对苹果的代码实现有更深层次的了解，这些知识涉及到了更深层次的技术，探究它们对我们的开发视野有着很重要的作用。同时，对比其他的回调方式，KVO的实现在创建子类、重写方法等等方面的内存消耗是很巨大的，因此博主更加推荐使用delegate、block等回调方式，甚至直接使用method-swizzling来替换这种重写setter方式也是可行的。

文章代码：自实现KVO

微信扫一扫

订阅每日移动开发及APP推广热点资讯

公众号：CocoaChina

我要投稿 收藏文章 分享到：

1

上一篇：源码推荐(12.14B)：无限图片轮播器，加载时日食效果

相关资讯

源码推荐(12.14)：iOS mansory纯代码自动计算cell高度，	【资源集合】94个iOS开发资源推荐，帮你加速应用开发
手把手教你如何分析 iOS 系统栈 crash	利用OC的消息转发机制实现多重代理
36氪客户端4.0版本开发架构总结: 第三方分享认证模块的	iOS新建项目架构规范
【译】自动更新订阅IAP浅谈（设置和测试）	iOS微信安装包瘦身
iOS 事件处理机制与图像渲染过程	如何跳到系统设置里的各种设置界面

我来说两句

你怎么看？快来评论一下吧！

发表评论

所有评论（0）