作者 树下的老男孩 (/users/811a70f4726a) 2015.06.05 23:13*
写了14833字，被127人关注，获得了173个喜欢
(/users/811a70f4726a)

添加关注 (/sign_in)

ios runtime浅析(三)：Method Swizzling

字数1558 阅读1056 评论2 喜欢12

看到nshipster的Method Swizzling (https://github.com/AFNetworking/AFNetworking/)这篇不错的文章还没翻译，就补充一下，没有逐字翻译，关于associated objects (http://nshipster.cn/associated-objects/)已经有翻译了，大家也可以去了解一下。

method swizzling也许是runtime中最有争议的技术，它的作用就是改变已经存在selector的实现，之所以可以这样是因为方法调用可以在运行时改变：通过改变类的分发表(dispatch table，该表包含selector的名称及对应实现函数的地址)里selector和实现之间的对应关系。

举个例子，比如你想记录一个iOS应用里每个view controller显示的次数：可以在每个view controller添加记录的代码，但这会导致大量的重复代码；通过继承也是一个方法，但需要同时创建UIViewController, UITableViewController, UINavigationController及其它中view controller的子类，同样也会产生许多重复的代码出现。

幸运的是，在UIViewController的category使用method swizzling：

```
#import <objc/runtime.h>

@implementation UIViewController (Tracking)

+ (void)load {
    static dispatch_once_t onceToken;
    dispatch_once(&onceToken, ^{
        Class class = [self class];

        SEL originalSelector = @selector(viewWillAppear:);
        SEL swizzledSelector = @selector(xxx_viewWillAppear:);

        Method originalMethod = class_getInstanceMethod(class, originalSelector);
        Method swizzledMethod = class_getInstanceMethod(class, swizzledSelector);

        // 如果 swizzling 的是类方法，采用如下的方式:
        // Class class = object_getClass((id)self);
        // ...
        // Method originalMethod = class_getClassMethod(class, originalSelector);
        // Method swizzledMethod = class_getClassMethod(class, swizzledSelector);

        //交换实现
        method_exchangeImplementations(originalMethod, swizzledMethod);
    });
}

#pragma mark - Method Swizzling

- (void)xxx_viewWillAppear:(BOOL)animated {
    [self xxx_viewWillAppear:animated];
    NSLog(@"viewWillAppear: %@", self);
}

@end
```

简
(/)

(/collections)

(/apps)

现在，当一个UIViewController或者其子类的实例调用viewWillAppear:方法时，就会打印出一条记录。假如要在在view controller的生命周期，view的绘制或者Foundation的网络协议栈注入一些自定的行为，method swizzling也许是你应该考虑的一个方向。

下面是使用method swizzling应该注意的点：

+load vs. +initialize

Swizzling应该只在load方法中使用

oc会在运行时自动调用每个类的两个方法，+load 会在类初始化加载的时候调用；+initialize方法会在程序调用类的第一个实例或者类方法的时候调用。这两个方法都是可选的，只会在实现的时候才去调用。由于method swizzling会影响到全局的状态，因此最小化竞争条件的出现变得很重要，+load方法能够确保在类的初始化时候调用，这能够保证改变应用行为的一致性，而+initialize在执行时并不提供这种保证，实际上，如果没有直接给这个类发送消息，该方法可能都不会调用到。

dispatch_once

Swizzling应该只在dispatch_once中完成

如上，由于swizzling会改变全局状态，所以我们需要在运行时采取一些预防措施。原子性就是其中的一种预防措施，因为它能保证不管有多少个线程，代码只会执行一次。GCD的dispatch_once 能够满足这种需求，因此在method swizzling应该将其作为最佳的实践方式。

选择器，方法和实现

在oc中，选择器、方法和实现是运行时的特殊方面，虽然在一般情况下，这些术语是用在消息发送的过程中。

下面是Apple对它们的几个描述：

- 选择器(Selector-typedef struct objc_selector *SEL)：用于在运行时表示一个方法的名称，一个方法选择器就是一个C字符串，在运行时会被注册或者映射，选择器是由编译器生成的，并在类被加载的时候由运行时自动进行映射。
- 方法(Method-typedef struct objc_method *Method)：在类的定义中代表一个方法的类型。
- 实现(Implementation- typedef id (*IMP)(id, SEL, ...))：这是一个指向方法实现函数起始地址的指针，这个函数的第一个参数是指向self的指针，第二个参数是方法选择器，然后是方法的参数。

  理解它们之间关系的最好方式是：一个类维护着一张分发表(dispatch table)，用来解析运行时发来的消息；该表的每个入口是一个方法(Method)，其中的key就是选择器(SEL)，对应一个实现(IMP)，即一个指向底层c函数的指针。

  swizzle 一个方法就是改变类的分发表，使它在解析消息的时候，将一个选择器selector对应到别的实现，并且将该选择器对应的原始实现关联到新的选择器中。

调用 _cmd

看起来下面的代码可能导致无限循环：

```
- (void)xxx_viewWillAppear:(BOOL)animated {
    [self xxx_viewWillAppear:animated];
    NSLog(@"viewWillAppear: %@", NSStringFromClass([self class]));
}
```

可奇怪的是，它并不会。在swizzling的过程中，xxx_viewWillAppear:已经被重新指向 UIViewController 的原始实现-viewWillAppear:，但是如果我们在这个方法中调用viewWillAppear:则会导致无限循环。

注意事项

通常认为Swizzling是一个比较危险的技术，容易产生不可预料的行为和无法预见的后果，但只要遵循以下几个注意事项，其实method swizzlin还是相对安全的。

- 总是调用一个方法的原始实现(除非你有足够好的理由不这么做):API提供了输入和输出的约定，但其中的实现却是黑盒。Swizzling 一个方法但不去调用其原始实现，可能造成私有状态的底层假设被打破，影响程序的其它部分。
- 避免冲突: 给category方法加前缀，确保不会跟其它依赖的代码产生冲突。
- 知道到底发生啥了：简单的复制粘贴swizzling 代码，而不清楚其如何工作不仅非常危险，而且浪费了好多深入学习objective-c运行时的机会，可以通过查看Objective-C Runtime Reference (https://developer.apple.com/library/mac/documentation/Cocoa/Reference/ObjCRuntimeRef/Reference/reference.html#//apple_ref/c/func/method_getImplementation) 和<objc/runtime.h>头文件了解其中的一些来龙去脉。
- 小心的处理：不管你在swizzling Foundation、UIKit或者其它内建framework方法时多么的充满自信，必须清楚在下一个版本这些可能都改变了，所以为了不出差错，还是需要多花点心思的。

推荐拓展阅读 (/sign_in)

如果觉得我的文章对您有用，请随意打赏。您的支持将鼓励我继续创作！

¥ 打赏支持

♡ 喜欢 | 12

分享到微博 分享到微信
更多分享

2条评论 （ 按时间正序 · 按时间倒序 · 按喜欢排序 ）

添加新评论 (/sign_in)

王大先森 (/users/18d63c18a2f2)
3楼 · 2015.09.07 18:07 (/p/b87fa689055f/comments/569450#comment-569450)
很棒，谢谢你，加油

♡ 喜欢(1) 回复

王大先森 (/users/18d63c18a2f2)
4楼 · 2015.09.07 18:10 (/p/b87fa689055f/comments/569458#comment-569458)
把那两个方法给交换后，那系统原来的viewWillAppear还会执行，这是为何呢？我感觉给它替换之后就是给他替代了吧，它不应该再走了，但是现在在它里面写的一些方法还是会执行的，为什么呢

♡ 喜欢(0) 回复

登录后发表评论 (/sign_in)

被以下专题收入，发现更多相似内容：

iOS Developer (/collection/3233d1a249ca)
分享 iOS 开发的知识，解决大家遇到的问题，讨论iOS开发的前沿，欢迎大家投稿~ 添加关注 (/sign_in)
480篇文章 (/collection/3233d1a249ca) · 5697人关注

iOS开发技巧 (/collection/19dbe28002a3)
【简介】 专题内容主要包括OC、swift等涉及到iOS开发进阶的内容。 【投稿需知】
为了保证专题订阅者有一个好的阅读环... + 添加关注 (/sign_in)

352篇文章 (/collection/19dbe28002a3) · 4882人关注

iOS开发 (/collection/1cb96ea7b540)

157篇文章 (/collection/1cb96ea7b540) · 2591人关注

添加关注 (/sign_in)

(/collection/1cb96ea7b540)