


# iOS RSSwizzle中的swizzle原理

 某某香肠 关注

 0.719 2019.08.25 20:58:03 字数 2,053 阅读 1,110

RSSwizzle 是一个简单的 hook 函数的第三方库，它的使用跟传统的 hook 方式比起来更加便捷，也更加安全。下面来分析它是怎么做到的。

## 传统的hook方法

### 实现

一般的，如果我们要 viewDidLoad ,我们需要写如下的代码：

```
1 + (void)load {
2     static dispatch_once_t onceToken;
3     dispatch_once(&onceToken, ^{
4         SEL originalSel = @selector(viewDidLoad);
5         SEL swizzleSel = @selector(swizzle_viewDidLoad);
6         Method originalMethod =
7             class_getInstanceMethod([self class], originalSel);
8         Method swizzleMethod = class_getInstanceMethod([self class], swizzleSel);
9         1.
10        BOOL didAddMethod = class_addMethod(
11            [self class], originalSel, method_getImplementation(swizzleMethod),
12            method_getTypeEncoding(swizzleMethod));
13        if (didAddMethod) {
14            2.
15            class_replaceMethod([self class], swizzleSel,
16                               method_getImplementation(originalMethod),
17                               method_getTypeEncoding(originalMethod));
18        } else {
19            3.
20            method_exchangeImplementations(originalMethod, swizzleMethod);
21        }
22    });
23 }
24
25 - (void)viewDidLoad {
26     [super viewDidLoad];
27 }
28
29 - (void)swizzle_viewDidLoad {
30     [self swizzle_viewDidLoad];
31     NSLog(@"hook viewDidLoad");
32 }
```

如上所示，主要的逻辑都在load函数里面，其核心思路就是交换 viewDidLoad 和 `swizzle\_viewDidLoad` 的实现，下面简单解析一下：

1. 尝试给当前类添加一个方法，该方法的方法名是 viewDidLoad ，实现是 swizzle\_viewDidLoad 的实现，这么做的目的是为了确保当前类一定有 viewDidLoad 这个方法名（否则使用 method\_exchangeImplementations交换不会成功）
2. 添加成功，则将原来的 swizzle\_viewDidLoad 的实现替换换成viewDidLoad的实现


2. 如果添加不成功，则交换两个方法的实现

写下你的评论...

评论0 赞7 ...



工业平板电脑

 某某香肠 关注

总资产25 (约1.43元)

iOS 动态库注入原理  
阅读 66

RSA技术原理详解  
阅读 21

FBKVOController源码解析  
阅读 240

### 推荐阅读

- 整理一份Swift面试题  
阅读 547
- 如何手写一个call方法  
阅读 272
- [iOS]三个问题全面理解runtime  
阅读 644
- iOS Runtime! !!  
阅读 275
- iOS Block基本使用(一)  
阅读 152



云呼叫中心



## 不足之处

这样写，大部分情况都是可以实现hook的，但是还是有一些边界情况没有考虑进去，比如 originalSel 在本类和父类都没有实现的情况，可以参考[这篇文章](#)。另外，没有一个hook就会要多写一个方法，写法上也不是很好。还有就是，hook的代码一旦不是写在 load 函数里面（一般不会出现这种情况），则还要考虑多线程的问题。

## RSSwizzle

RSSwizzle能规避上述的问题，如果要hook一个函数，不管在什么地方，只需要这么写：

```
1 static dispatch_once_t onceToken;
2 dispatch_once(&onceToken, ^{
3     RSSwizzleInstanceMethod([self class], @selector(viewDidLoad), RSSWReturnType(void), RS
4         RSSWCallOriginal();
5     NSLog(@"hook viewDidLoad");
6     }), RSSwizzleModeAlways, NULL)
7 });
```

其中 `RSSwizzleInstanceMethod` 就是交换方法的宏，除了要传 viewDidLoad 之外，还要传入方法的返回参数和方法的参数，在 block 里面，就是替换的实现，其中 `RSSWCallOriginal` 是另一个宏，就是调用原来的方法。

可以看出，这样调用比原来的方式要简洁多了。

## RSSwizzle代码实现

在 RSSwizzle.h 文件中，定义了两个类，`RSSwizzleInfo` 用于保存原函数的实现，`RSSwizzle` 则是 swizzle 的主要类，其中有两个方法

```
1 +(BOOL)swizzleInstanceMethod:(SEL)selector
2     inClass:(Class)classToSwizzle
3     newImpFactory:(RSSwizzleImpFactoryBlock)factoryBlock
4     mode:(RSSwizzleMode)mode
5     key:(const void *)key;
6
7 +(void)swizzleClassMethod:(SEL)selector
8     inClass:(Class)classToSwizzle
9     newImpFactory:(RSSwizzleImpFactoryBlock)factoryBlock;
```

看函数名可以知道，这两个方法一个是针对类方法的 swizzle 一个是针对实例方法的 swizzle。先看一下针对类方法的实现：

```
1 +(void)swizzleClassMethod:(SEL)selector
2     inClass:(Class)classToSwizzle
3     newImpFactory:(RSSwizzleImpFactoryBlock)factoryBlock
4 {
5     [self swizzleInstanceMethod:selector
6         inClass:object_getClass(classToSwizzle)
7         newImpFactory:factoryBlock
8         mode:RSSwizzleModeAlways
9         key:NULL];
10 }
```



```

1  +(BOOL)swizzleInstanceMethod:(SEL)selector
2      inClass:(Class)classToSwizzle
3      newImpFactory:(RSSwizzleImpFactoryBlock)factoryBlock
4      mode:(RSSwizzleMode)mode
5      key:(const void *)key
6  {
7      NSAssert(!(NULL == key && RSSwizzleModeAlways != mode),
8          @"Key may not be NULL if mode is not RSSwizzleModeAlways.");
9
10     @synchronized(swizzledClassesDictionary()){
11
12         if (key){
13             1.
14             NSMutableSet *swizzledClasses = swizzledClassesForKey(key);
15             if (mode == RSSwizzleModeOncePerClass) {
16                 if ([swizzledClasses containsObject:classToSwizzle]){
17                     return NO;
18                 }
19             }else if (mode == RSSwizzleModeOncePerClassAndSuperclasses){
20                 for (Class currentClass = classToSwizzle;
21                     nil != currentClass;
22                     currentClass = class_getSuperclass(currentClass))
23                 {
24                     if ([swizzledClasses containsObject:currentClass]) {
25                         return NO;
26                     }
27                 }
28             }
29         }
30         2.
31         swizzle(classToSwizzle, selector, factoryBlock);
32
33         if (key){
34             [swizzledClassesForKey(key) addObject:classToSwizzle];
35         }
36     }
37     return YES;
38 }
39

```

1. `RSSwizzleMode` 是一个枚举，用于决定这次hook是否能hook多次还是只能hook一次（或者是父类hook一次），它会根据 `key` 对应的集合是否有当前要hook的类决定是否使用这次hook，通常在开发中mode会传 `RSSwizzleModeAlways`，`key` 会传NULL，因此代码会直接走到2这边来。**PS**感觉这个功能比较鸡肋，如果别的模块使用传统的swizzle方法还是会hook住的，实在想不到应用场景。
2. 这是swizzle的核心方法，`RSSwizzleImpFactoryBlock` 的定义如下：

```
1 | typedef id (^RSSwizzleImpFactoryBlock)(RSSwizzleInfo *swizzleInfo);
```

`swizzle` 方法的实现如下：

```

1 | static void swizzle(Class classToSwizzle,
2 |                     SEL selector,
3 |                     RSSwizzleImpFactoryBlock factoryBlock)
4 | {
5 |     1.
6 |     Method method = class_getInstanceMethod(classToSwizzle, selector);
7 |
8 |     NSAssert(NULL != method,
9 |         @"Selector %@ not found in %@ methods of class %@.",

```



写下你的评论...

评论0

赞7

```
15     NSAssert(blockIsAnImpFactoryBlock(factoryBlock),
16               @"Wrong type of implementation factory block.");
17
18     3.
19     __block OSSpinLock lock = OS_SPINLOCK_INIT;
20     __block IMP originalIMP = NULL;
21
22     4.
23     RSSwizzleImpProvider originalImpProvider = ^IMP{
24
25         OSSpinLockLock(&lock);
26         IMP imp = originalIMP;
27         OSSpinLockUnlock(&lock);
28
29         if (NULL == imp){
30             Class superclass = class_getSuperclass(classToSwizzle);
31             imp = method_getImplementation(class_getInstanceMethod(superclass,selector));
32         }
33         return imp;
34     };
35
36     5.
37     RSSwizzleInfo *swizzleInfo = [RSSwizzleInfo new];
38     swizzleInfo.selector = selector;
39     swizzleInfo.impProviderBlock = originalImpProvider;
40
41     6.
42     id newIMPBlock = factoryBlock(swizzleInfo);
43
44     const char *methodType = method_getTypeEncoding(method);
45
46     NSAssert(blockIsCompatibleWithMethodType(newIMPBlock,methodType),
47               @"Block returned from factory is not compatible with method type.");
48
49     IMP newIMP = imp_implementationWithBlock(newIMPBlock);
50
51     7.
52     OSSpinLockLock(&lock);
53     originalIMP = class_replaceMethod(classToSwizzle, selector, newIMP, methodType);
54     OSSpinLockUnlock(&lock);
55 }
```

#### 1. 获取原方法的method

2. 确保factoryBlock是否是RSSwizzleImpFactoryBlock，如果传的是 `id(NSObject *obj){}` 类型的block编译器不会报错，但在这里会进断言，其实就是判断二者的签名是否一致，这里就不展开讲了，有兴趣可以看[这篇文章](#)

3. `originalIMP` 是原来方法的实现，目前是NULL,后续会给它赋值，它可能是线程不安全的（因为引用它的block不知道会被哪个线程调用），因此需要加锁保护

4. `originalImpProvider` 就是返回一个原来方法的实现，如果本类没有，还会往父类那里找直到找到为止。这是因为7中的 `class_replaceMethod` 只会返回本类的实现，不会再父类中查找

5. 初始化 `RSSwizzleInfo`，给它赋值

6. 调用 `factoryBlock`，拿到新的block,然后比较这个block跟原函数的函数签名是否一致，否则进断言，最后用这个block初始化 `newIMP`

7. `class_replaceMethod` 替换原来的实现，如果本类没有这个方法，会默认加上这个替换的方法，返回的 `originalIMP` 就是原来的实现

从6可以看出，传入的 `factoryBlock` 的返回只能是一个block，否则这逻辑走不通，这个函数执行完之后，调用原来的函数，就执行了 `newIMP`

此外，如果想在替换的方法里面调用原来的函数，我们就需要在 `RSSwizzleInfo` 那里拿到原方法并调用了，主要函数如下：



不难看出，其实就是调用4中的block取得IMP而已

如果只是通过 `swizzleInstanceMethod` 这个函数来实现swizzle，那么我们必须传对 `RSSwizzleImpFactoryBlock`，否则会进入断言，这样调用还是挺麻烦的，但是它提供的宏解决了这一问题。

## RSSwizzle宏实现

使用 `RSSwizzle` 进行hook的时候会使用到它的很多个宏，下面从它的参数开始说起：

### RSSWReturnType

```
1 | #define RSSWReturnType(type) type
```

这是一个返回值的宏，可以填写可以看出这个宏其实什么都没做，原样返回了，但是这样写增加了代码的易读性

### RSSWArguments

```
1 | #define RSSWArguments(arguments...) _RSSWArguments(arguments)
2 | #define _RSSWArguments(arguments...) DEL, ##arguments
3 |
```

这是一个可以填多个参数的宏，...是多个参数的意思，`arguments` 是这些参数的集合。

`##` 在这里的意思是：如果没有arguments，则删掉`##arguments`和前面的逗号，也就是说，`RSSWArguments()` 最后得到的是 `DEL`，而所有的传参前面都会插入 `DEL` 这个标志位，后续会移除这个标志位，这样做是为了规避没有参数的时候有时预编译会出现多余逗号的bug。

### RSSWReplacement

```
1 | #define RSSWReplacement(code...) code
```

这个宏封装替换的函数

### RSSwizzleInstanceMethod

```
1 | #define RSSwizzleInstanceMethod(classToSwizzle, \
2 |                               selector, \
3 |                               RSSWReturnType, \
4 |                               RSSWArguments, \
5 |                               RSSWReplacement, \
6 |                               RSSwizzleMode, \
7 |                               key) \
8 | _RSSwizzleInstanceMethod(classToSwizzle, \
9 |                          selector, \
10 |                          RSSWReturnType, \
11 |                          _RSSWWrapArg(RSSWArguments), \
12 |                          _RSSWWrapArg(RSSWReplacement), \
13 |                          RSSwizzleMode, \
14 |                          key)
15 |
```



写下你的评论...

评论0

赞7

```

21         RSSwizzleMode, \
22         KEY) \
23     [RSSwizzle \
24     swizzleInstanceMethod:selector \
25     inClass:[classToSwizzle class] \
26     newImpFactory:^(RSSwizzleInfo *swizzleInfo) { \
27         RSSWReturnType (*originalImplementation)(_RSSWDel3Arg(__unsafe_unretained id, \
28         SEL, \
29         RSSWArguments)); \
30         SEL selector_ = selector; \
31         return ^RSSWReturnType (_RSSWDel2Arg(__unsafe_unretained id self, \
32         RSSWArguments)) \
33         { \
34             RSSWReplacement \
35         }; \
36     } \
37     mode:RSSwizzleMode \
38     key:KEY];
39

```

这个是最主要的宏，它封装了整个函数的调用，将其展开那就是：

```

1  [RSSwizzle
2  swizzleInstanceMethod:selector
3  inClass:[classToSwizzle class]
4  newImpFactory:^(RSSwizzleInfo *swizzleInfo) {
5      1.
6          RSSWReturnType (*originalImplementation)(_RSSWDel3Arg(__unsafe_unretained id,
7          SEL,
8          RSSWArguments));
9      2.
10         SEL selector_ = selector;
11      3.
12         return ^RSSWReturnType (_RSSWDel2Arg(__unsafe_unretained id self,
13         RSSWArguments))
14         {
15             RSSWReplacement
16         };
17     }
18     mode:RSSwizzleMode
19     key:KEY];

```

前两个参数都挺好懂，主要看最后一个参数：

1.根据函数参数和返回值声明一个名为 `originalImplementation_` 的block，改block的返回值是 `RSSWReturnType`，也就是传入的返回值，参数值是id,SEL,和 `RSSWArguments` 中传入的参数，`_RSSWDel3Arg` 的宏定义如下：

```
1 | #define _RSSWDel3Arg(a1, a2, a3, args...) a1, a2, ##args
```

这是为了去掉第三个参数，前面说过 `RSSWArguments` 会在参数前面插入一个DEL，就在这里去掉了

2.定义 `selector_` 等于 `selector`，这是原函数的方法编号

3.定义了一个block作返回，这个block返回值是 `RSSWReturnType`，参数是self和 `RSSWArguments` 中传入的参数，这里 `_RSSWDel2Arg` 的意思跟 `_RSSWDel3Arg` 类似，都是除掉多余的DEL,这个block的内容就是替换的函数，从上面的代码分析中我们知道，这个block就是用来初始化 `newIMP` 的。

```
1 #define RSSWCallOriginal(arguments...) _RSSWCallOriginal(arguments)
2
3 #define _RSSWCallOriginal(arguments...) \
4     ((__typeof(originalImplementation_))[swizzleInfo \
5         getOriginalImplementation])(self, \
6                                     selector_, \
7                                     ##arguments)
```

可以看出，它是在 `swizzleInfo` 中拿到imp指针，然后将其强制转换为 `originalImplementation_` 这个block进行调用，这样的好处由于 `originalImplementation_` 的传参和返回值都由外界决定，因此如果传的不对编译器会报错，在一定程度上避免在运行期间进入断言。

有了这些宏之后，hook就变得方便多了

总结

`RSSwizzle` 虽然是个轻量易用的库，总共的代码量不多，但涉及到的知识还是挺多的，其中运行时和宏的相关的代码更是非常的精妙，推荐大家使用。

参考文献

[Method Swizzle实际使用的坑](#)

[RSSwizzle源码解析](#)

[iOS 开发 高级：使用 宏定义macros \(#, ##, ..., \\_\\_VA\\_ARGS\\_\)](#)

👍

7人点赞 >

👎

📖

IOS相关

...

更多精彩内容，就在简书APP



"小礼物走一走，来简书关注我"

赞赏支持

还没有人赞赏，支持一下



某某香肠 哔🐼皮实的iOS低级开发攻城师  
总资产25 (约1.43元) 共写了3.0W字 获得59个赞 共13个粉丝

关注





写下你的评论...

全部评论 0

只看作者

按时间倒序

按时间正序

推荐阅读

更多精彩内容>

快过年了，为过完年跳槽的人准备一份面试题

1.设计模式是什么？ 你知道哪些设计模式，并简要叙述？设计模式是一种编码经验，就是用比较成熟的逻辑去处理某一种类型...

 龍麟


阅读 1,473

评论 0

赞 12

黑暗中只能自己走，谁也给不了你糖！

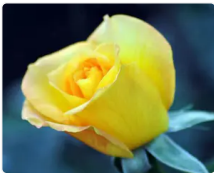
这个世上有很多事是解释不通的。比如突然的失落，莫名其妙的孤独，没有由来的落寞，以及突然冷漠的你。孤独的人...

 墨绿色的纠缠

阅读 68

评论 0

赞 0




有程序员接外包吗?靠谱



2019.1.15

A:从早上开始一直到晚上无休息，晚上开始学习后依然犯困。 M:接受自己晚上精力不足的现状，想办法恢复精力。 B:暂...

 有只熊爱冬眠

阅读 23

评论 0

赞 0

Android图形验证码

先看效果图 给上fragment\_mine.xml布局代码： 再上MineFragment代码： 最后附上Code



写下你的评论...

评论0

赞7



简书

首页 下载APP


beta

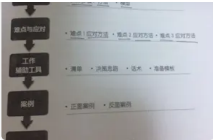
登录

注册

【于千万人之中，遇见天选之你，于亿万人之中，遇见更好的自己】

组织最佳实践，是一个管理学概念，即优秀个人或绩优团队在完成某项任务时所具备的特殊的认知结构、工作方法和技巧。最佳实...

 心跳快门Yo 阅读 3,015 评论 1 赞 11



写下你的评论...

评论0 赞7