iOS 修改私有方法总结

1.Category添加方法

原理： 通过Category添加方法，优先于类中相同方法的特点，进行私有函数修改。 category中的方法位于类方法列表的前面，在进行方法查找时，会先找到category中的方法，所以会优先执行Category中方法的目的，达到拦截原有私有方法的目的。

副作用：

- 不能直接调用 super，除了特殊的 + load 方法外，其他分类中的方法，在被调用之前，并不会去调用原始方法，这将导致外部在使用这个类的时候，永远调用不到原始的实现，尤其是一些系统类内部互相调用的情况，将会无法预估；
- 当项目中存在多个分类复写了同一个私有方法后，最终调用哪个分类中的方法将由编译顺序所决定，这又充满了不确定性，因为分类自身并不知道这个私有方法是否已经被其他分类所复写；
- 一旦通过分类复写私有方法，它的影响将会是全局的，所有用到这个类的地方，都将变成分类调用

2.Method Swizzling

原理： 通过使用runtime方法，交换函数imp地址，在调用原函数是，就变成调用新函数，达到修改的目的。

示例代码：

```
+ (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);

        // When swizzling a class method, use the following:
        // Class class = object_getClass((id)self);
        // ...
        // Method originalMethod = class_getClassMethod(class, originalSelector);
        // Method swizzledMethod = class_getClassMethod(class, swizzledSelector);

        BOOL didAddMethod =
            class_addMethod(class,
                originalSelector,
                method_getImplementation(swizzledMethod),
                method_getTypeEncoding(swizzledMethod));

        if (didAddMethod) {
            class_replaceMethod(class,
                swizzledSelector,
                method_getImplementation(originalMethod),
                method_getTypeEncoding(originalMethod));
        } else {
            method_exchangeImplementations(originalMethod, swizzledMethod);
        }
    });
}
```

副作用：

- 只能执行一次，多次执行会有问题。
- 要避免hook掉函数的父函数。
- hook方法中含有_cmd这种函数，会出现未知问题。
- 不支持实例对象的hook

使用场景：

3.objc_msgForward消息转发

原理： 在 OC 中，向一个对象发送消息后，最终会全部转化到 objc*msgSend 中，继而在方法列表中通过 SEL 开始查找对应的 IMP，如果没有查到对应的方法，则 IMP 会返回 objc*msgForward /

objc*msgForward*stret，从而触发未知消息转发流程，即 resolveInstanceMethod:、forwardingTargetForSelector: 和 forwardInvocation:。

Aspects主要是利用了forwardInvocation进行转发，Aspects其实利用和kvo类似的原理，通过动态创建子类的方式，把对应的对象isa指针指向创建的子类，然后把子类的forwardInvocation的IMP替成**ASPECTS*ARE*BEING_CALLED**，假设要hook的方法名XX,在子类中添加一个Aspects*XX的方法，然后将Aspects*XX的IMP指向原来的XX方法的IMP，这样方便后面调用原始的方法，再把要hook的方法XX的IMP指向*objc*msgForward，这样就进入了消息转发流程，而forwardInvocation的IMP被替换成了**ASPECTS*ARE*BEING_CALLED**，这样就会进入**ASPECTS*ARE*BEING_CALLED**进行拦截处理，这样整个流程大概结束。

示例代码：

```
    [UIViewController aspect_hookSelector:NSSelectorFromString(@"dealloc") withOptio
        NSLog(@"Controller is about to be deallocated: %@", [info instance]);
    } error:NULL];
```

副作用：

- Aspect 中使用了 OSSpinLockLock 来保证线程安全，但自旋锁已经被发现为存在优先级调度的问题
- Aspect 中利用了 sub class 来实现消息转发，但这种继承链消息转发存在缺陷，子类和父类如果同时 hook 一个方法，便会造成死循环MessageThrottle Safety

4.基于libbfi动态调用C函数

原理： 使用libffi中的ffi*closure*alloc构造与原方法参数一致的"函数" -- stingerIMP，以替换原方法函数指针；此外，生成了原方法和Block的调用的参数模板cif和blockCif。方法调用时，最终会调用到void *stffi*function(*ffi*cif *cif, void *ret, void **args, void *userdata), 在该函数内，可获取到方法调用的所有参数、返回值位置，主要通过ffi_call根据cif调用原方法实现和切面block。AOP库 `Stinger和BlockHook` 就是使用libbfi做的。

示例代码：

```
BlockHook使用
    void(^block)(void) = ^() {
        NSLog(@"This is global block!");
    };
    [block block_hookWithMode:BlockHookModeAfter usingBlock:^(BHToken *token) {
        NSLog(@"After global block!");
    }];
```

使用场景： 目前使用场景较多，执行效率较高，Stinger和BlockHook都是基于libbfi进行开发的。

5.基于桥的全量方法 Hook 方案 - TrampolineHook

原理： 把一个我们要替换的原方法 IMP A 取出来，保存起来。 给这个原方法塞一个动态分配的可执行地址 B。 当执行这个原方法的时候，会跳转到 可执行地址 B。 这个 B 经过一段简短的运算操作，可以获取到原先保存的 IMP A。 在跳转回 IMP A 之前，统一拦截函数先做些事情，比如检查是不是主线程调用之类的。但这里的这些操作时使用汇编来进行，执行效率更高。

示例代码：

```
THInterceptor *interceptor = [THInterceptor sharedInterceptorWithFunction:(IMP)myI
nterceptor];
Method m = class_getInstanceMethod([UIView class], @selector(initWithFrame:));
IMP imp = method_getImplementation(m);
THInterceptorResult *interceptorResult = [interceptor interceptFunction:imp];
if (interceptorResult.state == THInterceptStateSuccess) {
    method_setImplementation(m, interceptorResult.replacedAddress); // 设置替换的地址
}
UIView *view = [[UIView alloc] initWithFrame:CGRectMake(0, 0, 200, 200)]; // 执行到
这一行时，会调用 myInterceptor 方法
```

使用场景： 目前作者只编写arm64汇编代码，还不够完善，不能在线上使用

参考文章：

iOS 上修改私有方法的几种方式解析
基于桥的全量方法 Hook 方案
BlockHook with Struct