调用约定

```
@interface Test : NSObject

- (int)addA:(int)a B:(int)b;

@end

@implementation Test

- (int)addA:(int)a B:(int)b {
    return a + b;
}

@end
```

```
+ (void)insertCode {
    SEL sel = @selector(addA:B:);
    Method m = class_getInstanceMethod([Test class], sel);
    IMP imp0 = method_getImplementation(m);
    IMP imp1 = imp_implementationWithBlock(^int(Test *t, int a, int b) {
        // 这是我们要增加的代码
        puts(__func__);
        // 原方法
        int c = ((int (*)(Test*, SEL, int, int))imp0)(t, sel, a, b);
        return c;
    });
}
```

调用方法时，存在 call convention （调用约定），参数的类型、位置必须一一匹配。
如果`(int (*)(Test*, SEL, int, int))`
改为`(Long (*)(Test*, SEL, int, int))`会造成不可预知的行为。

因为存在调用约定，因此我们不能简单使用上面的方式来实现 aspects ，我们的目标转向 NSInvocation

实现原理

1. 为要插入的block块新增方法，方法的实现为该block。原方法名为addA:B，那么新方法名为aspects_block_addA:B。
2. 新增方法，新方法的实现为原方法的实现。原方法名为addA:B，那么新方法名为aspects_new_addA:B:。
3. 将原方法的实现改为_objc_msgForward，直接走消息转发流程。
4. 处理-forwardInvocation:。

代码示例

```objc
@interface NSObject (Aspects)

/**
   在sel方法中插入block代码段，block片段先执行，然后再执行sel
 */
+ (void)beforeHookSel:(SEL)sel block:(id)block;

@end
```

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

@implementation NSObject (Aspects)

- (NSMethodSignature *)methodSignatureForSelector:(SEL)aSelector {
    Method m = class_getInstanceMethod([self class], aSelector);
    if (m == nil) return nil;
    IMP imp = method_getImplementation(m);
    const char *types = method_getTypeEncoding(m);

    return [NSMethodSignature signatureWithObjCTypes:types];
}

/* 这里只处理before。懂了before，也就懂了after、replace */
- (void)forwardInvocation:(NSInvocation *)anInvocation {
    SEL sel0 = anInvocation.selector;

    // 调用block
    SEL sel1 = [[self class] blockMethodSel:sel0];
    anInvocation.selector = sel1;
    [anInvocation invoke];

    // 调用原来方法
    SEL sel2 = [[self class] newMethodSel:sel0];
    anInvocation.selector = sel2;
    [anInvocation invoke];
}

//MARK:-

/**
   使用方法交换，在原SEL的前面加上 "aspects_new_"
 */
+ (void)beforeHookSel:(SEL)sel block:(id)block {
    [self addBlockMethod:sel block:block];
    [self addNewMethod:sel];
    [self replaceOriginalMethod:sel];
}
```

代码示例

```
/**
   新增一个方法，在原SEL的前面加上 "aspects_new_"
 */
+ (void)addNewMethod:(SEL)sel {
    SEL sel1 = [self newMethodSel:sel];

    Method m = class_getInstanceMethod([self class], sel);
    IMP imp = method_getImplementation(m);
    const char *types = method_getTypeEncoding(m);

    class_addMethod([self class], sel1, imp, types);
}

/**
   将原来的方法替换为_objc_msgForward，直接进入消息转发
 */
+ (void)replaceOriginalMethod:(SEL)sel {
    Method m = class_getInstanceMethod([self class], sel);
    IMP imp = _objc_msgForward;
    method_setImplementation(m, imp);
}


//MARK:-        新方法名

+ (SEL)blockMethodSel:(SEL)sel {
    const char *s0 = sel_getName(sel);
    const char *s1 = [NSString stringWithFormat:@"%s%s", "aspects_block_", s0].UTF8String;
    SEL sel1 = sel_registerName(s1);
    return sel1;
}

+ (SEL)newMethodSel:(SEL)sel {
    const char *s0 = sel_getName(sel);
    const char *s1 = [NSString stringWithFormat:@"%s%s", "aspects_new_", s0].UTF8String;
    SEL sel1 = sel_registerName(s1);
    return sel1;
}

@end
```

```
    SEL sel = @selector(addA:B:);
    [Test beforeHookSel:sel block:^int(Test *t, int a, int b) {
        puts(__func__);
        // 这里返回值没有任何意义（但会影响返回结果），上面的puts()才是我们想要的
        return 123;
    }];

    _test = [Test new];
    int c = [_test addA:3 B:4];
```