Aspects框架

https://www.jianshu.com/p/399f4a1212e9

风险很小的热修复方案(JSPatch 可以任意修改, Aspects 范围较小)

使用 Runtime

Runtime

Class 反射创建

```
// 1
NSClassFromString(@"NSObject");

// 2
objc_getClass("NSObject");
```

SEL 反射创建

```
// 1
@selector(init);

// 2
sel_registerName("init");

// 3
NSSelectorFromString(@"init");
```

方法替换

```
static void cc_forwardInvocation(id slf, SEL sel, NSInvocation *invocation)
{
    // do what you want to do
}

class_replaceMethod(klass, @selector(forwardInvocation:), (IMP)cc_forwardInvocation, "v@:@");
```

方法新增

```
Class tClass = NSClassFromString(@"UIViewController");
SEL selector = NSSelectorFromString(@"viewDidLoad");

Method targetMethod = class_getInstanceMethod(tClass, selector);
IMP targetMethodIMP = method_getImplementation(targetMethod);
const char *typeEncoding = method_getTypeEncoding(targetMethod);

SEL aliasSelector = NSSelectorFromString([@"cc" stringByAppendingFormat:@"_%@",
NSStringFromSelector(selector)]);
class_addMethod(klass, aliasSelector, method_getImplementation(targetMethod), typeEncoding);
```

新类创建

```
Class cls = objc_allocateClassPair([NSObject class], “CCObject”, 0);
objc_registerClassPair(cls);
```

消息转发

```
// 1. 正常转发
+ (BOOL)resolveClassMethod:(SEL)sel
+ (BOOL)resolveInstanceMethod:(SEL)sel

– (id)forwardingTargetForSelector:(SEL)aSelector

– (NSMethodSignature *)methodSignatureForSelector:(SEL)aSelector
– (void)forwardInvocation:(NSInvocation *)anInvocation

// 2. 自定义转发
void _objc_msgForward(void /* id receiver, SEL sel, ... */ )
```

Method Invoke 的几种方式

调用函数的几种方式：

- 常规调用
- 反射调用
- objc_msgSend
- C 函数调用
- NSInvocation 调用

```
// 常规调用
People *people = [[People alloc] init];
[people helloWord];

// 反射调用
Class cls = NSClassFromString(@"People");
id obj = [[cls alloc] init];
[obj performSelector:NSSelectorFromString(@"helloWord")];

// objc_msgSend
((void(*)(id, SEL))objc_msgSend)(people, sel_registerName("helloWord"));

// C 函数调用
Method initMethod = class_getInstanceMethod([People class], @selector(helloWord));
IMP imp = method_getImplementation(initMethod);
((void (*) (id, SEL))imp)(people, @selector(helloWord));

// NSInvocation 调用
NSMethodSignature *sig = [[People class] ; //类签名
instanceMethodSignatureForSelector:sel_registerName("helloWord")];
NSInvocation *invocation = [NSInvocation invocationWithMethodSignature:sig];
invocation.target = people;
invocation.selector = sel_registerName("helloWord");
[invocation invoke];
```

Hook 流程

1.检查 selector 是否可以替换，里面涉及一些黑名单等判断
2.获取 AspectsContainer，如果为空则创建并绑定目标类
3.创建 AspectIdentifier，引用自定义实现（block）和 AspectOptions 等信息
4.将目标类 forwardInvocation: 方法替换为自定义方法
5.目标类新增一个带有aspects_前缀的方法，新方法（aliasSelector）实现跟目标方法相同
6.将目标方法实现替换为 _objc_msgForward

```
// 将目标类 **forwardInvocation:** 方法替换为自定义方法
IMP originalImplementation = class_replaceMethod(klass, @selector(forwardInvocation:),
(IMP)__ASPECTS_ARE_BEING_CALLED__, "v@:@");
if (originalImplementation) {
    class_addMethod(klass, NSSelectorFromString(AspectsForwardInvocationSelectorName), originalImplementation,
"v@:@");
}

// 目标类新增一个带有` aspects_`前缀的方法，新方法（aliasSelector）实现跟目标方法相同
Method targetMethod = class_getInstanceMethod(klass, selector);
IMP targetMethodIMP = method_getImplementation(targetMethod);

const char *typeEncoding = method_getTypeEncoding(targetMethod);
SEL aliasSelector = NSSelectorFromString([AspectsMessagePrefix stringByAppendingFormat:@"_%@",
NSStringFromSelector(selector)]);
class_addMethod(klass, aliasSelector, method_getImplementation(targetMethod), typeEncoding);

// 将目标方法实现替换为 `_objc_msgForward`
class_replaceMethod(klass, selector, aspect_getMsgForwardIMP(self, selector), typeEncoding);
```

热修复

https://www.jianshu.com/p/d7b24016854e

修改前原生代码

1.首先假如我们项目中写有如下代码：

```
MightyCrash *mc = [[MightyCrash alloc] init];

[mc divideUsingDenominator: 0];
```

divideUsingDenominator方法内部实现如下:

```
– (float)divideUsingDenominator:(NSInteger)denominator {
    return 1.f / denominator;
}
```

问题所在： 因为调用方法时候我们传入的值为0，所以除以0会出现问题

热修复解决：

didFinishLaunchingWithOptions 方法中

```
– (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    //初始化环境
    [Felix fixIt];
    //后台返回字符串格式
    NSString *fixScriptString = @" \
    fixInstanceMethodReplace('MightyCrash', 'divideUsingDenominator:', function(instance, originInvocation, originArguments){ \
        if (originArguments[0] == 0) { \
            console.log('zero goes here'); \
        } else { \
            runInvocation(originInvocation); \
        } \
    }); \
    \
    ";
    //修复
    [Felix evalString:fixScriptString];

    return YES;
}
```

原理

1. 首先第一步的[Felix fixIt];执行结果，会生成一个全局的JSContext对象来为执行JS方法提供环境

```
+ (JSContext *)context {
    static JSContext *_context;
    static dispatch_once_t onceToken;
    dispatch_once(&onceToken, ^{
        _context = [[JSContext alloc] init];
        [_context setExceptionHandler:^(JSContext *context, JSValue *value) {
            NSLog(@"Oops: %@", value);
        }];
    });
    return _context;
}
```

2. 同时，使用匿名函数的方式包装OC方法：

```
[self context][@"fixInstanceMethodBefore"] = ^(NSString *instanceName, NSString *selectorName,
JSValue *fixImpl) {
    [self _fixWithMethod:NO aspectionOptions:AspectPositionBefore instanceName:instanceName
selectorName:selectorName fixImpl:fixImpl];
};
```

包装的OC方法是什么呢？我们来看实现：

```
+ (void)_fixWithMethod:(BOOL)isClassMethod aspectionOptions:(AspectOptions)option instanceName:(NSString
*)instanceName selectorName:(NSString *)selectorName fixImpl:(JSValue *)fixImpl {
    Class klass = NSClassFromString(instanceName);
    if (isClassMethod) {
        klass = object_getClass(klass);
    }
    SEL sel = NSSelectorFromString(selectorName);
    [klass aspect_hookSelector:sel withOptions:option usingBlock:^(id<AspectInfo> aspectInfo){
        [fixImpl callWithArguments:@[aspectInfo.instance, aspectInfo.originalInvocation, aspectInfo.arguments]];
    } error:nil];
}
```

3. 字符串fixScriptString，该字符串其实是一个JS方法：

- 方法传入的参数1是MightyCrash类，即我们要来hack的目标类
- 方法传入的参数2是divideUsingDenominator，即我们要来hack的目标类的方法
- 方法传入的参数1是一个function，即我们要替换的方法实现

4. 第三步[Felix evalString:fixScriptString]

- 会在全局的JSContent环境中, 执行下面绿色部分fixInstanceMethodReplaceJS方法，从而执行对应OC方法
- OC方法使用Method Swizzling黑魔法完成了目标执行函数的替换，即MightyCrash类中的–(float)divideUsingDenominator:(NSInteger)denominator;方法替换为 [fixImpl callWithArguments:@[aspectInfo.instance, aspectInfo.originalInvocation, aspectInfo.arguments]];

5. 最后,当执行原生方法[mc divideUsingDenominator:0]; 时

- 其实是调用了[fixImpl callWithArguments:@[aspectInfo.instance, aspectInfo.originalInvocation, aspectInfo.arguments]];方法，该方法会执行第二步声明的JS方法的第三个function参数，即执行：

```
function(instance, originInvocation, originArguments){
    if (originArguments[0] == 0) {
        console.log('zero goes here');
    } else {
        runInvocation(originInvocation);
    }
```

LBYFix 使用

https://www.jianshu.com/p/d4574a4268b3

初始化LBYFix

```
[LBYFix fixIt];
```

替换方法实现

```
NSString *jsString = @"fixMethod('LBYFixDemo', 'instanceMightCrash:', 1, false, \
    function(instance, originInvocation, originArguments) { \
        if (originArguments[0] == null) { \
            runErrorBranch('LBYFixDemo', 'instanceMightCrash'); \
        } else { \
            runInvocation(originInvocation); \
        } \
    }); \
    ";
[LBYFix evalString:jsString];
```

在方法前插入代码

```
NSString *jsString = @"fixMethod('LBYFixDemo',  'runBeforeInstanceMethod', 2, false, \
    function(){ \
        runInstanceMethod('LBYFixDemo', 'beforeInstanceMethod:param2:', new Array('LBYFix', 888)); \
    });";
[LBYFix evalString:jsString];
```

在方法后插入代码

```
NSString *jsString = @"fixMethod('LBYFixDemo2', 'runAfterClassMethod', 0, true, \
    function(){ \
        runClassMethod('LBYFixDemo2', 'afterClassMethod:param2:', new Array('LBYFix', 999)); \
    }); \
    ";
[LBYFix evalString:jsString];
```

执行没有参数的方法

```
NSString *jsString = @"runInstanceMethod('LBYFixDemo3', 'instanceMethodHasNoParams')";
[LBYFix evalString:jsString];
```

执行带多个参数的方法

```
NSString *jsString = @"runInstanceMethod('LBYFixDemo3', 'instanceMethodHasMultipleParams:size:rect:', new Array({x:
1.1, y: 2.2},  {width: 3.3, height: 4.4}, {origin: {x: 5.5, y: 6.6}, size: {width: 7.7, height: 8.8}}))\
    ";
[LBYFix evalString:jsString];
```

上面js代码的意思是调用LBYFixDemo3类的instanceMethodHasMultipleParams:size:rect:实例方法，并通过数组传入参数。

原理解析等等