Clang Attributes 黑魔法小记

2016年5月14日

Clang Attributes 是 Clang 提供的一种**源码注解**，方便开发者向编译器表达某种要求，参与控制如 Static Analyzer、Name Mangling、Code Generation 等过程，一般以 `__attribute__(xxx)` 的形式出现在代码中；为方便使用，一些常用属性也被 Cocoa 定义成宏，比如在系统头文件中经常出现的 `NS_CLASS_AVAILABLE_IOS(9_0)` 就是 `__attribute__(availability(...))` 这个属性的简单写法。

常见属性的介绍，可以看 NSHipster 的介绍文章 和的 twitter 的介绍文章。本文还会介绍几个有意思的 **“黑魔法”** Attribute，说不定在某些场景下会起到意想不到的效果哦~

> 以下测试都以 Xcode 7.3 (Clang 3.8) 为准

objc_subclassing_restricted

使用这个属性可以定义一个 `Final Class` ，也就是说，一个不可被继承的类，假设我们有个名叫 `Eunuch（太监）` 的类，但并不希望有人可以继承自它：

```
@interface Eunuch : NSObject
@end
@interface Child : Eunuch // 太监不能够有孩砸
@end
```

只要在 @interface 前面加上 `objc_subclassing_restricted` 这个属性即可：

```
__attribute__((objc_subclassing_restricted))
@interface Eunuch : NSObject
@end
@interface Child : Eunuch // <--- Compile Error
@end
```

objc_requires_super

aka: `NS_REQUIRES_SUPER` ，标志子类继承这个方法时需要调用 `super` ，否则给出编译警告：

```
@interface Father : NSObject
- (void)hailHydra __attribute__((objc_requires_super));
@end
@implementation Father
- (void)hailHydra {
    NSLog(@"hail hydra!");
}
@end
@interface Son : Father
@end
@implementation Son
- (void)hailHydra {
} // <--- Warning missing [super hailHydra]
@end
```

objc_boxable

Objective-C 中的 `@(...)` 语法糖可以将基本数据类型 box 成 `NSNumber` 对象，假如想 box 一个 `struct` 类型或是 `union` 类型成 `NSValue` 对象，可以使用这个属性：

```
typedef struct __attribute__((objc_boxable)) {
    CGFloat x, y, width, height;
} XXRect;
```

这样一来， `XXRect` 就具备被 box 的能力：

```
CGRect rect1 = {1, 2, 3, 4};
NSValue *value1 = @(rect1); // <--- Compile Error
XXRect rect2 = {1, 2, 3, 4};
NSValue *value2 = @(rect2); // √
```

constructor / destructor

顾名思义，构造器和析构器，加上这两个属性的函数会在分别在可执行文件（或 shared library）**load** 和 **unload** 时被调用，可以理解为在 `main()` 函数调用前和 return 后执行：

```
__attribute__((constructor))
static void beforeMain(void) {
    NSLog(@"beforeMain");
}
__attribute__((destructor))
static void afterMain(void) {
    NSLog(@"afterMain");
}
int main(int argc, const char * argv[]) {
    NSLog(@"main");
```

```
    return 0;
}

// Console:
// "beforeMain" -> "main" -> "afterMain"
```

constructor 和 `+load` 都是在 main 函数执行前调用，但 `+load` 比 constructor 更加早一丢丢，因为 dyld（动态链接器，程序的最初起点）在加载 image（可以理解成 Mach-O 文件）时会先通知 `objc runtime` 去加载其中所有的类，每加载一个类时，它的 `+load` 随之调用，全部加载完成后，dyld 才会调用这个 image 中所有的 constructor 方法。

所以 constructor 是一个干坏事的绝佳时机：

1. 所有 Class 都已经加载完成
2. main 函数还未执行
3. 无需像 +load 还得挂载在一个 Class 中

`FDStackView` 的 FDStackViewPatchEntry 方法便是使用的这个时机来实现偷天换日的伎俩。

> PS：若有多个 constructor 且想控制优先级的话，可以写成 `__attribute__((constructor(101)))`，里面的数字越小优先级越高，1 ~ 100 为系统保留。

enable_if

这个属性只能用在 C 函数上，可以用来实现**参数的静态检查**：

```
static void printValidAge(int age)
__attribute__((enable_if(age > 0 && age < 120, "你丫火星人？"))) {
    printf("%d", age);
}
```

它表示调用这个函数时必须满足 `age > 0 && age < 120` 才被允许，于是乎：

```
printValidAge(26); // √
printValidAge(150); // <--- Compile Error
printValidAge(-1); // <--- Compile Error
```

cleanup

声明到一个变量上，当这个变量作用域结束时，调用指定的一个函数，Reactive Cocoa 用这个特性实现了神奇的 `@onExit` ，关于这个 attribute，在之前的文章中有介绍，传送门。

overloadable

用于 C 函数，可以定义若干个函数名相同，但参数不同的方法，调用时编译器会自动根据参数选择函数原型：

C

```
__attribute__((overloadable)) void logAnything(id obj) {
    NSLog(@"%@", obj);
}
__attribute__((overloadable)) void logAnything(int number) {
    NSLog(@"%@", @(number));
}
__attribute__((overloadable)) void logAnything(CGRect rect) {
    NSLog(@"%@", NSStringFromCGRect(rect));
}
// Tests
logAnything(@[@"1", @"2"]);
logAnything(233);
logAnything(CGRectMake(1, 2, 3, 4));
```

objc_runtime_name

用于 `@interface` 或 `@protocol` ，将类或协议的名字在编译时指定成另一个：

```
__attribute__((objc_runtime_name("SarkGay")))
@interface Sark : NSObject
@end

NSLog(@"%@", NSStringFromClass([Sark class])); // "SarkGay"
```

所有直接使用这个类名的地方都会被替换（唯一要注意的是这时用反射就不对了），最简单粗暴的用处就是去做个类名混淆：

```
__attribute__((objc_runtime_name("40ea43d7629d01e4b8d6289a132482d0dd5df4fa")))
@interface SecretClass : NSObject
@end
```

还能用数字开头，怕不怕--，假如写个脚本把每个类前加个随机生成的 `objc_runtime_name` ，岂不是最最精简版的代码混淆就完成了呢...

它是我所了解的唯一一个对 objc 运行时类结构有影响的 attribute，通过编码类名可以在编译时注入一些信息，被带到运行时之后，再反解出来，这就相当于开设了一条秘密通道，打通了写码时和运行时。脑洞一下，假如把这个 attribute 定义成宏，以 `annotation` 的形式完成某些功能，比如：

```
// @singleton 包裹了 __attribute__((objc_runtime_name(...)))
// 将类名改名成 "SINGLETON_Sark_sharedInstance"
@singleton(Sark, sharedInstance)
@interface Sark : NSObject
+ (instancetype)sharedInstance;
@end
```

在运行时用 `__attribute__((constructor))` 获取入口时机，用 runtime 找到这个类，反解出 “sharedInstance” 这个 selector 信息，动态将 `+ alloc` ， `- init` 等方法替换，返回 `+ sharedInstance` 单例。

References

http://llvm.org/releases/3.8.0/tools/clang/docs/AttributeReference.html
http://clang-analyzer.llvm.org/annotations.html

上一篇 下一篇