

Objective-C 中的消息与消息转发

由 ibireme | 2013-11-26 | iOS, 技术

[receiver message];

objective-c的这种有趣的语法被苹果称为“发消息”。与其他面向对象语言(C++/Java)的“方法调用”不同，objc的消息机制是由运行时实现、非常灵活动态。这篇文章简单记录一下objc运行时对于消息发送和转发的实现。

1.编译器的转换

```
[receiver message];
```

这一句的含义是：向receiver发送名为message的消息。

```
clang -rewrite-objc MyClass.m
```

执行上面的命令，将这一句重写为C代码，是这样的：

```
((void (*)(id, SEL))(void *)objc_msgSend)((id)receiver, sel_
registerName("message"));
```


去掉那些强制转换，最终[receiver message]会由编译器转化为以下的纯C调用。

```
objc_msgSend(receiver, @selector(message));
```

所以说，objc发送消息，最终大都会转换为objc_msgSend的方法调用。

苹果在[文档](#)里是这么写的：

```
id objc_msgSend(id self, SEL _cmd, ...)
```

将一个消息发送给一个对象，并且返回一个值。

其中，self是消息的接受者，_cmd是selector，...是可变参数列表。

当向一般对象发送消息时，调用objc_msgSend；当向super发送消息时，调用的是objc_msgSendSuper；如果返回值是一个结构体，则会调用objc_msgSend_stret或objc_msgSendSuper_stret。

2.运行时定义的数据结构

为了了解objc_msgSend方法做了什么，这里需要查看一下objc runtime的[源码](#)。

首先 runtime定义了如下的数据类型：

```
typedef struct objc_class *Class;
typedef struct objc_object *id;
struct objc_object {
    Class isa;
};
struct objc_class {
    Class isa;
}

/// 不透明结构体，selector
typedef struct objc_selector *SEL;

/// 函数指针，用于表示对象方法的实现
typedef id (*IMP)(id, SEL, ...);
```


`id`指代objc中的对象，每个对象的在内存的结构并不是确定的，但其首地址指向的肯定是isa。通过isa指针，运行时就能获取到objc_class。

`objc_class`表示对象的Class，它的结构是确定的，由编译器生成。

`SEL`表示选择器，这是一个不透明结构体。但是实际上，通常可以把它理解为一个字符串。例如
`printf(“%s”,@selector(isEqual:))`会打印出”isEqual:”。运行时维护着一张SEL的表，将相同字符串的方法名映射到唯一一个SEL。通过`sel_registerName(char *name)`方法，可以查找到

这张表中方法名对应的SEL。苹果提供了一个语法糖@selector用来方便地调用该函数。

IMP是一个函数指针。objc中的方法最终会被转换成纯C的函数，IMP就是为了表示这些函数的地址。

了解了这些基础的类型定义后，clang翻译的代码就能看懂了：

```
///objc代码
@implementation NyanCat
+ (void)nyan {
    printf("%p %p",self, _cmd);
}

- (int) setObject:(id)obj forKey:(id)key {
    printf("%p %p %p %p",self, _cmd, obj, key);
    return 0;
}
@end

///c翻译的版本
static void _C_NyanCat_nyan(Class self, SEL _cmd) {
    printf("%p %p",self, _cmd);
}

static int _I_NyanCat_setObject_forKey_(NyanCat * self, SEL
_cmd, id obj, id key) {
    printf("%p %p %p %p",self, _cmd, obj, key);
    return 0;
}
```


这里就可以看出来了，实际上消息发送，最终会转换成调用C函数。obj_msgSend的实际动作就是：找到这个函数指针，然后调用它。

3.obj_msgSend的动作

为了加快速度，苹果对这个方法做了很多优化，这个方法是用汇编实现的。我汇编渣渣。。所以这里稍微整理一下，用伪代码表示：

```
id objc_msgSend(id self, SEL op, ...) {
    if (!self) return nil;
    IMP imp = class_getMethodImplementation(self->isa, SEL op);
    imp(self, op, ...); //调用这个函数，伪代码...
}

//查找IMP
IMP class_getMethodImplementation(Class cls, SEL sel) {
    if (!cls || !sel) return nil;
    IMP imp = lookupImpOrNil(cls, sel);
    if (!imp) return _objc_msgForward; //这个是用于消息转发的
    return imp;
}

IMP lookupImpOrNil(Class cls, SEL sel) {
    if (!cls->initialize()) {
```

```
        _class_initialize(cls);
    }

    Class curClass = cls;
    IMP imp = nil;
    do { //先查缓存,缓存没有时重建,仍旧没有则向父类查询
        if (!curClass) break;
        if (!curClass->cache) fill_cache(cls, curClass);
        imp = cache_getImp(curClass, sel);
        if (imp) break;
    } while (curClass = curClass->superclass);

    return imp;
}
```


objc_msgSend的动作比较清晰：首先在Class中的缓存查找imp（没缓存则初始化缓存），如果没找到，则向父类的Class查找。如果一直查找到根类仍旧没有实现，则用_objc_msgForward函数指针代替imp。最后，执行这个imp。

_objc_msgForward是用于消息转发的。这个函数的实现并没有在objc-runtime的开源代码里面，而是在Foundation框架里面实现的。加上断点启动程序后，会发现__CFInitialize这个方法会调用objc_setForwardHandler函数来注册一个实现。

4.消息转发

上面可以知道，当向一个对象发送一条消息，但它并没有实现的时候，_objc_msgForward会尝试做消息转发。为了展示消息转发的具体动作，这里尝试向一个对象发送一条错误的消息，并查看一下_objc_msgForward是如何进行转发的。

首先开启调试模式、打印出所有运行时发送的消息：

可以在代码里执行下面的方法：

```
(void)instrumentObjcMessageSends(YES);
```

或者暂停程序运行，并在gdb中输入下面的命令：

```
call (void)instrumentObjcMessageSends(YES)
```


之后，运行时发送的所有消息都会打印到/tmp/msgSend-xxxx文件里了。

这里执行以下语句，向一个对象发送一条错误的消息：

```
Test *test = [Test new];  
[test performSelector:@selector(xxx)];
```

打印出来：

```
+ Test NSObject initialize
+ Test NSObject new
+ Test NSObject alloc
+ Test NSObject allocWithZone:
- Test NSObject init
- Test NSObject performSelector:
+ Test NSObject resolveInstanceMethod:
- Test NSObject forwardingTargetForSelector:
```

- Test NSObject methodSignatureForSelector:
- Test NSObject class
- Test NSObject doesNotRecognizeSelector:

结合NSObject文档可以知道，_objc_msgForward消息转发做了如下几件事：

- 1.调用resolveInstanceMethod:方法，允许用户在此时为该Class动态添加实现。如果有实现了，则调用并返回。如果仍没实现，继续下面的动作。

- 2.调用forwardingTargetForSelector:方法，尝试找到一个能响应该消息的对象。如果获取到，则直接转发给它。如果返回了nil，继续下面的动作。

- 3.调用methodSignatureForSelector:方法，尝试获得一个方法签名。如果获取不到，则直接调用doesNotRecognizeSelector抛出异常。

- 4.调用forwardInvocation:方法，将地3步获取到的方法签名包装成Invocation传入，如何处理就在这里面了。

上面这4个方法均是模板方法，开发者可以override，由runtime来调用。最常见的实现消息转发，就是重写方法3和4，吞掉一个消息或者代理给其他对象都是没问题的。

5.其他注意事项

由上面介绍可以知道，一个objc程序启动后，需要进行类的初始化、调用方法时的cache初始化，所以会有一段”热身”的时间。之后，再发送消息的时候就直接走缓存了，所以消息发送的效率非常高，且没有牺牲动态特性。

如果希望避免方法查找带来的那一丁点开销，可以用 `methodForSelector` 手动获得IMP来直接调用。用 `methodForSelector` 获取IMP时，会尝试forward机制，所以在没有对应方法时，返回的是 `_objc_msgForward`，不会返回 `NULL`。

用 `respondsToSelector` 判断对象是否能响应消息时，会避开forward机制，但是该方法会尝试一次 `resolveInstanceMethod`。

PS: stackoverflow上有两个不错的文章：[1](#) [2](#)

3 评论



Mark Chan 🇨🇳 🌐 🍏 在 2016 年 2 月 26 日的下午 4:10

回复

博主你好，请问你”首先开启调试模式、打印出所有运行时发送的消息：”这里是怎么设置才能把运行时发送的所有消息打印到/tmp/msgSend-xxxx文件里？能不能具体告知一下，thanks :???:



徐汉卿 🇨🇳 🌐 🍏 在 2016 年 12 月 16 日的下午 4:56

回复

太饶人了 看这行 * 头晕



youngsoft 🇨🇳 🌐 🍏 在 2017 年 2 月 7 日的下午 2:22

回复

作者的对_objc_msgForward的内部实现的解析有出入。_objc_msgForward是不会调用resolveInstanceMethod方法的,这个方法是在objc_msgSend内部调用。

引用/广播

1. [OC对象模型及运行时\(2/3\)－消息－剑客|关注科技互联网](#) 🇨🇳
[W ?](#) – [...] 1.Objective-C中的消息与消息转发2.Objective-C Runtime Programming Guide3.Understanding the Objective-C Runtime4.理解Objective-C Runtime5.objc_msgSend() Tour Part 1: The Road Map6.《Effective Objective-C 2.0: 编写高质量iOS与OS X代码的52个有效方法》7.深入分析 objc_msgSend8.stackoverflow上的一篇QA9.深入Objective-C

的动态特性 [...]

2. [iOS 常见知识点（一）：Runtime — 项目经验积累与分享](#) 🇨🇳

[W ?](#) – [...] 消息转发: [...]

3. [runtime进行曲，objc_msgSend的前世今生（一） | 微深度](#)

[🇺🇦 W ?](#) – [...] 1、

<http://yulingtianxia.com/blog/2014/11/05/objective-c-runtime/2、>

<http://blog.csdn.net/reysten/article/details/504404503、>

<http://cocoasamurai.blogspot.jp/2010/01/understanding-objective-c-runtime.html4、>

<http://blog.ibireme.com/2013/11/26/objective-c-messaging/> [...]

4. [Aspects AOP 的实现–演道网–演道网](#) 🇨🇳 [W ?](#) – [...]

<http://blog.ibireme.com/2013/...> [...]

5. [Aspects AOP 的实现–IT文库](#) 🇨🇳 [W ?](#) – [...]

<http://blog.ibireme.com/2013/...> [...]

关于

伽蓝之堂——

一只魔法师的工坊

相关链接

Github

Weibo

Twitter

LinkedIn

DeviantART

功能

登录

文章RSS

评论RSS

WordPress.org

Copyright 2018 ibireme