

Objective-C 中的类和对象

由 ibireme | 2013-11-25 | iOS, 技术

Objective-C的runtime是开源的，源码可以在苹果官网下载到：[objc4](#)。

在objc4-532.2以后，苹果把NSObject的实现也挪进来了。想要了解NSObject底层实现终于不用去抠GNUstep了～

好了，下面正文：

1.id和Class的定义

runtime里面，声明了id和Class的类型，简化一下如下：

```
struct objc_class {
    struct objc_class *isa;
};
struct objc_object {
    struct objc_class *isa;
};

typedef struct objc_class *Class; //类 (class object)
typedef struct objc_object *id;  //对象 (instance of class)
```


在objc中，id代表了一个对象。根据上面的声明，凡是首地址是*isa的struct指针，都可以被认为是objc中的对象。运行时可以通过isa指针，查找到该对象是属于什么类(Class)。

2.运行时的实现方式

根据上面的说法，类对象(Class)同样也算是对象，那它的isa又是指向了什么呢？为了了解这些东西是怎么回事，这里写一个简单的类NyanCat，并且用C重写一遍，看看编译器在底层到底是如何实现的。

```
@interface NyanCat : NSObject {
    int age;
    NSString *name;
}
- (void)nyan;
+ (void)nyan;
@end

@implementation NyanCat
- (void)nyan1 {
    printf("instance nyan~");
}
+ (void)nyan2 {
    printf("class nyan~");
}
@end
```


上面是一个简单的类，有两个instance variable，有一个类方法、一个实例方法。

```
clang -rewrite-objc NyanCat.m
```

在终端执行上面这一条语句，让clang将该类重写为cpp代码，我们就能查看到大概底层的实现机制了(实际编译的文件和这个会有些出入，不同目标架构和不同版本clang也会有不同..权且当参考了)。

rewrite后的代码基本是纯C的，稍微整理一下，可以提取出下面这些信息：

```
//Class的实际结构
struct _class_t {
    struct _class_t *isa;           //isa指针
```

```

    struct _class_t *superclass; //父类
    void *cache;
    void *vtable;
    struct _class_ro_t *ro;      //Class包含的信息
};

//Class包含的信息
struct _class_ro_t {
    unsigned int flags;
    unsigned int instanceStart;
    unsigned int instanceSize;
    unsigned int reserved;
    const unsigned char *ivarLayout;
    const char *name;           //类名
    const struct _method_list_t *baseMethods; //方法
    const struct _objc_protocol_list *baseProtocols; //协议
    const struct _ivar_list_t *ivars; //ivar
    const unsigned char *weakIvarLayout;
    const struct _prop_list_t *properties; //属性
};

//NyanCat(meta-class)
struct _class_t OBJC_METAClass_$_NyanCat = {
    .isa      = &OBJC_METAClass_$_NSObject,
    .superclass = &OBJC_METAClass_$_NSObject,
    .cache     = (void *)&objc_empty_cache,
    .vtable    = (void *)&objc_empty_vtable,
    .ro        = &OBJC_METAClass_ro_$_NyanCat, //包含了类方法等
};

//NyanCat(Class)
struct _class_t OBJC_Class_$_NyanCat = {
    .isa      = &OBJC_METAClass_$_NyanCat, //此处isa指向meta-class
    .superclass = &OBJC_Class_$_NSObject,
    .superclass = (void *)&objc_empty_cache,
    .vtable    = (void *)&objc_empty_vtable,
    .ro        = &OBJC_Class_ro_$_NyanCat, //包含了实例方法 ivar信息等
};

typedef struct objc_object NyanCat; //定义NyanCat类型
//更详细的不贴代码了..

```


所有NyanCat的实例的isa都指向了NyanCat(Class)。

NyanCat(Class)是一个全局变量，其中记录了类名、成员变量信息、property信息、protocol信息和实例方法列表等。

NyanCat(Class)的isa指向了全局变量NyanCat(meta-class)，meta-class里只记录了类名、类方法列表等。

画出图来就是这样：

举例来说一下：

```
NyanCat *cat = [[NyanCat alloc] init];  
[cat nyan1];
```


向cat (**instance**) 发送消息nyan1时，运行时会通过isa指针查找到NyanCat(**Class**)，这里保存着本类中定义的实例方法的指针。

```
[NyanCat nyan2];
```

向NyanCat(Class)发送消息nyan2时，运行时会通过isa查找到NyanCat(meta-class)，这里保存着本类中定义的类方法的指针。

运行时如何利用Class和meta-class来实现动态消息的，以后在记吧～

3.类的继承

在_class_t里面，第二个成员是superclass，很明显这个指针指向了它的父类。运行时可以通过isa和superclass获取一个类在继承树上的完整信息。

为了说明方便，这里把上面的例子稍微改一下：NyanCat : Cat : NSObject 这样一个继承树，画出图来就是这样子的：

如上面图中，跟随黑线，可以看到isa的指向。运行时，每个对象的isa都不为空，这样只要是一个id类型的对象，runtime都可以通过访问首地址偏移(isa)来获取该对象的信息了。

上图中跟随绿线，可以看到superclass的指向。当运行时在搜寻方法、ivar信息时，如果没有找到信息，则会沿superclass的线查找上去，最终NSObject(根类)的superclass是nil。

如果自己定义了一个根类(比如NSProxy), 则这个根类会替换图中NSObject的位置。

为了验证上面的说法, 可以敲一下代码看看:

```
#import "NyanCat.h"
#import <objc/runtime.h>
#import <objc/objc.h>

void test() {
    NyanCat *cat = [[NyanCat alloc] init];

    Class cls = object_getClass(cat); //NyanCat(Class)
    class_getName(cls);               //"NyanCat"
    class_isMetaClass(cls);           //NO

    Class meta = object_getClass(cls); //NyanCat(meta-class)
    class_getName(meta);              //"NyanCat"
    class_isMetaClass(cls);           //YES

    Class meta_meta = object_getClass(meta); //NSObject(meta
-class)
    class_getName(meta_meta);         //"NSObject"
    class_isMetaClass(meta_meta);     //YES
}
```


最后吐槽一下：平时开发时，meta-class基本是用不着接触的，superclass指针无法访问，isa指针可能稍后也会隐藏起来(苹果的动作真多)。。所以上面说得这些，了解一下就好～～

9 评论



满月 🇨🇳 🌐 🍏 在 2015 年 11 月 8 日的下午 6:28

[回复](#)

:o :o :o :o 非常实用，感谢分享。



yingkong1987 🇨🇳 🌐 🍏 在 2015 年 11 月 30 日的下午 4:29

[回复](#)

运行时如何利用Chass和meta-class来实现动态消息的，以后在记吧～

“class”有拼写问题



kuengye 🇨🇳 🌐 🍏 在 2016 年 8 月 1 日的上午 11:27

[回复](#)

```
Class meta = object_getClass(cls);  
//NyanCat(meta-class)  
class_getName(meta); //”NyanCat”  
class_isMetaClass(cls); //YES
```

最新测试这个 class_isMetaClass为NO。



李长成 🇨🇳 🌐 🍏 在 2017 年 7 月 28 日的下午 2:45

[回复](#)

作者那里写错了啊很明显..

陈晓明 🇨🇳 🌐 🍏 在 2017 年 10 月 17 日的下午 4:30

[回复](#)

应该是class_isMetaClass(meta);//YES



谢俊逸 🇨🇳 🌐 🍏 在 2017 年 2 月 8 日 的 下午 4:49

回复

Hook 的时候 isa,meta-class 很常用诶



vinnyxiong 🇨🇳 🌐 🍏 在 2017 年 2 月 9 日 的 上午

回复

11:07

```
@interface NyanCat : NSObject {  
    int age;  
    NSString *name;  
}  
- (void)nyan;  
+ (void)nyan;  
@end
```

这里的方法名是不是写错了，应该是：

```
@interface NyanCat : NSObject {  
    int age;  
    NSString *name;  
}  
- (void)nyan1;  
+ (void)nyan2;  
@end
```



teanfoo 🇨🇳 🌐 🍏 在 2017 年 2 月 10 日 的 上午 9:59

回复

Class meta = object_getClass(cls);

```
//NyanCat(meta=class)
class_getName(meta); //"NyanCat"
class_isMetaClass(cls); //YES
```

这里应该是class_isMetaClass(meta); 吧?



liangzai 🇨🇳 🌐 🍏 在 2017 年 9 月 9 日 的 下午 3:07

[回复](#)

问下 runtime.h 和objc-runtime-new.h是什么关系?

```
typedef struct class_t {
    struct class_t *isa;
    struct class_t *superclass;
    Cache cache;
    IMP *vtable;
    .....
```

} 比如后者中定义了一个这样的结构体看着像是类的结构体定义

```
struct objc_class {
    Class isa;
```

```
#if !__OBJC2__
```



```
Class super_class OBJC2_UNAVAILABLE;
const char *name OBJC2_UNAVAILABLE;
long version OBJC2_UNAVAILABLE;
long info OBJC2_UNAVAILABLE;
long instance_size OBJC2_UNAVAILABLE;
struct objc_ivar_list *ivars
```



```
OBJC2_UNAVAILABLE;  
struct objc_method_list **methodLists  
OBJC2_UNAVAILABLE;  
struct objc_cache *cache  
OBJC2_UNAVAILABLE;  
struct objc_protocol_list *protocols  
OBJC2_UNAVAILABLE;  
#endif
```

} 前者中定义了一个这样的结构体 看着也像是类的定义

引用/广播

1. [OC对象模型及运行时\(1/3\)－对象模型](#) — 剑客|关注科技互联网   ? – [...]

1.Objective-C中的类和对象
(instance)2.Objective-C对象模型及应用3.what is meta class
in objective-c4.Objective-C 中的 Meta-class 是什么?

5.Object, Class and Meta Class in Objective-C6.Objective-C Runtime Reference7.KVO/KVC 实现原理进一步分析

8.KVC/KVO原理详解及编程指南 [...]

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](#)