

然后在同一目录下会多出一个 MyClass.cpp 文件，双击打开，可以看到 init 方法已经被编译器转化为下面这样：

```
1 static instancetype _I_MyClass_init(MyClass * self, SEL _cmd) {
2     if (self = ((MyClass (*)(__rw_objc_super *, SEL))(void *)objc_msgSendSuper)((__rw_c
3         ((void (*)(id, SEL))(void *)objc_msgSend)((id)self, sel_registerName("showUserNa
4     }
5     return self;
6 }
```

我们要找的就是它：

```
1 ((void (*)(id, SEL))(void *)objc_msgSend)((id)self, sel_registerName("showUserName"))
```

objc_msgSend 函数被定义在 objc/message.h 目录下，其函数原型是酱紫滴：

```
1 OBJC_EXPORT void objc_msgSend(void /* id self, SEL op, ... */ )
```

该函数有两个参数，一个 id 类型，一个 SEL 类型。

2、SEL

SEL 被定义在 objc/objc.h 目录下：

```
1 typedef struct objc_selector *SEL;
```

其实它就是个映射到方法的C字符串，你可以用 Objective-C 编译器命令 @selector() 或者 Runtime 系统的 sel_registerName 函数来获得一个 SEL 类型的方法选择器。

3、id

与 SEL 一样，id 也被定义在 objc/objc.h 目录下：

```
1 typedef struct objc_object *id;
```

id 是一个结构体指针类型，它可以指向 Objective-C 中的任何对象。objc_object 结构体定义如下：

```
1 struct objc_object { Class isa OBJC_ISA_AVAILABILITY};
```

我们通常所说的对象，就长这个样子，这个结构体只有一个成员变量 isa，对象可以通过 isa 指针找到其所属的类。isa 是一个 Class 类型的成员变量，那么 Class 又是什么呢？

4、Class

Class 也是一个结构体指针类型：

```
1 typedef struct objc_class *Class;
```

objc_class 结构体是酱紫滴：

```
1 struct objc_class {
2     Class isa OBJC_ISA_AVAILABILITY;
3     #if !__OBJC2__
4         Class super_class OBJC2_UNAVAILABLE;
5         const char *name OBJC2_UNAVAILABLE;
6         long version OBJC2_UNAVAILABLE;
7         long info OBJC2_UNAVAILABLE;
8         long instance_size OBJC2_UNAVAILABLE;
9         struct objc_ivar_list *ivars OBJC2_UNAVAILABLE;
10        struct objc_method_list **methodLists OBJC2_UNAVAILABLE;
11        struct objc_cache *cache OBJC2_UNAVAILABLE;
12        struct objc_protocol_list *protocols OBJC2_UNAVAILABLE;
13    #endif
14 } OBJC2_UNAVAILABLE;
```

我们通常说的类就长这样子：

- Class 也有一个 isa 指针，指向其所属的元类（meta）。
- super_class：指向其超类。
- name：是类名。

chennyshan 评论了 程序员应该接外包吗？ ...
请教下，1. 把vc里臃肿的代码挪到vm里了，是否还是臃肿？ 2. 号称可测试性
chennyshan 评论了 iOS MVVM+RAC从框架到实战...
讲的不怎么样
zl520k 评论了 移动端路由层设计
这篇文章还是很有营养的..
ohMyKing 评论了 二级指针与ARC不为人知的特性...
mark
zsny 评论了 iOS动画-从不会到熟练应用...
free();释放指针！
细雨蒙蒙飘 评论了 二级指针与ARC不为人知的特性...

相关帖子
五星宏辉符号单__百度__ 37ㄱ2822ㄱ2 40
五星宏辉彩票机厂家__百度__ 37ㄱ2822ㄱ2 40
怎么样把应用上传到第三方市场
关于岗位职责的一些问题
朱军请看，我是不是被坑了。。。
block里怎么向外部传值呢？
哪一款pdf转换成txt转换器比较好
问一下啊
求助 iOS更新后size中的尺寸不显示
五星宏辉符号单__百度__ 37ㄱ2822ㄱ2 40
五星宏辉彩票机厂家__百度__ 37ㄱ2822ㄱ2 40
怎么样把应用上传到第三方市场
关于岗位职责的一些问题
朱军请看，我是不是被坑了。。。
block里怎么向外部传值呢？
哪一款pdf转换成txt转换器比较好
问一下啊
求助 iOS更新后size中的尺寸不显示

- ·version：是类的版本信息。
- ·info：是类的详情。
- ·instance_size：是该类的实例对象的大小。
- ·ivars：指向该类的成员变量列表。
- ·methodLists：指向该类的实例方法列表，它将方法选择器和方法实现地址联系起来。methodLists 是指向 ·objc_method_list 指针的指针，也就是说可以动态修改 *methodLists 的值来添加成员方法，这也是 Category 实现的原理，同样解释了 Category 不能添加属性的原因。
- ·cache：Runtime 系统会把被调用的方法存到 cache 中（理论上讲一个方法如果被调用，那么它有可能今后还会被调用），下次查找的时候效率更高。
- ·protocols：指向该类的协议列表。

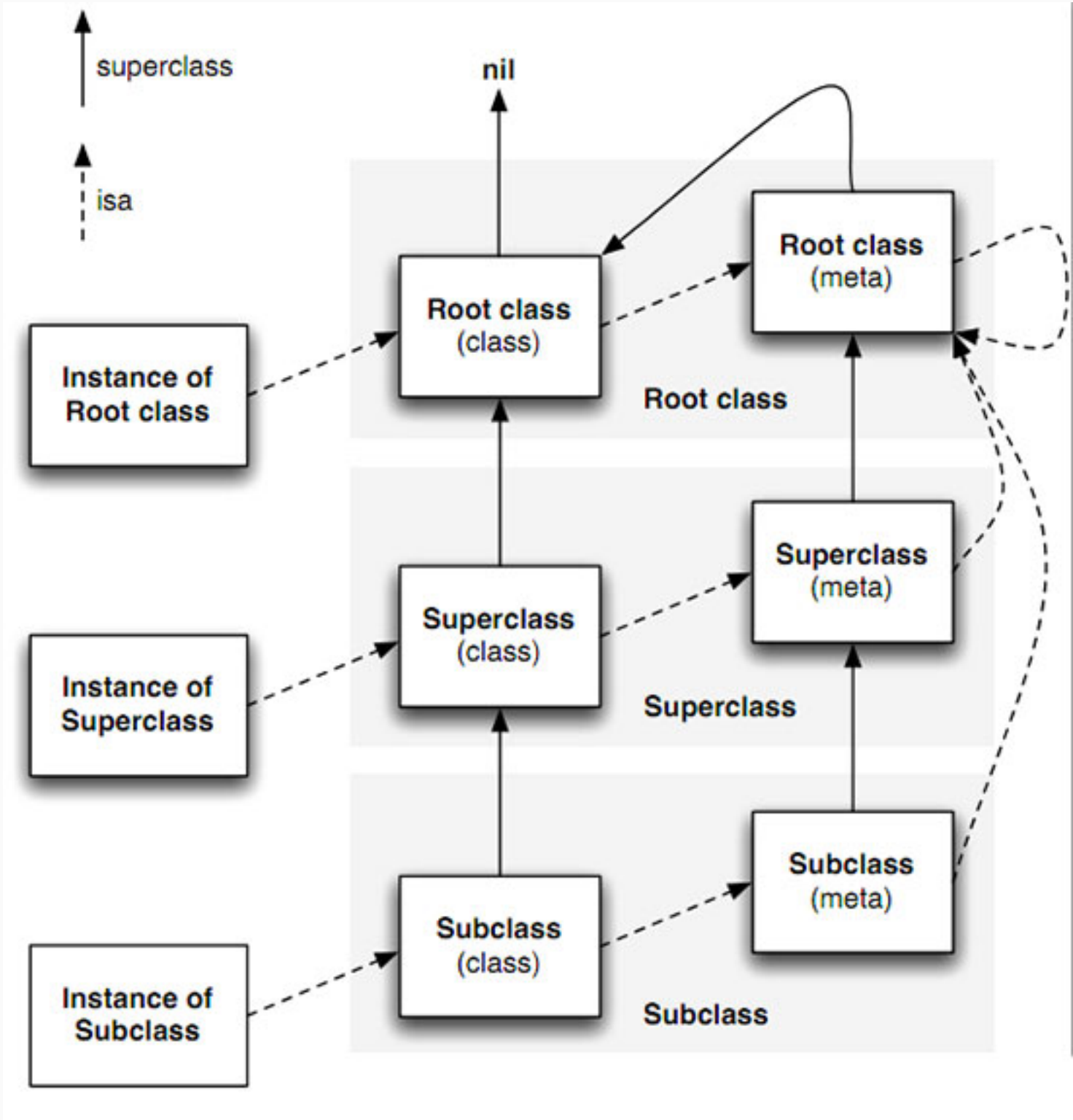
说到这里有点乱了，我们来捋一下，当我们调用一个方法时，其运行过程大致如下：

首先，Runtime 系统会把方法调用转化为消息发送，即 objc_msgSend，并且把方法的调用者，和方法选择器，当做参数传递过去。

此时，方法的调用者会通过 isa 指针来找到其所属的类，然后在 cache 或者 methodLists 中查找该方法，找得到就跳到对应的方法去执行。

如果在类中没有找到该方法，则通过 super_class 往上一级超类查找（如果一直找到 NSObject 都没有找到该方法的话，这种情况，我们放到后面消息转发的时候再说）。

前面我们说 methodLists 指向该类的实例方法列表，实例方法即-方法，那么类方法（+方法）存储在哪儿呢？类方法被存储在元类中，Class 通过 isa 指针即可找到其所属的元类。



上图实线是 superclass 指针，虚线是 isa 指针。根元类的超类是 NSObject，而 isa 指向了自己。NSObject 的超类为 nil，也就是它没有超类。

5、使用objc_msgSend



CocoaChina

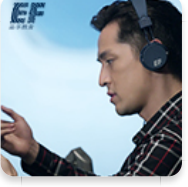
+ 加关注

【iOS分析崩溃日志】<http://t.cn/RlSt3Qe> iOS分析定位崩溃问题有很多种方式，但是发布到AppStore的应用如果崩溃了，我们该怎么办呢？通常我们都会系统中接入统计系统，在系统崩溃的时候记录下崩溃日志，下次启动时将日志发送到服务端，比较好的第三方有umeng之类的。今天我们来讲一下通过崩溃日志来



今天 14:19 转发(18) | 评论(1)

【UITableView如何开启极简模式】<http://t.cn/RJBENQi> ①UITableView中如何适应需求多变（新增删除、经常



想出国，英语很烂怎么办？

[武汉专享]免费订阅每日英语，像胡歌一样，每天5分钟，自信开口说！

前面我们使用 clang 重写命令，看到 Runtime 是如何将方法调用转化为消息发送的。我们也可以依样画葫芦，来学习使用一下 objc_msgSend。新建一个类 TestClass，添加如下方法：

```
1  -(void)showAge{
2      NSLog(@"24");
3  }
4  -(void)showName:(NSString *)aName{
5      NSLog(@"name is %@",aName);
6  }
7  -(void)showSizeWithWidth:(float)aWidth andHeight:(float)aHeight{
8      NSLog(@"size is %.2f * %.2f",aWidth, aHeight);
9  }
10 -(float)getHeight{
11     return 187.5f;
12 }
13 -(NSString *)getInfo{
14     return @"Hi, my name is Dave Ping, I'm twenty-four years old in the year, I like ap
15 }
```

我们可以像下面这样，使用 objc_msgSend 依次调用这些方法：

```
1  TestClass *object = [[TestClass alloc] init];
2  ((void (*)(id, SEL)) objc_msgSend)(object, sel_registerName("showAge"));
3  ((void (*)(id, SEL, NSString *)) objc_msgSend)(object, sel_registerName("showName:"), @
4  ((void (*)(id, SEL, float, float)) objc_msgSend)(object, sel_registerName("showSizeWith
5  float f = ((float (*)(id, SEL)) objc_msgSend_fpret)(object, sel_registerName("getHeight
6  NSLog(@"height is %.2f",f);
7  NSString *info = ((NSString* (*)(id, SEL)) objc_msgSend)(object, sel_registerName("getI
8  NSLog(@"%@",info);
```

也许你已经注意到，objc_msgSend 在使用时都被强制转换了一下，这是因为 objc_msgSend 函数可以hold住各种不同的返回值以及多个参数，但默认情况下是没有参数和返回值的。如果我们把调用 showAge 方法改成这样：

```
1  objc_msgSend(object, sel_registerName("showAge"));
```

Xcode 就会报错：

```
1  Too many arguments to function call, expected 0, have 2.
```

完整的 objc_msgSend 使用代码在[这里](#)。

6、objc_msgSendSuper

编译器会根据情况在 objc_msgSend，objc_msgSend_stret，objc_msgSendSuper，objc_msgSendSuper_stret 或 objc_msgSend_fpret 五个方法中选择一个来调用。如果消息是传递给超类，那么会调用 objc_msgSendSuper 方法，如果消息返回值是数据结构，就会调用 objc_msgSendSuper_stret 方法，如果返回值是浮点数，则调用 objc_msgSend_fpret 方法。

这里我们重点说一下 objc_msgSendSuper，objc_msgSendSuper 函数原型如下：

```
1  OBJC_EXPORT void objc_msgSendSuper(void /* struct objc_super *super, SEL op, ... */) ;
```

当我们调用 [super selector] 时，Runtime 会调用 objc_msgSendSuper 方法，objc_msgSendSuper 方法有两个参数，super 和 op，Runtime 会把 selector 方法选择器赋值给 op。而 super 是一个 objc_super 结构体指针，objc_super 结构体定义如下：

```
1  struct objc_super {
2      /// Specifies an instance of a class.
3      __unsafe_unretained id receiver;
4      /// Specifies the particular superclass of the instance to message.
5  #if !defined(__cplusplus) && !__OBJC2__
6      /* For compatibility with old objc-runtime.h header */
7      __unsafe_unretained Class class;
8  #else
9      __unsafe_unretained Class super_class;
10 #endif
11     /* super_class is the first class to search */
12 };
```

Runtime 会创建一个 objc_spuer 结构体变量，将其地址作为参数（super）传递给 objc_msgSendSuper，并且将 self 赋值给 receiver：super—>receiver=self。

举个栗子，问下面的代码输出什么：

```
1 | @implementation Son : Father
2 | - (id)init
3 | {
4 |     self = [super init];
5 |     if (self)
6 |     {
7 |         NSLog(@"%@", NSStringFromClass([self class]));
8 |         NSLog(@"%@", NSStringFromClass([super class]));
9 |     }
10 |    return self;
11 | }
12 | @end
```

答案是全部输出 Son。

使用 clang 重写命令，发现上述代码被转化为：

```
1 | NSLog((NSString *)&__NSConstantStringImpl__var_folders_gm_0jk35cwn1d3326x0061qym280000gr
2 | NSLog((NSString *)&__NSConstantStringImpl__var_folders_gm_0jk35cwn1d3326x0061qym280000gr
```

当调用 [super class] 时，会转换成 objc_msgSendSuper 函数：

- 第一步先构造 objc_super 结构体，结构体第一个成员就是 self。第二个成员是 (id)class_getSuperclass(objc_getClass(“Son”)).
- 第二步是去 Father 这个类里去找 - (Class)class，没有，然后去 NSObject 类去找，找到了。最后内部是使用 objc_msgSend(objc_super->receiver, @selector(class)) 去调用，此时已经和 [self class] 调用相同了，所以两个输出结果都是 Son。

7、对象关联

对象关联允许开发者对已经存在的类在 Category 中添加自定义的属性：

```
1 | OBJC_EXPORT void objc_setAssociatedObject(id object, const void *key, id value, objc_Ass
```

·object 是源对象

·value 是被关联的对象

·key 是关联的键，objc_getAssociatedObject 方法通过不同的 key 即可取出对应的被关联对象

·policy 是一个枚举值，表示关联对象的行为，从命名就能看出各个枚举值的含义：

```
1 | typedef OBJC_ENUM(uintptr_t, objc_AssociationPolicy) {
2 |     OBJC_ASSOCIATION_ASSIGN = 0,           /**< Specifies a weak reference to the assoc
3 |     OBJC_ASSOCIATION_RETAIN_NONATOMIC = 1, /**< Specifies a strong reference to the ass
4 |         * The association is not made atomically.
5 |     OBJC_ASSOCIATION_COPY_NONATOMIC = 3,    /**< Specifies that the associated object is
6 |         * The association is not made atomically.
7 |     OBJC_ASSOCIATION_RETAIN = 01401,        /**< Specifies a strong reference to the ass
8 |         * The association is made atomically. */
9 |     OBJC_ASSOCIATION_COPY = 01403           /**< Specifies that the associated object is
10 |         * The association is made atomically. */
11 | };
```

要取出被关联的对象使用 objc_getAssociatedObject 方法即可，要删除一个被关联的对象，使用

objc_setAssociatedObject 方法将对应的 key 设置成 nil 即可：

```
1 | objc_setAssociatedObject(self, associatedKey, nil, OBJC_ASSOCIATION_COPY_NONATOMIC);
```

objc_removeAssociatedObjects 方法将会移除源对象中所有的关联对象。

举个栗子，假如我们要给 UIButton 添加一个监听单击事件的 block 属性，新建 UIButton 的 Category，其.m文件如下：

```
1 | #import "UIButton+ClickBlock.h"
2 | #import static const void *associatedKey = "associatedKey";
```



```

3  @implementation UIButton (ClickBlock)
4  //Category中的属性，只会生成setter和getter方法，不会生成成员变量
5  -(void)setClick:(clickBlock)click{
6      objc_setAssociatedObject(self, associatedKey, click, OBJC_ASSOCIATION_COPY_NONATOMIC);
7      [self addTarget:self action:@selector(buttonClick) forControlEvents:UIControlEventTouchUpInside];
8      if (click) {
9          [self addTarget:self action:@selector(buttonClick) forControlEvents:UIControlEventTouchUpInside];
10     }
11 }
12 -(clickBlock)click{
13     return objc_getAssociatedObject(self, associatedKey);
14 }
15 -(void)buttonClick{
16     if (self.click) {
17         self.click();
18     }
19 }
20 @end

```

然后在代码中，就可以使用 UIButton 的属性来监听单击事件了：

```

1  UIButton *button = [UIButton buttonWithType:UIButtonTypeCustom];
2  button.frame = self.view.bounds;
3  [self.view addSubview:button];
4  button.click = ^{
5      NSLog(@"buttonClicked");
6  };

```

完整的对象关联代码点[这里](#)。

8、自动归档

博主在学习 Runtime 之前，归档的时候是酱紫写的：

```

1  - (void)encodeWithCoder:(NSCoder *)aCoder{
2      [aCoder encodeObject:self.name forKey:@"name"];
3      [aCoder encodeObject:self.ID forKey:@"ID"];
4  }
5  - (id)initWithCoder:(NSCoder *)aDecoder{
6      if (self = [super init]) {
7          self.ID = [aDecoder decodeObjectForKey:@"ID"];
8          self.name = [aDecoder decodeObjectForKey:@"name"];
9      }
10     return self;
11 }

```

那么问题来了，如果当前 Model 有100个属性的话，就需要写100行这种代码：

```

1  [aCoder encodeObject:self.name forKey:@"name"];

```

想想都头疼，通过 Runtime 我们就可以轻松解决这个问题：

- 1.使用 class_copyIvarList 方法获取当前 Model 的所有成员变量。
- 2.使用 ivar_getName 方法获取成员变量的名称。
- 3.通过 KVC 来读取 Model 的属性值（encodeWithCoder:），以及给 Model 的属性赋值（initWithCoder:）。

举个栗子，新建一个 Model 类，其.m文件如下：

```

1  #import "TestModel.h"
2  #import #implementation TestModel
3  - (void)encodeWithCoder:(NSCoder *)aCoder{
4      unsigned int outCount = 0;
5      Ivar *vars = class_copyIvarList([self class], &outCount);
6      for (int i = 0; i < outCount; i++) {
7          Ivar var = vars[i];
8          const char *name = ivar_getName(var);
9          NSString *key = [NSString stringWithUTF8String:name];
10         // 注意kvc的特性是，如果能找到key这个属性的setter方法，则调用setter方法
11         // 如果找不到setter方法，则查找成员变量key或者成员变量_key，并且为其赋值
12         // 所以这里不需要再另外处理成员变量名称的“_”前缀
13         id value = [self valueForKey:key];
14         [aCoder encodeObject:value forKey:key];
15     }
16 }
17 - (nullable instancetype)initWithCoder:(NSCoder *)aDecoder{
18     if (self = [super init]) {
19         unsigned int outCount = 0;
20         Ivar *vars = class_copyIvarList([self class], &outCount);

```

```

21         for (int i = 0; i < outCount; i++) {
22             Ivar var = vars[i];
23             const char *name = ivar_getName(var);
24             NSString *key = [NSString stringWithUTF8String:name];
25             id value = [aDecoder decodeObjectForKey:key];
26             [self setValue:value forKey:key];
27         }
28     }
29     return self;
30 }
31 @end

```

完整的自动归档代码在[这里](#)。

9、字典与模型互转

最开始博主是这样用字典给 Model 赋值的：

```

1 -(instancetype)initWithDictionary:(NSDictionary *)dict{
2     if (self = [super init]) {
3         self.age = dict[@"age"];
4         self.name = dict[@"name"];
5     }
6     return self;
7 }

```

可想而知，遇到的问题跟归档时候一样（后来使用MJExtension），这里我们稍微来学习一下其中原理，字典转模型的时候：

- 1.根据字典的 key 生成 setter 方法
- 2.使用 objc_msgSend 调用 setter 方法为 Model 的属性赋值（或者 KVC）

模型转字典的时候：

- 1.调用 class_copyPropertyList 方法获取当前 Model 的所有属性
- 2.调用 property_getName 获取属性名称
- 3.根据属性名称生成 getter 方法
- 4.使用 objc_msgSend 调用 getter 方法获取属性值（或者 KVC）

代码如下：

```

1  #import "NSObject+KeyValues.h"
2  #import #import @implementation NSObject (KeyValues)
3  //字典转模型
4  +(id)objectWithKeyValues:(NSDictionary *)aDictionary{
5      id objc = [[self alloc] init];
6      for (NSString *key in aDictionary.allKeys) {
7          id value = aDictionary[key];
8          /*判断当前属性是不是Model*/
9          objc_property_t property = class_getProperty(self, key.UTF8String);
10         unsigned int outCount = 0;
11         objc_property_attribute_t *attributeList = property_copyAttributeList(property,
12         objc_property_attribute_t attribute = attributeList[0];
13         NSString *typeString = [NSString stringWithUTF8String:attribute.value];
14         if ([typeString isEqualToString:@"@"@"TestModel\"]) {
15             value = [self objectWithKeyValues:value];
16         }
17         /*******/
18         //生成setter方法，并用objc_msgSend调用
19         NSString *methodName = [NSString stringWithFormat:@"set%@%@",[key substringToI
20         SEL setter = sel_registerName(methodName.UTF8String);
21         if ([objc respondsToSelector:setter]) {
22             ((void (*)(id,SEL,id)) objc_msgSend) (objc,setter,value);
23         }
24     }
25     return objc;
26 }
27 //模型转字典
28 -(NSDictionary *)keyValuesWithObject{
29     unsigned int outCount = 0;
30     objc_property_t *propertyList = class_copyPropertyList([self class], &outCount);
31     NSMutableDictionary *dict = [NSMutableDictionary dictionary];
32     for (int i = 0; i < outCount; i++) {
33         objc_property_t property = propertyList[i];

```

```
34 //生成getter方法, 并用objc_msgSend调用
35 const char *propertyName = property_getName(property);
36 SEL getter = sel_registerName(propertyName);
37 if ([self respondsToSelector:getter]) {
38     id value = ((id (*)(id,SEL)) objc_msgSend) (self,getter);
39     /*判断当前属性是不是Model*/
40     if ([value isKindOfClass:[self class]] && value) {
41         value = [value keyValuesWithObject];
42     }
43     /*******/
44     if (value) {
45         NSString *key = [NSString stringWithUTF8String:propertyName];
46         [dict setObject:value forKey:key];
47     }
48 }
49 }
50 return dict;
51 }
52 @end
```

完整代码在[这里](#)。

10、动态方法解析

前面我们留下了一点东西没说，那就是如果某个对象调用了不存在的方法时会怎么样，一般情况下程序会crash，错误信息类似下面这样：

unrecognized selector sent to instance 0x7fd0a141afd0

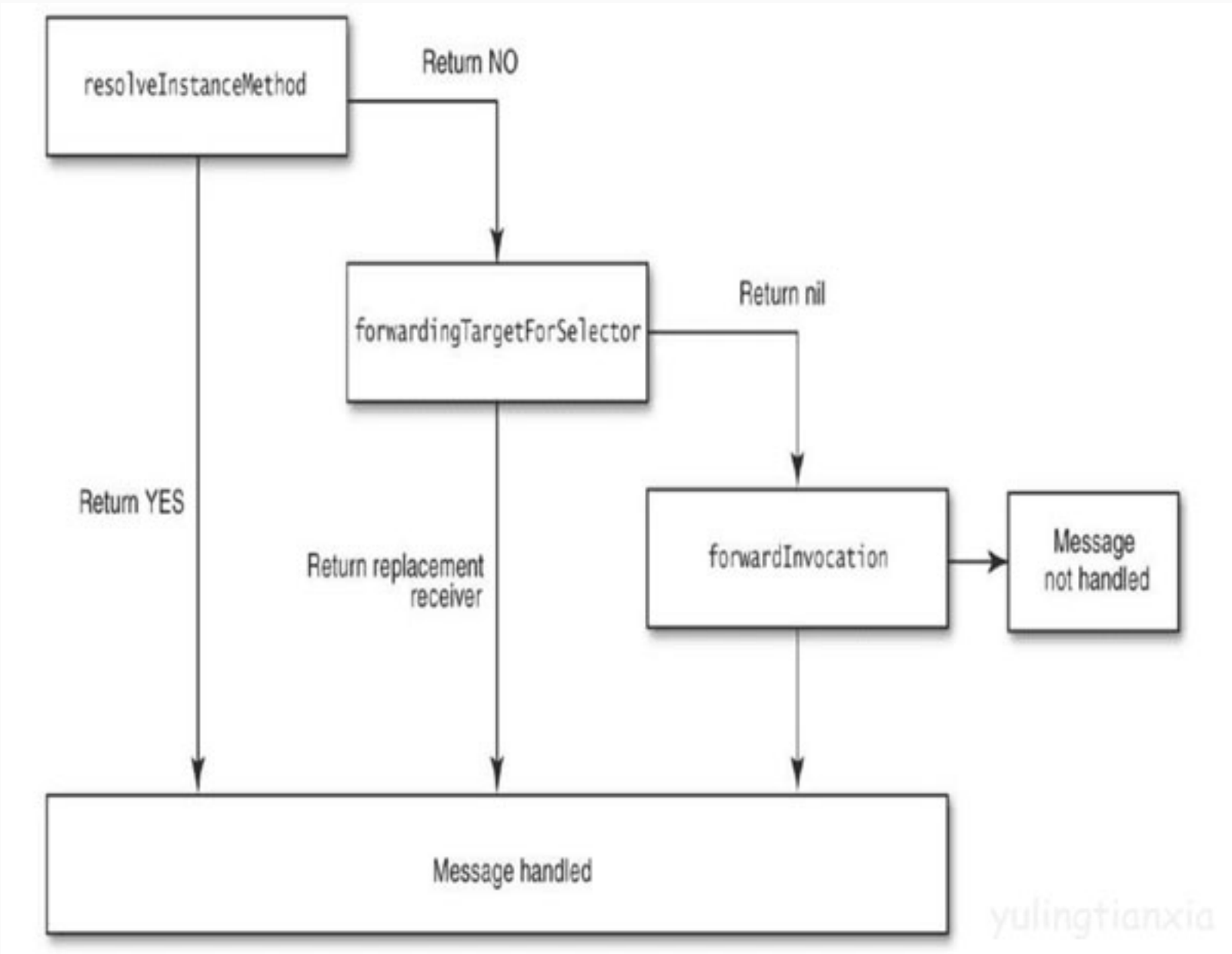
但是在程序crash之前，Runtime 会给我们动态方法解析的机会，消息发送的步骤大致如下：

- 1.检测这个 selector 是不是要忽略的。比如 Mac OS X 开发，有了垃圾回收就不理会 retain，release 这些函数了
- 2.检测这个 target 是不是 nil 对象。ObjC 的特性是允许对一个 nil 对象执行任何一个方法不会 Crash，因为会被忽略掉
- 3.如果上面两个都过了，那就开始查找这个类的 IMP，先从 cache 里面找，完了找得到就跳到对应的函数去执行

如果 cache 找不到就找一下方法分发表

- 4.如果分发表找不到就到超类的分发表去找，一直找，直到找到NSObject类为止

如果还找不到就要开始进入消息转发了，消息转发的大致过程如图：



1.进入 resolveInstanceMethod: 方法， 指定是否动态添加方法。若返回NO， 则进入下一步， 若返回YES， 则通过 class_addMethod 函数动态地添加方法， 消息得到处理， 此流程完毕。

2.resolveInstanceMethod: 方法返回 NO 时， 就会进入 forwardingTargetForSelector: 方法， 这是 Runtime 给我们的第二次机会， 用于指定哪个对象响应这个 selector。返回nil， 进入下一步， 返回某个对象， 则会调用该对象的方法。

3.若 forwardingTargetForSelector: 返回的是nil， 则我们首先要通过 methodSignatureForSelector: 来指定方法签名， 返回nil， 表示不处理， 若返回方法签名， 则会进入下一步。

4.当第 methodSignatureForSelector: 方法返回方法签名后， 就会调用 forwardInvocation: 方法， 我们可以通过 anInvocation 对象做很多处理， 比如修改实现方法， 修改响应对象等。

如果到最后， 消息还是没有得到响应， 程序就会crash， 详细代码在[这里](#)。

参考文章：

Objective-C Runtime

标哥的技术博客 Runtime系列文章

刨根问底Objective – C Runtime（1） – Self & Super



微信扫一扫

订阅每日移动开发及APP推广热点资讯
公众号：CocoaChina

我要投稿

收藏文章

分享到：



上一篇： 实战分享：iOS高仿下厨房（Objective-C版）

下一篇：iOS 小游戏——贪食蛇

相关资讯

- iOS开发之Runtime常用示例总结
- Runtime在实际开发中的应用
- 一道题理清Objective-C中的load和initialize
- 征服恐惧！用 Vim 写 iOS App
- OC快速开发工具集-TFEasyCoder
- Objective-C库文件使用小记
- 从Swift看Objective-C的数组使用
- Swift 3.0 令人兴奋， 但Objective-C也有小改进--
- iOS开发实用技巧—Objective-C中的各种遍历（迭代）方
- Objective-C 深入理解 +load 和 +initialize


首月免费

你上课 我买单

领取免费课程

广告 X

我来说两句



你怎么看？快来评论一下吧！

您还没有登录！请 [登录](#) 或 [注册](#)

发表评论

所有评论（29）

	清蒸鱼 哎呀 看得我脑阔疼	2017-01-31 08:34:01	<div> 0</div> <div> 0</div> <div>回复</div>
	liouly 归档代码缺少了对属性类型的判断啊po主。。。	2016-07-10 15:12:53	<div> 0</div> <div> 0</div> <div>回复</div>
	hmf190195390 mark	2016-06-30 11:05:41	<div> 1</div> <div> 0</div> <div>回复</div>
	冷一萧6 mark	2016-06-20 10:37:32	<div> 0</div> <div> 0</div> <div>回复</div>
	ssxIOS mark	2016-06-12 19:15:18	<div> 0</div> <div> 0</div> <div>回复</div>
	Platycodon_ 从我的项目经验看，实际开发runtime用的并不多，大多用在比较底层的框架上，或者个别用来巧妙解决一些问题。 不过没事翻翻runtime的头文件会有惊喜，突然就会出现灵感。	2016-06-01 10:39:43	<div> 2</div> <div> 0</div> <div>回复</div>
	jiodg45 归档接档时候属性type不一致会崩么？	2016-05-30 17:18:32	<div> 0</div> <div> 0</div> <div>回复</div>
	HWdan 用了class_copyPropertyList，而不用free释放，这样真的没有问题吗	2016-05-27 17:52:38	<div> 4</div> <div> 0</div> <div>回复</div>
	菜刀来了 666666	2016-05-27 17:17:51	<div> 0</div> <div> 0</div> <div>回复</div>
	榕树头 给满分	2016-05-27 16:19:37	<div> 0</div> <div> 0</div> <div>回复</div>

	sneuzb	2016-05-27 14:47:55
	mark	<div><div><div></div></div>1</div> <div><div></div></div> 0

回复

	<div>hmf190195390</div> <div>mark</div>	2016-06-30 11:05:41	<div> 1</div> <div> 0</div> <div>回复</div>
	<div>冷一萧6</div> <div>mark</div>	2016-06-20 10:37:32	<div> 0</div> <div> 0</div> <div>回复</div>
	<div>ssxIOS</div> <div>mark</div>	2016-06-12 19:15:18	<div> 0</div> <div> 0</div> <div>回复</div>
	<div>Platycodon_</div> <div>从我的项目经验看，实际开发runtime用的并不多，大多用在比较底层的框架上，或者个别用来巧妙解决一些问题。 不过没事翻翻runtime的头文件会有惊喜，突然就会出现灵感。</div>	2016-06-01 10:39:43	<div> 2</div> <div> 0</div> <div>回复</div>
	<div>jiodg45</div> <div>归档接档时候属性type不一致会崩么？</div>	2016-05-30 17:18:32	<div> 0</div> <div> 0</div> <div>回复</div>
	<div>HWdan</div> <div>用了class_copyPropertyList，而不用free释放，这样真的没有问题吗</div>	2016-05-27 17:52:38	<div> 4</div> <div> 0</div> <div>回复</div>
	<div>菜刀来了</div> <div>666666</div>	2016-05-27 17:17:51	<div> 0</div> <div> 0</div> <div>回复</div>
	<div>榕树头</div> <div>给满分</div>	2016-05-27 16:19:37	<div> 0</div> <div> 0</div> <div>回复</div>
	<div>scneuzb</div> <div>mark</div>	2016-05-27 14:47:55	<div> 1</div> <div> 0</div> <div>回复</div>
	<div>kengsir</div> <div>runtime 详解：http://www.code4app.com/forum.php?mod=viewthread&tid=8241&highlight=runtime runtime 视频讲解：http://www.code4app.com/forum.php?mod=viewthread&tid=8304&highlight=runtime</div>	2016-05-27 10:59:34	<div> 2</div> <div> 0</div> <div>回复</div>
	<div>tt695544</div> <div>mark</div>	2016-05-26 16:56:00	<div> 0</div> <div> 0</div> <div>回复</div>
	<div>chinawanggebi</div> <div>以后面试装逼用</div>	2016-05-26 13:52:23	<div> 0</div> <div> 0</div> <div>回复</div>

- 

lix

我就是想知道，我知道了本文的知识后，有什么实际用处？在什么时候什么地方我能用得着？因为我开发了一年了，都没写过本文的代码



3



6

回复



_军军

我盡然看完了，雖然沒有看懂



3



1

回复



牛奶你个面包

留着以后装逼用



1



1

回复



panzhengquan

mark



1



1

回复



sany322

太多了 记不住 装不了



1



1

回复



rsxs1103417

mark



1



1

回复

更多评论

关于我们

商务合作

联系我们

合作伙伴

北京触控科技有限公司版权所有

©2016 Chukong Technologies, Inc.

京ICP备 11006519号

京ICP证 100954号

京公网安备11010502020289



京网文[2012]0426-138号