iOS

课题:runtime面试资料

授课老师：Cooci

http://www.tzios.com/

iOS学院 公开课

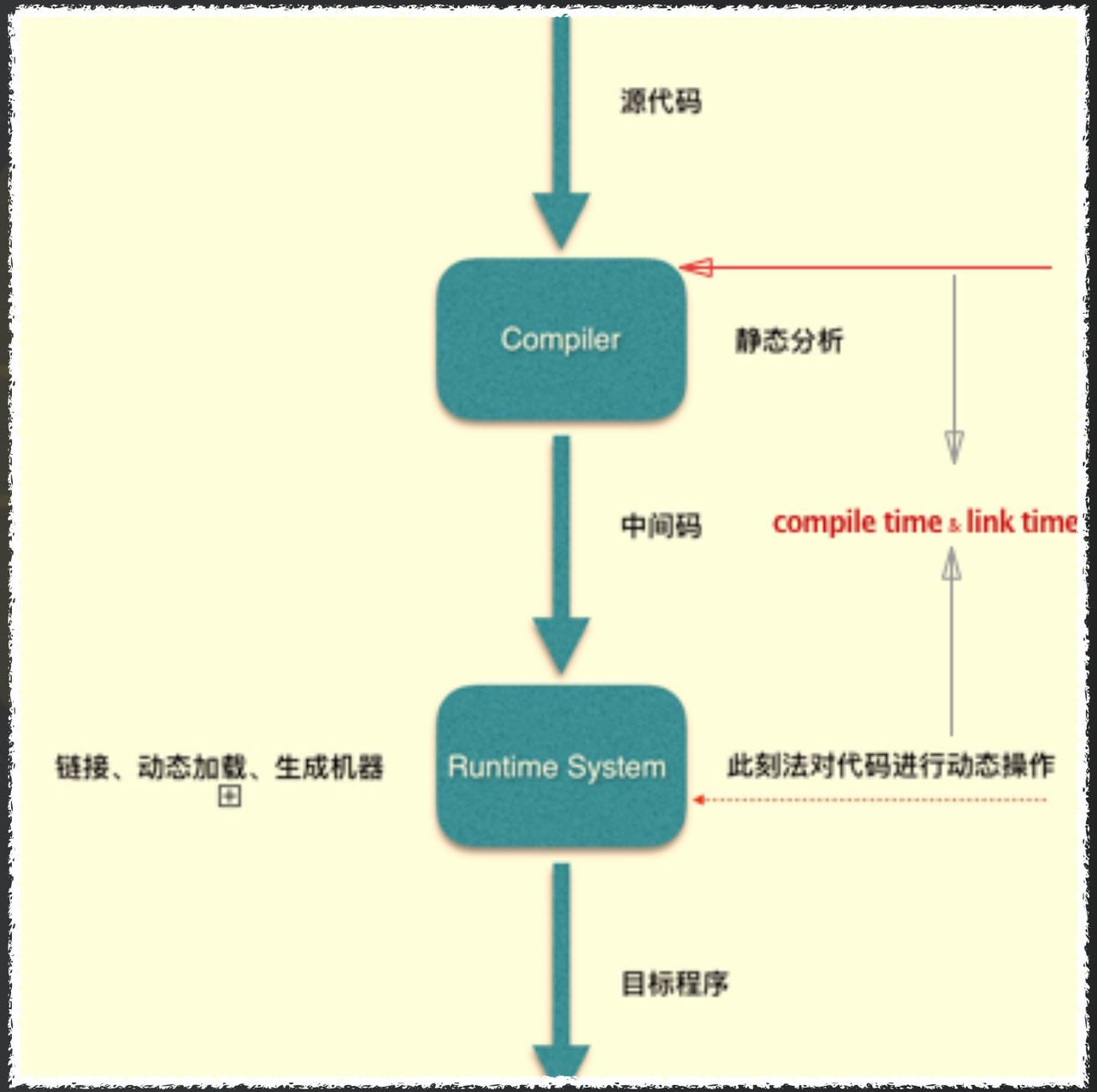
深刻理解Runtime机制

- What ：什么是Runtime?
- why：为什么要学习Runtime机制呀?
- How ：知道Runtime底层工作原理吗?
- where：常见的Runtime应用场景在哪里呢?

什么是Runtime?

Runtime是什么？见名知意，其概念无非就是"因为 Objective-C 是一门动态语言，所以它需要一个运行时系统……这就是 Runtime 系统"云云。对博主这种菜鸟而言，Runtime 在实际开发中，其实就是一组C语言的函数。胡适说："多研究些问题，少谈些主义"，云山雾罩的概念听多了总是容易头晕，接下来我们直接从代码入手学习Runtime。

The Objective-C language defers as many decisions as it can from compile time and link time to runtime. Whenever possible, it does things dynamically. This means that the language requires not just a compiler, but also a runtime system to execute the compiled code. The runtime system acts as a kind of operating system for the Objective-C language; it's what makes the language work.



clang is a C, C++, and Objective-C compiler which encompasses preprocessing, parsing, optimization, code generation, assembly, and linking.

```objc
int main(int argc, const char * argv[]) {
    @autoreleasepool {
        // insert code here...
        [[NSObject alloc] init];
    }
    return 0;
}
```

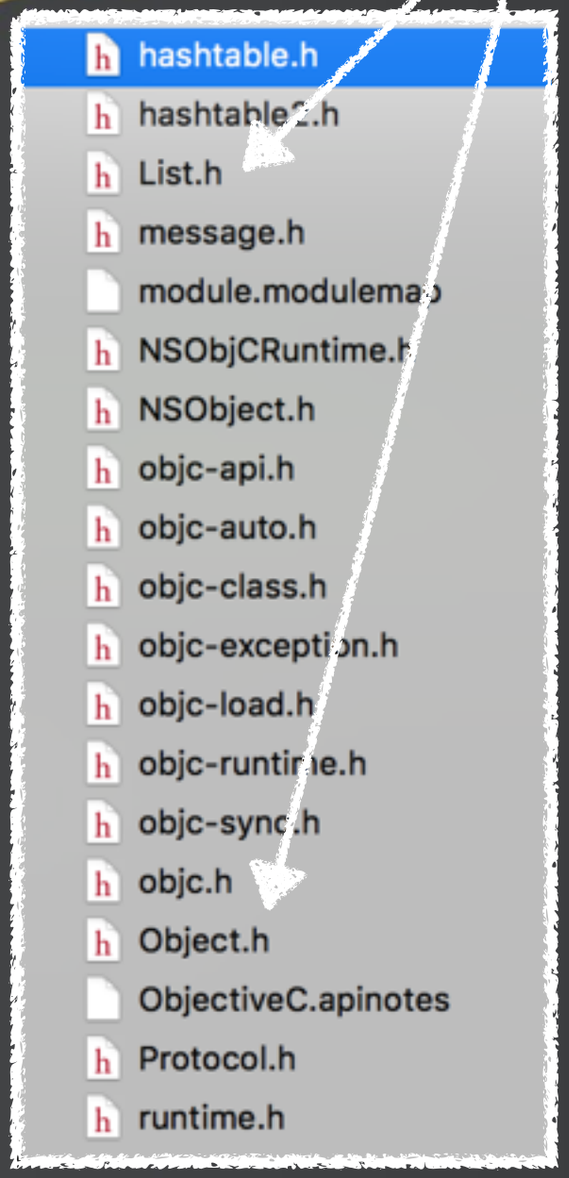
编译器：clang -rewrite-objc

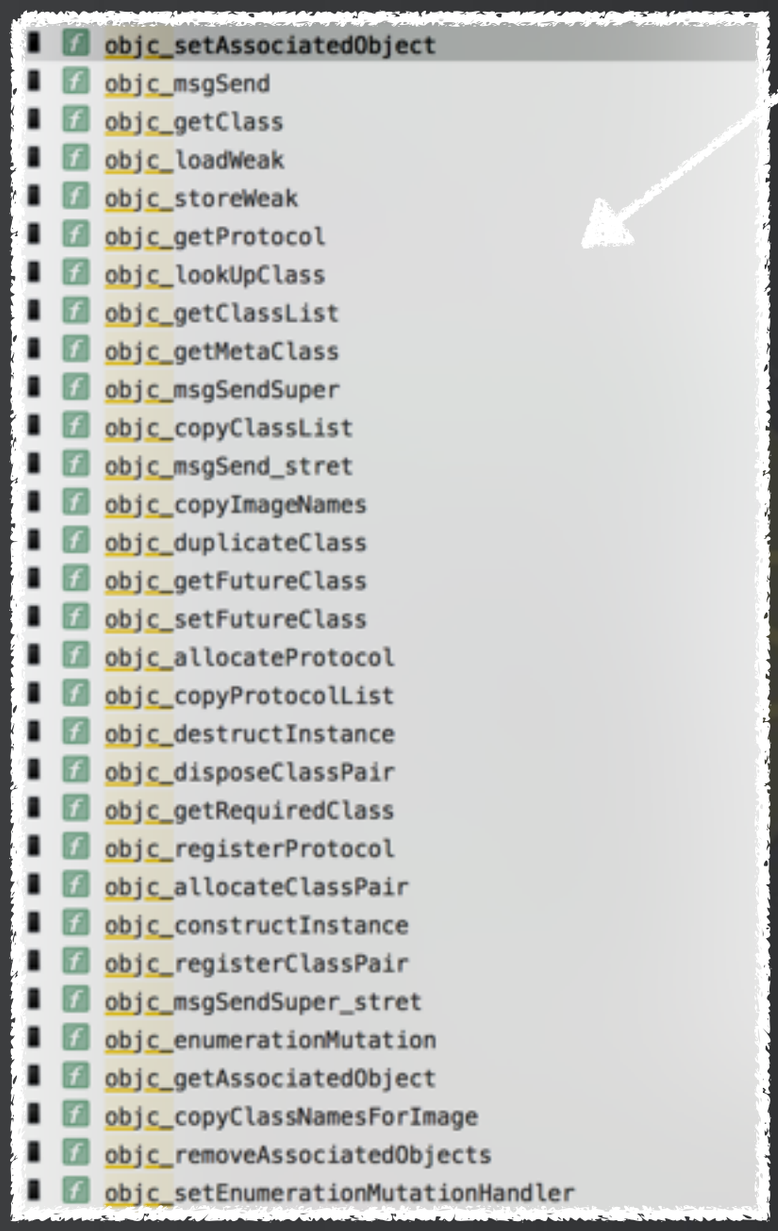
```objc
int main(int argc, const char * argv[]) {
    /* @autoreleasepool */ { __AtAutoreleasePool __autoreleasepool;

        ((NSObject *(*)(id, SEL))(void *)objc_msgSend)((id)((NSObject *(*)
            (id, SEL))(void *)objc_msgSend)((id)objc_getClass("NSObject"),
            sel_registerName("alloc")), sel_registerName("init"));
    }
    return 0;
}
```

经过编译后的中间码哦！

Runtime文件其实就在/usr/include/objc下的一组文件





Runtime System 涉及到的所有方法，
其实还真是一组C函数，
它们将会被Runtime System调用哦！

坚持啰嗦三个点哈：）：

Interacting with the Runtime

A. Objective-C Source Code

B. NSObject Methods

C. Runtime Functions

isKindOfClass:
isMemberOfClass:
respondsToSelector
conformsToProtocol

NSObject 类里面的几个动态运行时的方法

```
+ (BOOL)instancesRespondToSelector:(SEL)aSelector;
+ (BOOL)conformsToProtocol:(Protocol *)protocol;
- (IMP)methodForSelector:(SEL)aSelector;
+ (IMP)instanceMethodForSelector:(SEL)aSelector;
- (void)doesNotRecognizeSelector:(SEL)aSelector;
```

```
- (id)forwardingTargetForSelector:(SEL)aSelector OBJC_AVAILABLE(10.5, 2.0, 9.0, 1.0, 2.0);
- (void)forwardInvocation:(NSInvocation *)anInvocation OBJC_SWIFT_UNAVAILABLE("");
- (NSMethodSignature *)methodSignatureForSelector:(SEL)aSelector OBJC_SWIFT_UNAVAILABLE("");

+ (NSMethodSignature *)instanceMethodSignatureForSelector:(SEL)aSelector OBJC_SWIFT_UNAVAILABLE("");
```

```
+ (BOOL)resolveClassMethod:(SEL)sel OBJC_AVAILABLE(10.5, 2.0, 9.0, 1.0, 2.0);
+ (BOOL)resolveInstanceMethod:(SEL)sel OBJC_AVAILABLE(10.5, 2.0, 9.0, 1.0, 2.0);
```

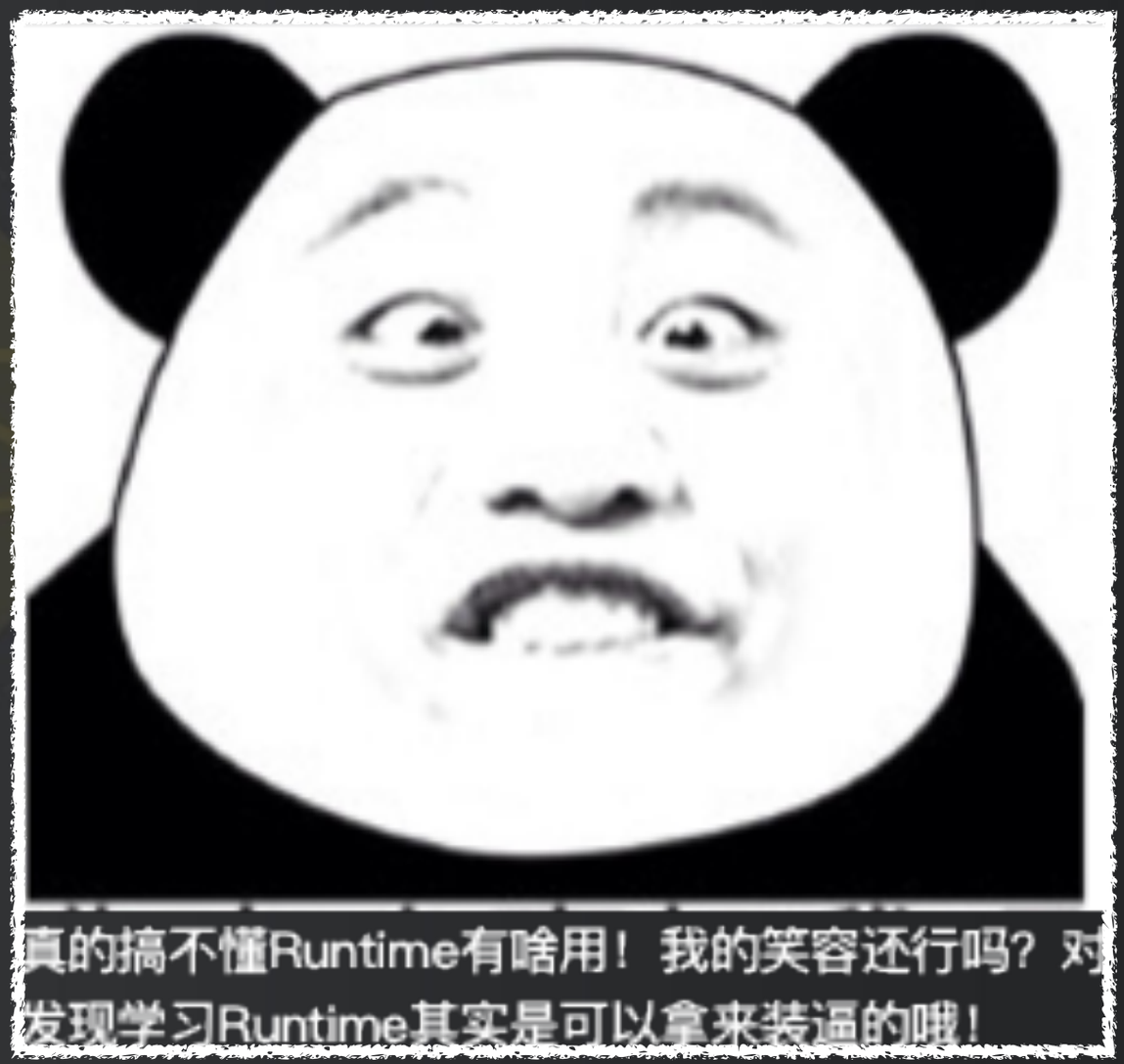
为什么要学习Runtime机制呀?

- 在开始分享之前，我也是犹豫了很长时间是否需要在团队里面分享runtime，因为runtime是一柄双刃剑，理解的好，用的好，就能写出一些高效实用的代码，用的不好，也能坑队友于无形之中。正所谓，步子大了容易扯着蛋呀。

- 学习runtime是程序员进阶的一个必要的过程，不单单是oc，所有语言的编程学习到一定的水平，就要开始研究改语言底层的编译运行机制，我们这里所说的研究，并不是值要灵活使用底层的方法，而是去了解和熟知底层运行的机制，所以只能分享一些基础的学习内容，后续大家可以自行研究

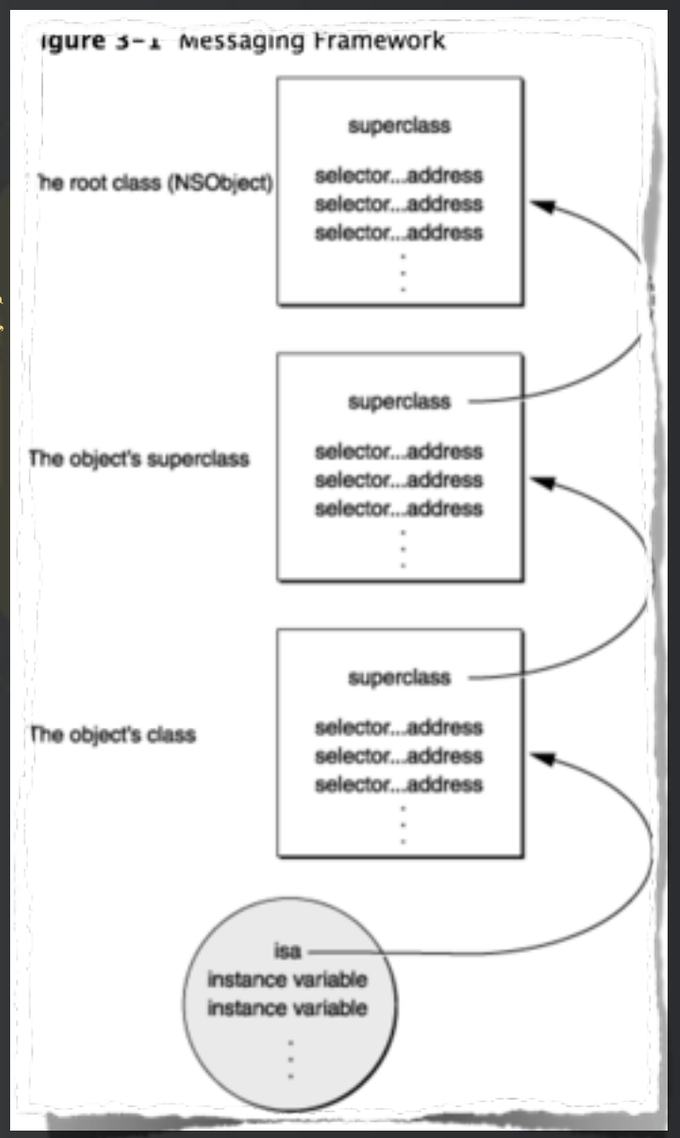
- 基本功能点：1.交换方法、2.动态添加方法、3.动态添加属性、4.字典转模型 其实好的三方库或多或少都有对runtime 运用哦！

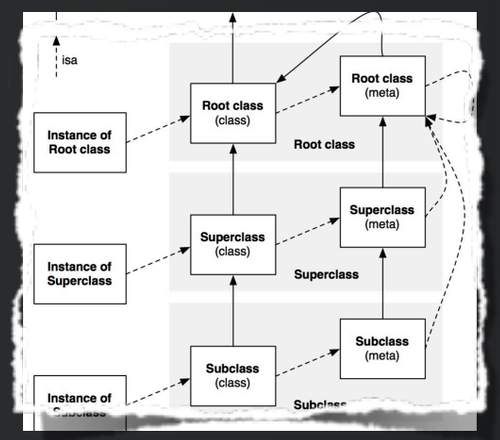
- 更好的理解、类、对象、消息机制、以及运行期间代码动态交互过程，黑魔法必备武器库！

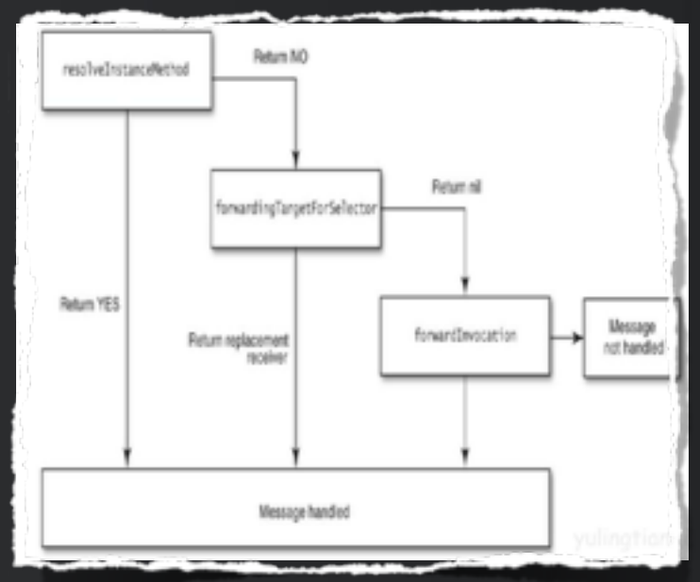


How：Runtime底层工作原理是怎样呢?

- 发送消息（Messaging）
- 动态方法解析（Dynamic Method Resolution）
- 消息转发（Message Forwarding）
- 类型编码&属性申明







发送消息（Messaging）

1. [receiver message]

编译得到

动态绑定

2. objc_msgSend(receiver, selector)

3. objc_msgSend(receiver, selector, arg1, arg2, ...)

```
id objc_msgSend ( id self, SEL op, ... );
```

此乃真身也！，哈哈

了解Runtime 数据结构！

程序设计 = 数据结构 + 算法

- id : typedef struct objc_object *id;
- SEL : typedef struct objc_selector *SEL;
- Class : 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。
- ·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。

```
struct objc_class {
    Class _Nonnull isa  OBJC_ISA_AVAILABILITY;

#if !__OBJC2__
    Class _Nullable super_class
    const char * _Nonnull name
    long version
    long info
    long instance_size
    struct objc_ivar_list * _Nullable ivars
    struct objc_method_list * _Nullable * _Nullable
    struct objc_cache * _Nonnull cache
    struct objc_protocol_list * _Nullable protocols
#endif
```

```
objc_cache
objc_super
objc_object
objc_property_t
objc_AssociationPolicy
objc_method_description
objc_property_attribute_t
```

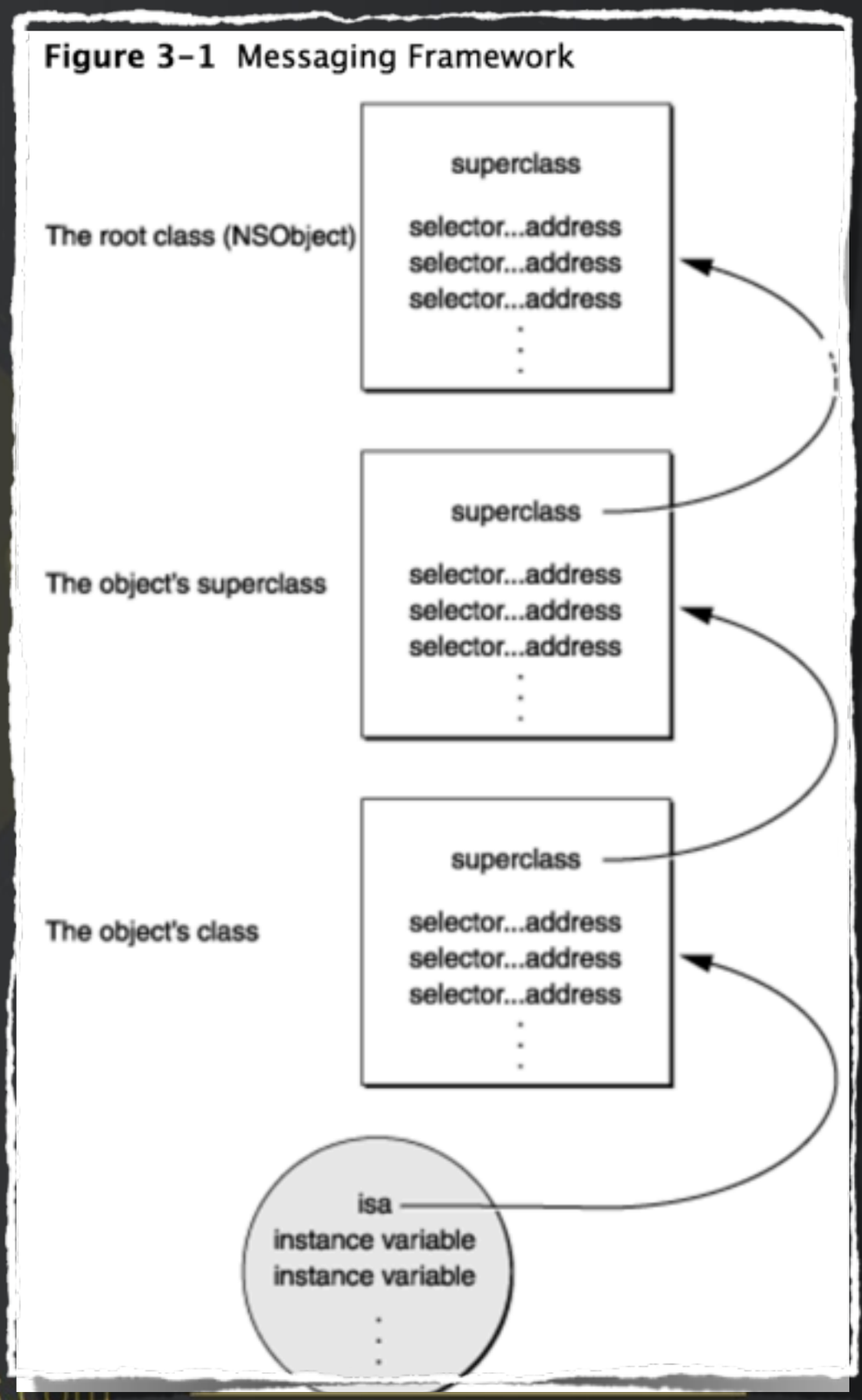
发送消息(Messaging forward)

1. 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.

2. 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（IMP）去执行。

3. 如果在类中没有找到该方法，会检查本类是否有被动态加载的方法来处理该消息。若没有，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）。

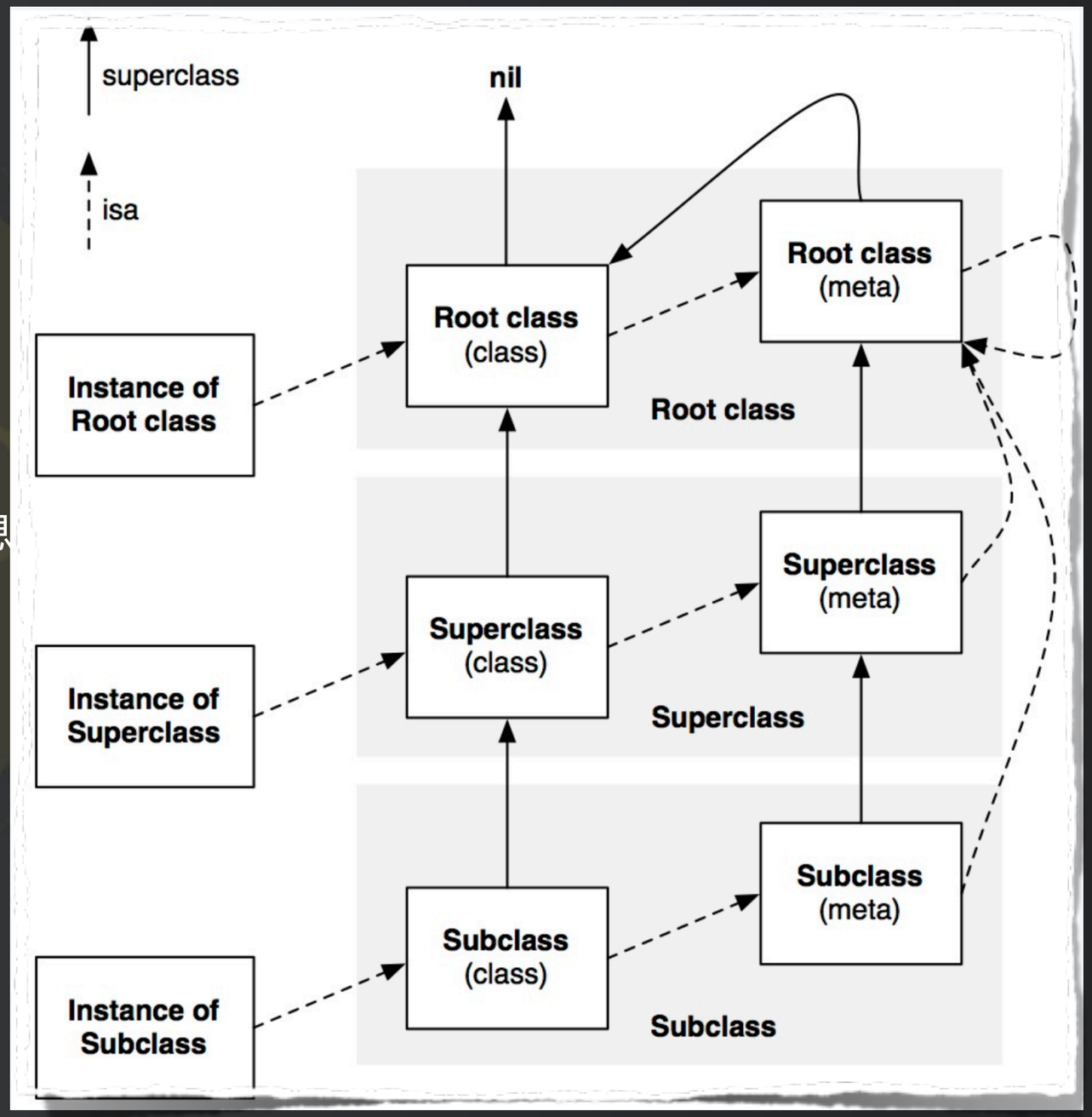
4. 前面我们说 methodLists 指向该类的实例方法列表，实例方法即-方



www.tzios.com

发送消息(Messaging forward)

从有图可以看出，Objc 类本身同时也是一个对象，为了处理类和对象的关系，runtime 库创建了一种叫做元类 (Meta Class) 的东西，类对象所属类型就叫做元类，它用来表述类对象本身所具备的元数据。类方法就定义于此处，因为这些方法可以理解成类对象的实例方法。每个类仅有一个类对象，而每个类对象仅有一个与之相关的元类。当你发出一个类似 [NSObject alloc] 的消息时，你事实上是把这个消息发给了一个类对象 (Class Object)，这个类对象必须是一个元类的实例，而这个元类同时也是一个根元类 (root meta class) 的实例。所有的元类最终都指向根元类为其超类。所有的元类的方法列表都有能够响应消息的类方法。所以当 [NSObject alloc] 这条消息发给类对象的时候，objc_msgSend() 会去它的元类里面去查找能够响应消息的方法，如果找到了，然后对这个类对象执行方法调用。



动态方法解析（Dynamic Method Resolution）

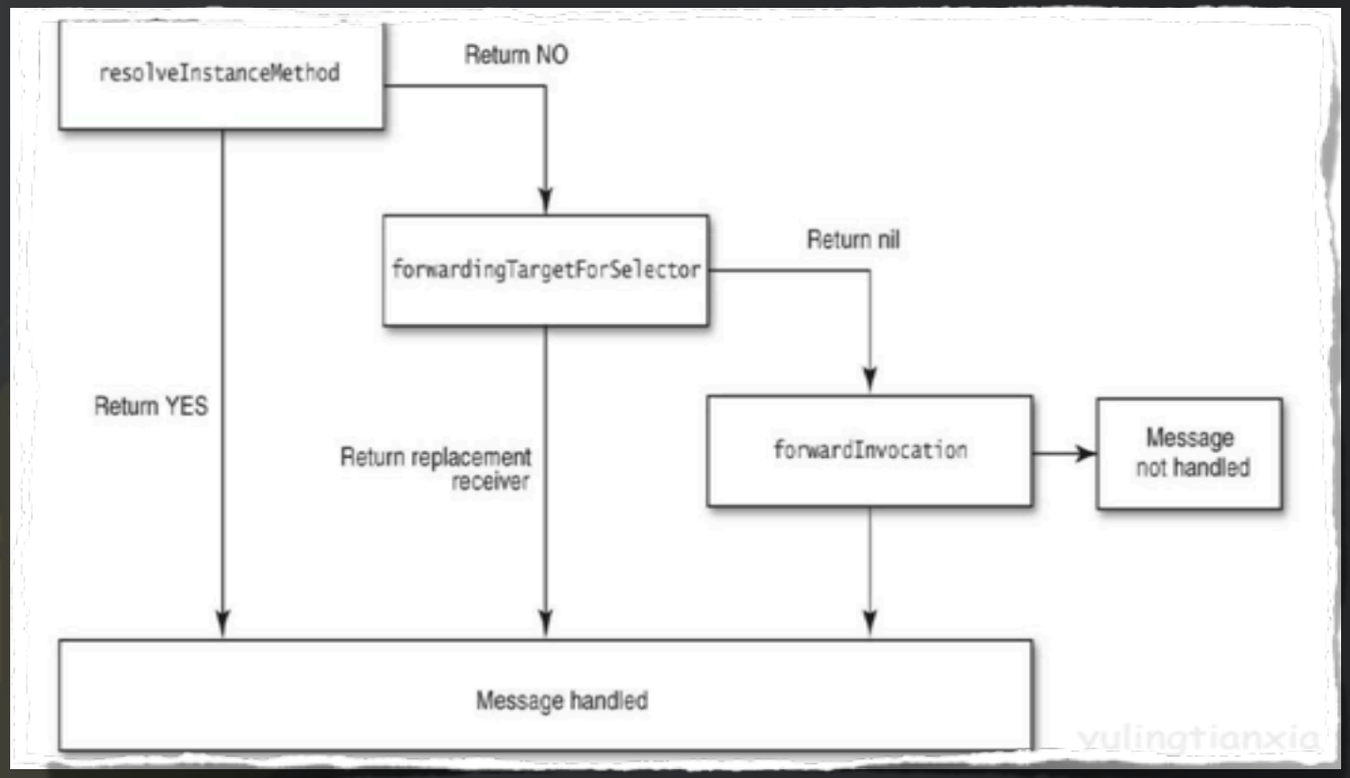
- How you can provide an implementation of a method dynamically！！！
- 在Runtime System没有在本类的method_lists没有找到匹配的实现方法时，我们可以动态的添加一个方法，这是开始进行消息转发（messaging forward）前的第一阶段，例如我们用@dynamic关键字在类的实现文件中修饰一个属性：这表明我们会为这个属性动态提供存取方法，编译器不会默认为我们生成setPropertyName:和propertyName方法，而需要我们动态提供。
- 同样我们可以通过分别重载resolveInstanceMethod:和resolveClassMethod:方法分别添加实例方法实现和类方法实现。因为当 Runtime 系统在Cache

```objc
void dynamicMethodIMP(id self, SEL _cmd) {
    // implementation ....
}
@implementation MyClass
+ (BOOL)resolveInstanceMethod:(SEL)aSEL
{
    if (aSEL == @selector(resolveThisMethodDynamically)) {
          class_addMethod([self class], aSEL, (IMP) dynamicMethodIMP, "v@:");
          return YES;
    }
    return [super resolveInstanceMethod:aSEL];
}
@end
```

这是Type Encoding 呀！

完整消息转发机制

- 消息转发分为两大阶段。第一阶段先征询接收者，所属的类，看其是否能动态添加方法，以处理当前这个“未知的选择子”，这叫做“动态方法解析”。
- 第二阶段涉及“完整的消息转发机制（full forwarding mechanism）”如果运行期系统已经执行完第一阶段，此时，运行期系统会请求我接收者以其它手段来处理消息。可以细分两小步。1.首先查找有没有replacement receiver进行处理。若无，2.运行期系统把Selector相关信息封装到NSInvocation对象中，再给一次机会，若依旧未处理则让NSObject调用doNotReconizeSelector：



- 接收者在每一步中均有机会处理消息，步骤越往后，处理消息的代价越大。最好能在第一步就处理完，这样Runtime System可以将方法缓存起来，第三步除了修改目标相比第二步还要创建处理NSInvocation。

Talking is cheap Show Me the Code

```objc
#import <Foundation/Foundation.h>

@interface Student : NSObject
+ (void)learnClass:(NSString *) string;
- (void)goToSchool:(NSString *) name;
@end
```

A. 需要深刻理解 [self class] 与 object_getClass(self) 甚至 object_getClass([self class]) 的关系，其实并不难，重点在于 self 的类型：

B. 当 self 为实例对象时，[self class] 与 object_getClass(self) 等价，因为前者会调用后者。object_getClass([self class]) 得到元类。

C. 当 self 为类对象时，[self class] 返回值为自身，还是 self。
object_getClass(self) 与 object_getClass([self class]) 等价。

D. 凡是涉及到类方法时，一定要弄清楚元类、selector、IMP 等概念，这样才能做到举一反三，随机应变。

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

@implementation Student
+ (BOOL)resolveClassMethod:(SEL)sel {
    if (sel == @selector(learnClass:)) {
        class_addMethod(object_getClass(self), sel, class_getMethodImplementation(object_getClass(self), @selec
        return YES;
    }
    return [class_getSuperclass(self) resolveClassMethod:sel];
}

+ (BOOL)resolveInstanceMethod:(SEL)aSEL
{
    if (aSEL == @selector(goToSchool:)) {
        class_addMethod([self class], aSEL, class_getMethodImplementation([self class], @selector(myInstanceMet
        return YES;
    }
    return [super resolveInstanceMethod:aSEL];
}

+ (void)myClassMethod:(NSString *)string {
    NSLog(@"myClassMethod = %@", string);
}

- (void)myInstanceMethod:(NSString *)string {
    NSLog(@"myInstanceMethod = %@", string);
}
@end
```

Runtime常用API汇总

方法	描述（Descrips）	场景
objc_setAssoci	Sets an associated value	对象关联
objc_getAssoci	Returns the value	对象关联
method_excha	Exchanges the	方法调配（method
objc_getClass	Obtains the list of	获取所有类的列表
object_getIvar	Reads the value of an	读取一个实例变量的值
class_copyPro	Describes the properties	获取属性列表（模型转字
protocol_copy	Returns an array of the	

补充环节

知识点1: Type Encoding

- To assist the runtime system, the compiler encodes the return and argument types for each method in a character string and associates the string with the method selector. The coding scheme it uses is also useful in other contexts and so is made publicly available with the @encode() compiler directive. When given a type specification, @encode() returns a string encoding that type. The type can be a basic type such as an int, a pointer, a tagged structure or union, or a class name—any type, in fact, that can be used as an argument to the C sizeof() operator.

知识点2:关于动态加载 Dynamic Loading

- An Objective-C program can load and link new classes and categories while it's running. The new code is incorporated into the program and treated identically to classes and categories loaded at the start.Dynamic loading can be used to do a lot of different things. For example, the various modules in the System Preferences application are dynamically loaded.
- Although there is a runtime function that performs dynamic loading of Objective-C modules in Mach-O files (objc_loadModules, defined in objc/objc-load.h), Cocoa's NSBundle class provides a significantly more convenient interface for dynamic loading—one that's object-oriented and integrated with related services. See the NSBundle class specification in the Foundation framework reference for information on the NSBundle class and its use. See OS X ABI Mach-O File Format Reference for information on Mach-O files.

www.tzios.com

where: 常见的Runtime应用场景在哪里呢?

1. 实现第一个场景：跟踪程序每个ViewController展示给用户的次数，可以通过Method Swizzling替换ViewDidAppear初始方法。创建一个UIViewController的分类，重写自定义的ViewDidAppear方法，并在其+load方法中实现ViewDidAppear方法的交换
2. 开发中常需要在不改变某个类的前提下为其添加一个新的属性，尤其是为系统的类添加新的属性，这个时候就可以利用Runtime的关联对象（Associated Objects）来为分类添加新的属性了
3. 三实现字典的模型和自动转换，优秀的JSON转模型第三方库JSONModel、YYModel等都利用runtime对属性进行获取，赋值等操作，要比KVC进行模型转换更加强大，更有效率。阅读YYModel的源码可以看出，YY大神对NSObject的内容进行了又一次封装，添加了许多描述内容。其中YYClassInfo是对Class进行了再次封装，而YYClassIvarInfo、YYClassMethodInfo、YYClPropertyInfo分别是对Class的Ivar、Method和property进行了封装和描述。在提取

看我，看我，还在思考Runtime问题：）

相关资料

官方文档：Object-c runtime Programing guide

书籍：《Effective object-c 2.0》

博客：http://yulingtianxia.com/blog/2014/11/05/objective-c-runtime/

开源项目YYkit: https://github.com/ibireme/YYKit

iOS

Thanks For Your Attention!

感谢观看!

http://www.tzios.com/