Runtime

参考掘金博客
参考掘金博客2

什么是Runtime?

源代码转换为可执行的程序，通常要经过三个步骤：**编译**、**链接**、**运行**。不同的编译语言，在这三个步骤中所进行的操作又有些不同。
C语言：静态语言，在编译阶段已经确定了所有变量的数据类型，以及函数实现；
OC在C的基础上加入了面向对象特性和消息传递机制，这样能使得OC变得更加灵活。这一切的基础就是`Runtime`。
Runtime实际上是一个库，使我们可以在程序运行时动态的创建对象、检查对象，修改类和对象的方法。

消息传递

一个对象的方法像这样`[obj foo]`，编译器转成消息发送`objc_msgSend(obj, foo)`，`Runtime`时执行的流程是这样的：

- 首先，通过`obj`的`isa`指针找到它的`class`；
- 在`class`的`method list`找`foo`；
- 如果`class`中没到`foo`，继续往它的`superclass`中找；
- 一旦找到`foo`这个函数，就去执行它的实现`IMP`。

这个过程中还有一个cache缓存，因为少部分方法被大部分调用，重复遍历所有method很耗时。

概念解析

objc_msgSend

所有oc方法在编译期会转化为`objc_msgSend(receiver,selector)`的调用。

Class 类对象

在`objc/runtime.h`中，`Class（类）`被定义为指向`objc_class 结构体`的指针。
`struct objc_class`结构体定义了很多变量，通过命名不难发现， 结构体里保存了指向父类的指针、类的名字、版本、实例大小、实例变量列表、方法列表、缓存、遵守的协议列表等， 一个类包含

的信息也不就正是这些吗？没错，类对象就是一个结构体`struct objc_class`，这个结构体存放的数据称为元数据(`metadata`)， 该结构体的第一个成员变量也是`isa`指针，这就说明了`Class`本身其实也是一个对象，因此我们称之为类对象，**类对象在编译期产生用于创建实例对象，是单例**。

Object 对象

对象被定义为`objc_object`，含有一个指向它所属类的isa指针。

Meta Class 元类

object对象的isa指针指向对应的类对象，那么类对象的isa指针指向什么呢？实际指向的是自身的`meta class元类`。

那么类方法的调用过程和对象方法的调用过程差不多，流程如下：

1. 通过类对象的isa指针找到所属的meta class；
2. 在meta class的method list中找到对应的selector；
3. 执行对应的selector。

Method

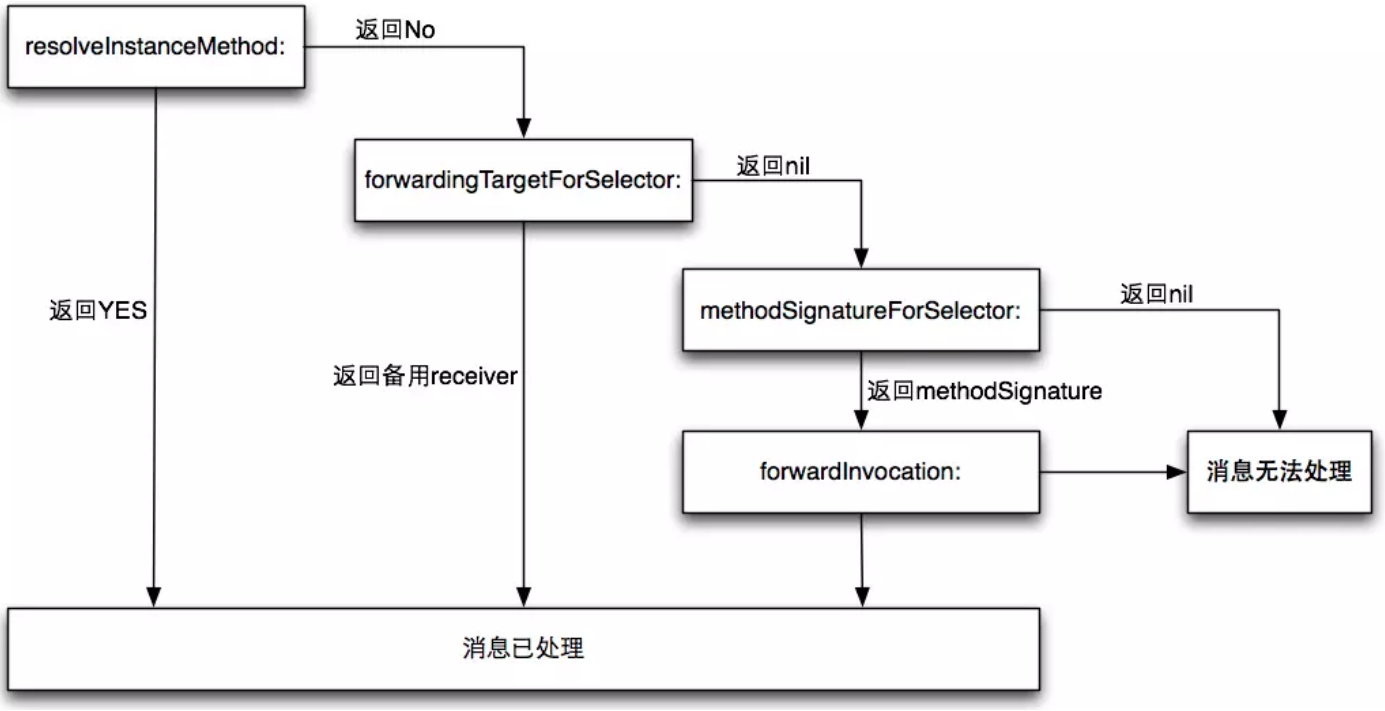
- SEL 方法名，本质指向objc_selector的指针，是一个保存方法名的字符串
- IMP 方法实现，本质是一个函数指针，指向方法的实现
- method_type，本质是个字符串，用来存储方法的参数类型和返回值类型

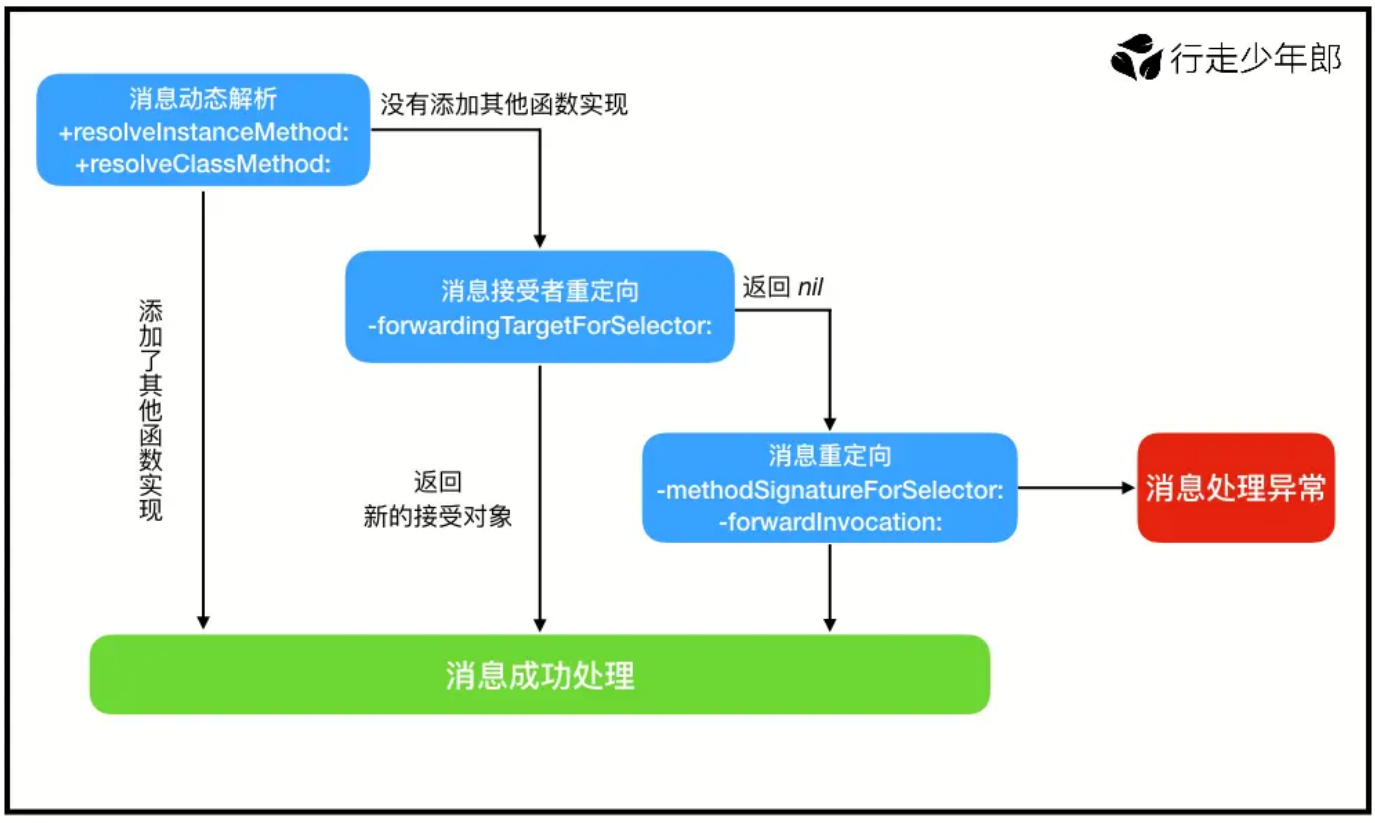
objc_cache

为加速消息分发，系统会对方法和对应的地址进行缓存，实际运行的速度非常快。

消息转发

消息转发流程简图：





关于消息动态解析示例：可看博客。

消息发送以及转发机制的总结

来自参考掘金博客，在top有链接。

调用 `[receiver selector];` 后，进行的流程：

1. 编译阶段：`[receiver selector];` 方法被编译器转换为:
   a. `objc_msgSend(receiver, selector)` （不带参数）
   b. `objc_msgSend(recevier, selector, org1, org2, …)` （带参数）
2.
3. 运行时阶段：消息接受者 `recever` 寻找对应的 `selector`。
   a. 通过 `recevier` 的 `isa 指针` 找到 `recevier` 的 `class（类）`；
   b. 在 `Class（类）` 的 `cache（方法缓存）` 的散列表中寻找对应的 `IMP（方法实现）`；
   c. 如果在 `cache（方法缓存）` 中没有找到对应的 `IMP（方法实现）` 的话，就继续在 `Class（类）` 的 `method list（方法列表）` 中找对应的 `selector`，如果找到，填充到 `cache（方法缓存）` 中，并返回 `selector`；
   d. 如果在 `class（类）` 中没有找到这个 `selector`，就继续在它的 `superclass（父类）` 中寻找；
   e. 一旦找到对应的 `selector`，直接执行 `recever` 对应 `selector` 方法实现的 `IMP（方法实现）`。
   f. 若找不到对应的 `selector`，Runtime 系统进入消息转发机制。
4.
5. 运行时消息转发阶段：
   a. 动态解析：通过重写 `+resolveInstanceMethod:` 或者 `+resolveClassMethod:` 方法，利用 `class_addMethod` 方法添加其他函数实现；
   b. 消息接受者重定向：如果上一步添加其他函数实现，可在当前对象中利用 `-forwardingTargetForSelector:` 方法将消息的接受者转发给其他对象；
   c. 消息重定向：如果上一步没有返回值为 `nil`，则利用 `-methodSignatureForSelector:` 方法获取函数的参数和返回值类型。
      i. 如果 `-methodSignatureForSelector:` 返回了一个 `NSMethodSignature` 对象（函数签名），Runtime 系统就会创建一个 `NSInvocation` 对象，并通过 `-forwardInvocation:` 消息通知当前对象，给予此次消息发送最后一次寻找 IMP 的机会。
      ii. 如果 `-methodSignatureForSelector:` 返回 `nil`。则 Runtime 系统会发出 `-doesNotRecognizeSelector:` 消息，程序也就崩溃了。

Runtime应用

参考掘金博客2 讲的很详细

关联对象给分类增加属性

关联对象（Objective-C Associated Objects）给分类增加属性。

方法添加和替换和KVO实现

方法添加

在消息转发的时候就提到了

```
1 //class_addMethod(Class  _Nullable __unsafe_unretained cls, SEL  _Nonnull name,
  IMP  _Nonnull imp, const char * _Nullable types)
2 class_addMethod([self class], sel, (IMP)fooMethod, "v@:");
3 // cls 被添加方法的类
4 // name 添加方法的名称SEL
5 // imp 方法实现
6 // 类型编码
```

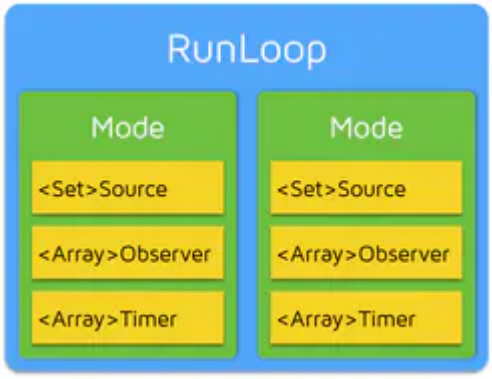
（其他的先略过，有兴趣再去看）

RunLoop

参考博客：掘金博客

一个Runloop对象，主要包含了一个线程，若干个Mode，若干个commonMode，还有一个当前运行的Mode。

Mode可以视为事件的管家，一个Mode管理各种事件。一个Mode对象有一个name，若干source，timer，observer和若干port，事件都是Mode在管理，而Runloop管理Mode。



如图所示，Runloop Mode 实际上是 Source，Timer 和 Observer 的集合，不同的 Mode 把不同组的 Source，Timer 和 Observer 隔绝开来。Runloop 在某个时刻只能跑在一个 Mode 下，处理这一个 Mode 当中的 Source，Timer 和 Observer。

NSTimer

参考博客：NSTimer的用法

一个 NSTimer 注册到 RunLoop 后，RunLoop 会为其重复的时间点注册好事件。例如 10:00, 10:10, 10:20 这几个时间点。RunLoop为了节省资源，并不会在非常准确的时间点回调这个Timer。Timer 有个属性叫做 Tolerance (宽容度)，标示了当时间点到后，容许有多少最大误差。由于 NSTimer 的这种机制，因此 NSTimer 的执行必须依赖于 RunLoop，如果没有 RunLoop，NSTimer 是不会执行的。