Mr.Wu-小子
编辑于 2024-09-27 09:57:24 · 繁简文记录
49 阅读 · 0 评论 · 0 点赞 · 0 收藏 · 0 分享
1370 · 11 · 55 · 5 · 142
粉丝 · 积分 · 活跃 · 原创 · 周榜 · 总榜
私信 · 关注

2024年度CSDN个人总结
生成时间：2024年12月31日
查看我的2024年度总结

搜索文章

热门文章
AwesomePaper 3279
内存对齐，内存对齐规则解析，内存对齐原理 757
AwesomeRecorder 原型 1603
Mac系统环境配置：flutter、Android Studio、Java等环境变量配置 3685
Could not load the "_libbrotlicommon_x" image referenced from a nib in the bundle with identifier "com... 5385

最新评论
内存对齐，内存对齐规则解析，内存对齐... 1 hour ago
MRDS 源码原理篇：一个不能模仿内存的实现 1 hour ago
内存对齐，内存对齐规则解析，内存对齐... 1 hour ago
Tyrrrz/xxxxxx 最后更新是2年之前了(3.1.1320)，还是在2年之前了(3.1.1320) 1 hour ago
内存对齐，内存对齐规则解析，内存对齐... 1 hour ago
@FSDaxFly @xiaoXiaoWu_ 1 hour ago
内存对齐，内存对齐规则解析，内存对齐... 1 hour ago
chi13024404th 引用：「Discode默认对齐系数8」，请问Discode默认是多少？ 1 hour ago
Mac环境配置 4 days ago
Tesla.__ 什么叫做总结呢！ 1 hour ago

大家在看
常量的作用域与生命周期
【基础函数学习系列3: 0991.1.1世纪 (L1 None)】 1 day ago
北美新能源龙头吉利Geely公证开化美国雷霆 1 day ago
【NoSQL】简单掌握数据类型与表操作，让数据库性能飞跃 1 day ago
分布式文件系统HDFS

最新文章
IDE 启动优化
iOS多线程【pthread、NSThread】总结
加密总结【OpenSSL、对称加密（DES、3DES、AES）、哈希加密算法（MD、sha家族）、HMAC加密方案、数字签名、自签证书】

2020年 4篇 · 2019年 22篇
2018年 28篇 · 2017年 3篇
2016年 1篇

Runtime（一）基础知识

Mr.Wu-小子 关注
2019-04-07 09:57:24 阅读量: 2.1k 点赞数: 0 收藏数: 0
分类专栏：iOS 文章标签：Runtime基础知识 Runtime运行原理 消息转发机制
编辑

iOS 专栏收录该博客

0 订阅 · 43 篇文章 · 订阅专栏

本文介绍Runtime的基础知识

文章目录：

一、如何理解OC底层动态语言，Runtime又是什么？
二、分析Runtime的作用及原理？
三、深入理解Runtime底层实现原理
四、Runtime消息转发原理总结

一、如何理解OC底层动态语言，Runtime又是什么？

源代码1 解释为可执行的程序，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很不同。

C语言语言：它的语言，编译阶段将要决定调用哪个函数，如果函数未实现会报错。

OC动态语言：OC的语言，编译阶段并不能决定真正调用哪个函数，只要函数声明过即使没有实现也不会报错。

我们常说的一句话“OC动态语言，就是因为它在底层一些决定性的工作从编译阶段推迟到运行时阶段，OC代码的运行不仅要编译器，还需要运行时系统(Runtime System)来保证程序的正确。

Runtime是一套底层纯C语言API，OC代码最终都会被编译器转化为运行时代码，通过调用底层API来完成调用方式，这也是OC作为动态语言实现的基础。

二、分析Runtime的作用及原理？

1、数据结构分析

新建一个工程，工程中引入头文件：

```
#import <objc/runtime.h>
#import <objc/message.h>
```

“Command + 鼠标点击”，进入Runtime的源码文件，搜下来分析OC代码在中间过程的结构，最终我们不对可以代码时加调的类，对象，方法，协议在Runtime中是怎样存在。

```
1 struct objc_class {
2 Class _Nonnull isa  OBJC_ISA_AVAILABILITY; // objc_class 结构体的实例指针
3
4 #if !__OBJC2__
5 Class _Nullable super_class;  // 指向父类的指针
6 const char * _Nonnull name;   // 类的名字
7 long version;                 // 类的版本信息，默认为 0
8 long info;                    // 类的状态，供运行时使用的一些位标识
9 long instance_size;           // 该类的实例变量大小
10 struct objc_ivar_list * _Nullable ivars;  // 该类的实例变量列表
11 struct objc_method_list * _Nullable * _Nullable methodLists;  // 方法定义的列表
12 struct objc_cache * _Nonnull cache;
13 struct objc_protocol_list * _Nullable protocols;  // 遵守协议的列表
14 #endif
15
```

光大概其实一下Runtime的大文件中引用的相关方法，再具体分析都有什么作用

2. Runtime中的数据结构

2.1 objc_msgSend

```
1 对象 objective-C 方法调用在编译时都会转化为对 C 函数 objc_msgSend 的调用，objc_msgSend(receiver, selector), 第一个
```

2.2 Class（类）

```
1 objc_class 结构体 定义了很多变量，包括的结构实例变量 (ivars)，所有方法定义 (methodLists)，遵守协议的列表 (protocols)
2 objc_class 结构体 同样一个类实变量 isa 指针，isa 指针 指向的是该类实例对象的结构体定义的指针，这是某个的类 objc_class 
```

2.3 Object（对象）

```
1 这里的 id 被定义为一个指向 objc_object 结构体 的指针，此外可以看出 objc_object 结构体 只包含一个 Class 类型的 isa 指
2 换句话说，一个 Object (对象) 唯一指向的就是它所属 Class (类) 的指针，而其它的一个字段，进行方法调用时，实现 [receiver s
```

2.4 Meta Class（元类）

```
1 对象 objc_object 结构体 的 isa 指针 指向的是对应的 类对象 objc_class 结构体，那么，那对象 objc_class 结构体 的
2 objc_class 结构体 的 isa 指针 指向的是 类对象 指向 Meta Class。
```

那么什么是 Meta Class（元类）？

Meta Class（元类）就是一个类对象所属的 类，一个对象所属的类叫做 类对象，而一个类对象所属的类叫做 元类。

2.5 方法（Method）

```
1 struct objc_method {
2  SEL _Nonnull method_name  // 方法的名字
3  char _Nullable method_types  // 方法的类型
4  IMP _Nonnull method_imp  // 指向实现的指针
5 }
```

三、深入理解Runtime消息发送原理

1、消息转发的基本原理

Objective-C 语言 中，对象方法调用都是类似 [receiver selector]; 的形式，其本质就是对对象在运行时发送消息的过程。

我们来看看方法调用 [receiver selector]; 在 *编译阶段*，和 *运行阶段*，分别做了什么？

编译阶段：[receiver selector]; 方法被编译器转换为：

```
1 objc_msgSend(receiver, selector)（不带参数）
2 objc_msgSend(receiver, selector, arg1, arg2, ...)（带参数）
```

运行时阶段：消息接受者 receiver 将找对应的 selector。

```
1 通过 receiver 的 isa 指针 找到 receiver 的类 Class 类
2 在 Class 类 的 cache（方法缓存）中查找 selector 的实现方法 IMP（方法实现）
3 如果在 cache 中找到 selector 对应的实现方法 IMP（方法实现），则调 缓存的 Class 类 的 method list（方法列表）中找对应
4 如果在 Class 类 中没有找到这个 selector，则继续在它的 superClass（父类）中寻找；
5  一旦找到了对应的 selector，直接执行 receiver 对应 selector 方法实现的 IMP（方法实现）
6 找到了对应的方 selector，消息接收变成等待对象 receiver 通知这个 selector 对应的实现方法，直到消息发送结束。
```

2、消息转发

2.1消息动态解析

举例：创建 T-MessageSendM_ 声明一个方法 -m 文件不写实现，然后调用这个方法。

```
1 @interface MessageSend : NSObject
2 [selfsendSelfMessage:[NSString -img]
3 @end
```

毫无悬念的崩了，报错

```
1 [2019-08-03 23:21:20.950576-0800 Runtime[844:378585] -[MessageSend sendMessage:]: unrecognized selector sen
```

下面动态添加方法，即调用另一个方法

先添加一个方法

```
1 void dynamicMethodIMP(id self, SEL _cmd, NSString -msg) {
2  NSLog(@"动态添加方法——%@%@%@", self, NSStringFromSelector(_cmd), msg);
3 }
```

然后重写系统的 resolveClassMethod:(SEL)sel 或者 resolveInstanceMethod:(SEL)sel 方法，我们声明的属性方法，并以最写这个：

```
1 +(BOOL)resolveClassMethod:(SEL)sel {//实现
2
3  //@param .       接收个类类型方法
4  //@param sel     方法选择器名称，方法编号
5  //@param .       方法实现
6  //@param .       参数
7  //@return        如果函数实现成功返回 YES，否则返回 NO
8  //
9  //BOOL class_addMethod(Class cls, SEL name, IMP imp, const char * _Nullable types);
10
11  if (sel == @selector(receiveMessage:)) {
12  return class_addMethod(self.class), sel, (IMP)dynamicMethodIMP, "@@:");
13  }
14  //没有匹配方法直接进行转发
15 //  NSString -msgName = NSStringFromSelector(sel);
16 //  if [msgName isEqualToStringwithMessage:"]) {
17 //      return class_addMethod(self.class), sel, (IMP)dynamicMethodIMP, "@@:");
18 //  }
19  return [super resolveInstanceMethod:sel];;
20 }
21
```

再将的参数类型可查看官方文档：传送门

运行结果：

```
1 [2019-08-03 23:38:19.787108-0800 Runtime[851:379263] -MessageSend: 0x10bc387c0 转发到注意 — receiveMessage:
```

可以看到程序运行并且成功调用了 dynamicMethodIMP 方法，说明消息转发成功。

2.2消息接受者重定向

如果上一步中 +resolveInstanceMethod: 或者 +resolveClassMethod: 没有添加其他动态实现，运行时就会进行下一步：消息接受者重定向。

如果当前对象实现了 -/+forwardingTargetForSelector:，Runtime 就会调用这个方法，允许我们将消息的接受者转发给其他对象。

```
1 -(id)forwardingTargetForSelector:(SEL)aSelector {
2  NSString -methodName = NSStringFromSelector(aSelector);
3  if (methodName isEqualToString:@"sendMessage:") {
4  return [MessageSendImpl alloc] init];
5  }
6  return [super forwardingTargetForSelector:aSelector];
7 }
```

PersonMessage.m中

```
1 [-(id)sendMessage:(NSString -msg);
2 @end]
```

结果

```
1 [2019-08-04 00:41:37.243994-0800 Runtime[905:382458] PersonMessage
```

2.3消息重定向

如果经过消息动态解析，消息接受者重定向，Runtime 系统还是找不到相应的方法实现而无法响应消息，Runtime 系统会利用 methodSignatureForSelector: 方法获取函数的参数和返回值类型。

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，给予最后次发送最后一次转发的机会 IMP 的机会。

如果 -methodSignatureForSelector: 返回 nil，则 Runtime 系统会发送 -doesNotRecognizeSelector: 消息，程序也就崩溃了。

```
1 -(void)forwardInvocation:(NSInvocation -)anInvocation {
2  SEL sel = anInvocation.selector;  // 从 anInvocation 中获取消息
3
4  PersonMessage -p = [PersonMessage alloc] init];
5
6  if(p respondsToSelector:sel) {  // 判断 Person 对象方法是否可以响应 sel
7  [anInvocation invokeWithTarget:p];  // 修改可以响应，则将消息转发给该对象处理
8  } else {
9  [self doesNotRecognizeSelector:sel];  // 未找到方法响应，抛异常，并不再继续传递
10  }
11 }
12
13 -(void)doesNotRecognizeSelector:(SEL)aSelector {
14  NSLog(@"doesNotRecognizeSelector");
15 }
```

3、消息发送以及转发机制总结

调用 [receiver selector]; 后，进行的流程：

编译阶段：[receiver selector]; 方法被编译器转换为：

```
objc_msgSend(receiver, selector)（不带参数）
objc_msgSend(receiver, selector, arg1, arg2, ...)（带参数）
```

运行时阶段：消息接受者 receiver 将找对应的 selector。

通过 receiver 的 isa 指针 找到 receiver 的 class（类）；

在 Class（类）的 cache（方法缓存）的散列表中寻找对应的 IMP（方法实现）；

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，调用到 cache（方法缓存）中，并通过 selector；

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；

一旦找到了对应的 selector，直接执行 receiver 对应 selector 方法实现的 IMP（方法实现）；

若找不到对应的 selector，Runtime 系统进入消息转发机制。

运行时消息转发阶段：

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；

消息接受重定向：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；

消息重定向：如果上一步没有函数实现 nil，则利用 -methodSignatureForSelector:方法获取函数的参数和返回值类型，

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，给予最后次机会。

如果 -methodSignatureForSelector: 返回 nil，则 Runtime 系统会发送 -doesNotRecognizeSelector:消息，程序也就崩溃了。

【Java基础笔记】Java入门级概述 李Java、前端向 10.5k+
CSDN最强Java专栏，包含全面Java基础知识点，Java进阶博客，Java面试，Java多线程，Java并发实现，常见Java的经验学习。
【iOS】消息传递机制原理及转发机制 m0_63692268的博客 4k
在苹果系统的Mac或iOS系统中，为保证不崩溃的情况下实现Objective-C语言的动态特性，苹果提供了Runtime运行时系统，Runtime系统是一个用C语言编写的库，它为Objective-C语言的动态特性提供了支持。Runtime系统的核心是消息传递（Messaging）机制，Objective-C语言中的对象方法调用，都是通过Runtime系统来发送消息来实现的。
iOS开发之进阶篇(四)——runtime运行时_runtimeid 2-2
对象runtime实现自动化，自动化的runtime运行时，是一个以内置方法，动态调用方法来为对象的一些属性和数据结构赋值，例如属性的，给予属性的，给予属性的，给予