

# 目录

前言	1.1
ObjC运行时概览	1.2
Runtime基础知识	1.3
ObjC函数和对象	1.4
NSObject	1.4.1
objc系列函数	1.4.2
objc_msgSend	1.4.2.1
object系列函数	1.4.3
其他心得	1.5
附录	1.6
参考资料	1.6.1

# iOS逆向开发：ObjC运行时

- 最新版本: v0.7
- 更新时间: 20230710

## 简介

介绍iOS逆向期间常会涉及到的ObjC的底层知识Runtime运行时。以及介绍Runtime中的常遇到的各个函数和内容，包括NSObject、objc开头的系列函数，尤其是objc\_msgSend函数、object开头的系列函数。

## 源码+浏览+下载

本书的各种源码、在线浏览地址、多种格式文件下载如下：

### HonKit源码

- [crifan/ios\\_re\\_objc\\_runtime: iOS逆向开发：ObjC运行时](#)

### 如何使用此HonKit源码去生成发布为电子书

详见：[crifan/honkit\\_template: demo how to use crifan honkit template and demo](#)

### 在线浏览

- [iOS逆向开发：ObjC运行时 book.crifan.org](#)
- [iOS逆向开发：ObjC运行时 crifan.github.io](#)

### 离线下载阅读

- [iOS逆向开发：ObjC运行时 PDF](#)
- [iOS逆向开发：ObjC运行时 ePUB](#)
- [iOS逆向开发：ObjC运行时 Mobi](#)

## 版权和用途说明

此电子书教程的全部内容，如无特别说明，均为本人原创。其中部分内容参考自网络，均已备注了出处。如发现有侵权，请通过邮箱联系我 [admin 艾特 crifan.com](mailto:admin@crifan.com)，我会尽快删除。谢谢合作。

各种技术类教程，仅作为学习和研究使用。请勿用于任何非法用途。如有非法用途，均与本人无关。

## 鸣谢

感谢我的老婆陈雪的包容理解和悉心照料，才使得我 [crifan](#) 有更多精力去专注技术专研和整理归纳出这些电子书和技术教程，特此鸣谢。

## 其他

### 作者的其他电子书

本人 crifan 还写了其他 150+ 本电子书教程，感兴趣可移步至：

[crifan/crifan\\_ebook\\_readme: Crifan的电子书的使用说明](#)

## 关于作者

关于作者更多介绍，详见：

[关于CrifanLi李茂 – 在路上](#)

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新：2023-07-10 10:46:31

# ObjC运行时概览

iOS逆向时，常会涉及到底层的一些机制，其中就包括，objc的运行时 = Runtime。

iOS逆向期间涉及到的很多内容，都和Runtime有关：

- 逆向技术
  - Method Swizzling
    - 底层就依赖于ObjC的Runtime机制
  - 导出头文件
    - 据说底层机制就依赖于ObjC的Runtime
    - 据说，如果代码换成Swift，就无法导出头文件
- 动态调试
  - 可以输出类的属性和函数
    - 底层就涉及到，Runtime中的 `NSObject`、`isa` 等内容

crifan.org, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved, powered by Gitbook最后更新：2022-10-24 17:37:58

# Runtime基础知识

此处整理iOS的ObjC的Runtime的相关基础知识。

- ObjC Runtime

- 官网文档
    - [Objective-C Runtime | Apple Developer Documentation](#)

- 概述
    - The Objective-C runtime is a runtime library that provides support for the dynamic properties of the Objective-C language, and as such is linked to by all Objective-C apps. Objective-C runtime library support functions are implemented in the shared library found at `/usr/lib/libobjc.A.dylib`.
    - You typically don't need to use the Objective-C runtime library directly when programming in Objective-C. This API is useful primarily for developing bridge layers between Objective-C and other languages, or for low-level debugging

TODO:

- 【整理】iOS运行时iOS Runtime基础知识
- 其他资料:
  - Swift & the Objective-C Runtime - NSHipster
    - [英文](#)
    - [中文](#)

## load

TODO:

【未解决】iOS的ObjC基础知识: load方法

## objc\_classlist 和 objc\_nlcatlist

在 `objc-file.mm` 文件中存有以下定义

```
// 类似于 C++ 的模板写法，通过宏来处理泛型操作
// 函数内省是从内存数据段的某个区下查询该位置的情况，并回传指针
#define GETSECT(name, type, sectname)
    type *name(const header_type mhdr, size_t outCount) {
        return getDataSection<type>(mhdr, sectname, nil, outCount);
    }
    type *name(const header_info hi, size_t outCount) {
        return getDataSection<type>(hi->mhdr(), sectname, nil, outCount);
    }
// 根据 dyld 对 images 的解析来在特定区域查询内存
GETSECT(_getObjc2ClassList, classref_t, "__objc_classlist");
GETSECT(_getObjc2NonLazyCategoryList, category_t, "__objc_nlcatlist");
```

-> 才明白：

```
+ (void)load
```

对于每一个 Class 和 Category 来说，必定会调用此方法，而且仅调用一次。当包含 Class 和 Category 的程序库载入系统时，就会执行此方法，并且此过程通常是在程序启动的时候执行

->

- `__objc_classlist` : `class = Class类`
- `__objc_nlcatlist` : `nlcat = Non Lazy Category = 非懒加载的Category`

## iOS逆向技术：Method Swizzling

iOS逆向中的一种技术叫： Method Swizzling = 方法交换，其底层就是利用了ObjC的Runtime的特性

TODO:

【记录】研究抖音越狱检测逻辑：iOS的ObjC的方法交换Method Swizzling

## 越狱检测和反越狱检测

TODO:

【已解决】iOS越狱检测和反越狱检测：iOS运行时iOS Runtime

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新：2022-10-24 17:38:39

# ObjC函数和对象

TODO:

- 【整理】iOS逆向：ObjC底层函数objc\_alloc\_init
- 

此处整理iOS的ObjC的Runtime相关的函数和对象，和底层机制。

## dispatch相关

### dispatch\_once

TODO:

- 【已解决】iOS基础知识：dispatch\_once

**SEL = objc\_selector**

TODO:

- 【未解决】iOS基础知识：SEL的struct objc\_selector结构体的定义
  - 【已解决】iOS从SEL的selector得到和打印出函数名

### OS\_dispatch\_queue

TODO:

- 【已解决】iOS基础知识：OS\_dispatch\_queue

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新：2023-07-10 10:33:19

# NSObject

TODO:

【整理】iOS逆向心得：Runtime运行时NSObject对象

---

此处介绍iOS的ObjC中和Runtime有关的：`NSObject` 类相关的内容。

## 调试时看到的类的名称和正向开发时所用的类名不一样

比如：

- 正向开发时写的类名是：`NSDictionary`
- 逆向调试时看到的类名是：`_NSDictionaryM`

具体解释，详见：

TODO:

【整理】iOS逆向心得：调试时看到的ObjC的底层数据类型和上层类型的对应关系

crifan.org, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved, powered by Gitbook最后更新：2022-10-24 17:01:48

# objc系列函数

此处介绍iOS的ObjC的Runtime中，常见的，以 `objc` 开头的一些系列函数。

其中对应官网资料就是：

- [Objective-C Automatic Reference Counting \(ARC\) — Clang 15.0.0git documentation \(llvm.org\)](https://clang.llvm.org/docs/AutomaticReferenceCounting.html#runtime-support)

对应的 `objc` 开头的系列函数是：

- `id objc_autorelease(id value);`
- `void objc_autoreleasePoolPop(void *pool);`
- `void *objc_autoreleasePoolPush(void);`
- `id objc_autoreleaseReturnValue(id value);`
- `void objc_copyWeak(id dest, id src);`
- `void objc_destroyWeak(id *object);`
- `id objc_initWeak(id *object, id value);`
- `id objc_loadWeak(id *object);`
- `id objc_loadWeakRetained(id *object);`
- `void objc_moveWeak(id dest, id src);`
- `void objc_release(id value);`
- `id objc_retain(id value);`
- `id objc_retainAutorelease(id value);`
- `id objc_retainAutoreleaseReturnValue(id value);`
- `id objc_retainAutoreleasedReturnValue(id value);`
- `id objc_retainBlock(id value);`
- `void objc_storeStrong(id *object, id value);`
- `id objc_storeWeak(id *object, id value);`
- `id objc_unsafeClaimAutoreleasedReturnValue(id value);`

## 1 About this document

### 1.1 Purpose

The first and primary purpose of this document is to serve as a complete technical specification of Automatic Reference Counting. Given a core Objective-C compiler and runtime, it should be possible to write a compiler and runtime which implements these new semantics.

The secondary purpose is to act as a rationale for why ARC was designed in this way. This should remain tightly focused on the technical design and should not stray into marketing speculation.

下面详细介绍部分函数：

## objc\_autoreleaseReturnValue

TODO:

- 【整理】iOS逆向心得：iOS函数objc\_autoreleaseReturnValue
- 【已解决】iOS逆向Xcode调试：\_objc\_autoreleaseReturnValue继续运行导致后续release相关代码运行不到

## objc\_alloc

objc\_alloc 是NSObject的类的分配空间，常和 objc\_init 搭配使用

TODO:

【整理】iOS底层函数：objc\_alloc

## objc\_getClass

iOS逆向的动态调试时，底层写hook代码，往往涉及到这个 objc\_getClass

- 底层写hook代码

举例：

```
Class NSErrorClass = objc_getClass("NSError");
Class NSDictionaryClass = objc_getClass("NSDictionary");

// 写代码解析传入的变量，是什么类型，然后决定取出对应属性，即request url
// NSURL* getHamErrReqUrl(NSError* curError){
NSURL* getHamErrReqUrl(id erroOrDict) {
    NSURL* curUrl = NULL;
    if (curError) {
        if (erroOrDict) {
            NSDictionary* curUserInfo = NULL;
            if ([erroOrDict isKindOfClass: NSErrorClass]) {
                curUserInfo = [erroOrDict userInfo];
            } else if ([erroOrDict isKindOfClass: NSDictionaryClass]) {
                curUserInfo = (NSDictionary*)erroOrDict;
            }
            if (curUserInfo) {
                id hamErrUrlReq = curUserInfo[@"HAMErrorURLRequest"];
                if (hamErrUrlReq != NULL) {
                    BOOL isUrlReq = [hamErrUrlReq isKindOfClass: NSMutableURLRequestClass];
                    if (isUrlReq) {
                        curUrl = [hamErrUrlReq URL];
                    }
                }
            }
        }
    }
    return curUrl;
}
```

TODO:

【无法解决】iOS越狱检测和反越狱检测：objc\_getClass

【整理】iOS运行时Runtime：objc\_getClass相关函数

## objc\_storeStrong

定义：

```
void objc_storeStrong(id object, id value);
```

说明：

Precondition: object is a valid pointer to a `__strong` object which is adequately aligned for a pointer. value is null or a pointer to a valid object.

Performs the complete sequence for assigning to a `__strong` object of non-block type [\*]. Equivalent to the following code:

内部实现逻辑：

```
void objc_storeStrong(id object, id value) {
    id oldValue = object;
    value = [value retain];
    object = value;
    [oldValue release];
}
```

- This does not imply that a `__strong` object of block type is an invalid argument to this function. Rather it implies that an `objc_retain` and not an `objc_retainBlock` operation will be emitted if the argument is a block.

相关理解：

- 在 Objective-c 中，对象的引用关系由引用修饰符来决定，如 `__strong`、`__weak`、`__autorelease` 等等，编译器会根据不同的修饰符生成不同逻辑的代码来管理内存。
  - 在 MRC 时代 `Retain` 修饰符将会使被引用的对象引用计数 +1
  - 在 ARC 中 `__strong` 修饰符作为其替代者

在正向开发写代码，在给 `__strong` 变量赋值时

```
obj = otherObj;
```

内部其实会调用对应的 runtime 的函数：

```
// 会变成如下函数调用
objc_storeStrong(obj, otherObj);
```

## objc\_enumerationMutation

iOS逆向期间，常会看到，伪代码中有个 `objc_enumerationMutation`，其实就是表示：代码循环而已

-» 反推：正向代码中，用了 `for`、`while` 等循环的逻辑。

TODO：

【已解决】iOS底层函数：`objc_enumerationMutation`

## objc\_retainBlock

`objc_retainBlock` 是和ObjC中的 `Block` 相关的函数。和ARC中的引用计数中相关。

TODO：

【已解决】iOS基础知识：`objc_retainBlock`

## objc\_loadWeakRetained

TODO:

【未解决】Xcode的lldb中objc\_loadWeakRetained传入的对象是什么类

## objc\_copyImageNames

TODO:

【无需解决】iOS越狱检测和反越狱检测: objc\_copyImageNames

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新: 2022-10-24 17:33:12

# objc\_msgSend

TODO:

- 【未解决】IDA中如何解析objc\_msgSend函数调用
- 【整理】iOS逆向和IDA使用心得：调用objc\_msgSend时传递给MLPlayerItemQOEErrorEvent的initWithError:fatal:absoluteTime:的参数不够
- 【整理】iOS逆向开发心得：如何判断ObjC的objc\_msgSend的参数个数
- 【已解决】Xcode的lldb调试iOS的ObjC或Swift时如何打印出objc\_msgSend第一个参数是什么类的实例
- 【整理】ObjC底层函数：objc\_msgSendSuper2、objc\_msgSendSuper
- 【未解决】iOS逆向：如何找到objc\_msgSendSuper2的第一个参数调用的是什么类
- 【基本解决】Xcode的lldb中动态调试objc\_msgSend第一个参数self是哪个类
- 
- 【未解决】研究YouTube逻辑：谁调用了\_dispatch\_call\_block\_and\_release以及\_pthread\_wqthread
- 【整理】iOS逆向调试心得：如何找到Block的\_pthread\_wqthread和\_dispatch\_call\_block\_and\_release的函数调用最初来源
- 【整理】iOS逆向心得：ObjC函数调用时参数顺序和汇编代码中寄存器传递的参数顺序不一致

iOS逆向中，肯定会涉及到底层的ObjC的Runtime中，最重要的一个函数：objc\_msgSend

- objc\_msgSend

- 是什么：ios 的 objc 中的 Runtime 中，算是最重要的一个底层实现函数了
- 定义

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

- 核心逻辑：Objc中所有的类的函数/属性的调用，（经过编译器编译后）底层其实都是转换成objc\_msgSend方法的调用

- 核心内容

```
+[classOrObj someMethod: inputValue];
-[classOrObj someMethod: inputValue];
```

- 被翻译成：

```
objc_msgSend(classOrObj, @selector(someMethod:), inputValue);
```

- 举例

```
+[NSString stringWithUTF8String: "Hello Crifan"];
```

- 被翻译成：

```
objc_msgSend(NSString, @selector(stringWithUTF8String:), "Hello Crifan");
```

# object系列函数

此处介绍iOS的ObjC的Runtime中，常见的，以 `object` 开头的一些系列函数。

## object\_getClassName

iOS逆向的动态调试，有时会用到：`object_getClassName`

比如：

- 用lldb调试时，想要查看第一个参数是什么类：

```
(lldb) reg r x0
x0 = 0x000000028195cccc0
(lldb) po 0x000000028195cccc0
10764012736

(lldb) po (char *)object_getClassName(0x000000028195cccc0)
"MLOneSieRequestContext"
```

和：

```
(lldb) reg r x0 x10
x0 = 0x000000010a43a000  (void *)0x000000010a509788: YTGLUILabel
x10 = 0x0000000109375578 @"partid"
(lldb) po 0x000000010a43a000
4467171328

(lldb) po (char *)0x000000010a43a000
"\x88\x97P\n\x00\x00\x00\x01"

(lldb) po object_getClassName(0x000000010a43a000)
0x00000001080535b5

(lldb) po (char *)object_getClassName(0x000000010a43a000)
"YTGLUILabel"
```

- MonkeyDev 的 `LLDBTools.m` 中也用到过：

```
NSString *choose(const char *classname){
    NSMutableString *result = [NSMutableString new];
    NSArray *results = choose_inner(classname);
    [result appendFormat @"Find %lu instance objects in memory!\n", (unsigned long)results.count];
    for (id item in results) {
        [result appendFormat @"<%s: 0x%llx>\n", object_getClassName(item), (long long)item];
    }
    return result;
}
```

TODO：

- 【已解决】iOS逆向心得：`object_getClassName`获取类名\_\_NSDictionaryM是什么意思

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新：2022-10-24 17:33:22

# 其他心得

## iOS Runtime Header

可以查询和搜索iOS运行时的头文件的网站：

<https://developer.limneos.net/>

## ObjC Runtime源码

- objc4 = Objc = ObjC Runtime
  - 下载
    - <https://opensource.apple.com/tarballs/objc4/>
  - 在线浏览
    - <https://opensource.apple.com/source/objc4/>
    - 常用版本
      - <https://opensource.apple.com/source/objc4/objc4-750/>
      - <https://opensource.apple.com/source/objc4/objc4-818.2/>

具体例子：

- objc\_retainAutoreleasedReturnValue + objc\_autoreleaseReturnValue
  - <https://opensource.apple.com/source/objc4/objc4-532.2/runtime/NSObject.mm>
- objc\_retainBlock
  - 相关源码：runtime/objc-arr.mm
    - 离线下载
      - <https://opensource.apple.com/tarballs/objc4/objc4-493.9.tar.gz>
    - 在线浏览
      - <https://opensource.apple.com/source/objc4/objc4-493.9/runtime/objc-arr.mm.auto.html>
- objc\_alloc
  - <https://opensource.apple.com/source/objc4/objc4-646/runtime/objc-internal.h>

```
OBJC_EXPORT id objc_alloc(Class cls)
__OSX_AVAILABLE_STARTING(__MAC_10_9, __IPHONE_7_0);

OBJC_EXPORT id objc_allocWithZone(Class cls)
__OSX_AVAILABLE_STARTING(__MAC_10_9, __IPHONE_7_0);
```

- objc\_loadWeakRetained
  - <https://opensource.apple.com/source/objc4/objc4-706/runtime/NSObject.mm.auto.html>

## 可调试的objc runtime代码

看到一个经过别人整理，是可以运行和调试的objc runtime的代码：

[RetVal/objc-runtime: A debuggable objc runtime \(github.com\)](https://github.com/RetVal/objc-runtime)

如果以后需要，可以去尝试去编译和运行和调试

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新：2023-07-10 10:44:56



## 附录

下面列出相关参考资料。

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新: 2022-03-17 20:39:28

## 参考资料

- [Objective-C Automatic Reference Counting \(ARC\) — Clang 15.0.0git documentation \(llvm.org\)](#)
- [理解 ARC 实现原理 \(xietao3.com\)](#)
- [Objective-C 小记 \(9\) \\_\\_strong - 简书 \(jianshu.com\)](#)
- [load 方法全程跟踪 - 知乎 \(zhihu.com\)](#)
- [RetVal/objc-runtime: A debuggable objc runtime \(github.com\)](#)
- [Understanding The Objective-C Runtime | iEasynote](#)
- [objc\\_msgSend | Apple Developer Documentation](#)
- [objc-runtime-new.m](#)
- [message.h](#)
- 

crifan.org, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook最后更新: 2023-07-10 10:35:13