

iOS开发 App Store研究 苹果相关 安卓相关 区块链 AI 业界动态 程序人生 平台任务

首页 > iOS开发

## iOS 底层解析weak的实现原理（包含weak对象的初始化，引用，释放的分析）

2017-03-28 06:56 编辑: suiling 分类: iOS开发 来源: 此生逍遥

26

Runtime 弱引用 weak表



### 原文

很少有人知道weak表其实是一个hash（哈希）表，Key是所指对象的地址，Value是weak指针的地址数组。更多人的只是知道weak是弱引用，所引用对象的计数器不会加一，并在引用对象被释放的时候自动被设置为nil。通常用于解决循环引用问题。但现在单知道这些已经不足以应对面试了，好多公司会问weak的原理。weak的原理是什么呢？下面就分析一下weak的工作原理（只是自己对这个问题好奇，学习过程中的笔记，希望对读者也有所帮助）。


### weak 实现原理的概括

Runtime维护了一个weak表，用于存储指向某个对象的所有weak指针。weak表其实是一个hash（哈希）表，Key是所指对象的地址，Value是weak指针的地址（这个地址的值是所指对象的地址）数组。


weak 的实现原理可以概括一下三步：

- 1、初始化时：runtime会调用objc\_initWeak函数，初始化一个新的weak指针指向对象的地址。
- 2、添加引用时：objc\_initWeak函数会调用 objc\_storeWeak() 函数， objc\_storeWeak() 的作用是更新指针指向，创建对应的弱引用表。
- 3、释放时，调用clearDeallocating函数。clearDeallocating函数首先根据对象地址获取所有weak指针地址的数组，然后遍历这个数组把其中的数据设为nil，最后把这个entry从weak表中删除，最后清理对象的记录。


### 热门资讯

- 


对不起，我被裁员了

点击量 2913
- 


Flutter仿微信项目实战，这才是期待中的跨平台

点击量 2766
- 


TikTok(抖音国际版)逆向，全球的小姐姐们，

点击量 2542
- 


2018 我的技术进阶之路

点击量 1984
- 


程序员怒了！阿里 Antd 圣诞彩蛋害我被

点击量 1878
- 


同样是iOS程序员，6K与30K的差距到底在

点击量 1755
- 


32岁程序员面试，Leader称高龄直接送

点击量 1726
- 

iOS WHGradientHelper（线

点击量 1589
- 

市值蒸发3900亿美元，苹果低头，iPhone

点击量 1483
- 

开发小知识

点击量 1401

### 综合评论

- 前天被出轨，今天被沦陷。哈哈哈哈哈
- Dcook 评论了 程序员被沦陷！国内程序员真的饱和了？ ...
- 谁知道意外和明天那个先到，坐着等着吧！
- 甘林梦 评论了 “久坐会增加早死风险”被证实 到底多久算久坐？ ...
- 对不起这个标题~
- Coke先生 评论了 中高级iOS大厂面试宝典，拿到offer率80%， ...
- 好
- wzq6556033 评论了 一个优质的项目应该具有什么特点...
- 做做马甲包的上架大神加我企鹅：1602381782
- 小猪不上树 评论了 iOS 最新审核被拒及解决方案 (包括2.1大礼包...

下面将开始详细介绍每一步：

**1、初始化时：**runtime会调用objc\_initWeak函数，objc\_initWeak函数会初始化一个新的weak指针指向对象的地址。

示例代码：

```
1 | {
2 |     NSObject *obj = [[NSObject alloc] init];
3 |     id __weak obj1 = obj;
4 | }
```

当我们初始化一个weak变量时，runtime会调用 NSObject.mm 中的objc\_initWeak函数。这个函数在Clang中的声明如下：

```
1 | id objc_initWeak(id *object, id value);
```

而对于 objc\_initWeak() 方法的实现

```
1 | id objc_initWeak(id *location, id newObj) {
2 |     // 查看对象实例是否有效
3 |     // 无效对象直接导致指针释放
4 |     if (!newObj) {
5 |         *location = nil;
6 |         return nil;
7 |     }
8 |     // 这里传递了三个 bool 数值
9 |     // 使用 template 进行常量参数传递是为了优化性能
10 |    return storeWeakfalse/*old*/, true/*new*/, true/*crash*/>
11 |    (location, (objc_object*)newObj);
12 | }
```

可以看出，这个函数仅仅是一个深层函数的调用入口，而一般的入口函数中，都会做一些简单的判断（例如 objc\_msgSend 中的缓存判断），这里判断了其指针指向的类对象是否有效，无效直接释放，不再往深层调用函数。否则，object将被注册为一个指向value的\_\_weak对象。而这事应该是objc\_storeWeak函数干的。

**注意：**objc\_initWeak函数有一个前提条件：就是object必须是一个没有被注册为\_\_weak对象的有效指针。而value则可以是null，或者指向一个有效的对象。

**2、添加引用时：**objc\_initWeak函数会调用 objc\_storeWeak() 函数，objc\_storeWeak() 的作用是更新指针指向，创建对应的弱引用表。

objc\_storeWeak的函数声明如下：

```
1 | id objc_storeWeak(id *location, id value);
```

objc\_storeWeak() 的具体实现如下：

```
1 | // HaveOld:      true - 变量有值
2 | //               false - 需要被及时清理，当前值可能为 nil
3 | // HaveNew:      true - 需要被分配的新值，当前值可能为 nil
4 | //               false - 不需要分配新值
5 | // CrashIfDeallocating: true - 说明 newObj 已经释放或者 newObj 不支持弱引
6 | //               false - 用 nil 替代存储
7 | template bool HaveOld, bool HaveNew, bool CrashIfDeallocating>
8 | static id storeWeak(id *location, objc_object *newObj) {
9 |     // 该过程用来更新弱引用指针的指向
10 |    // 初始化 previouslyInitializedClass 指针
11 |    Class previouslyInitializedClass = nil;
12 |    id oldObj;
13 |    // 声明两个 SideTable
14 |    // ① 新旧散列创建
15 |    SideTable *oldTable;
16 |    SideTable *newTable;
17 |    // 获得新值和旧值的锁存位置（用地址作为唯一标示）
18 |    // 通过地址来建立索引标志，防止桶重复
19 |    // 下面指向的操作会改变旧值
20 |    retry:
21 |        if (HaveOld) {
22 |            // 更改指针，获得以 oldObj 为索引所存储的值地址
23 |            oldObj = *location;
24 |            oldTable = &SideTables()[oldObj];
```

死猪不怕开水烫的我，狮子多了不怕咬的我，反正都是死的我怕个球  
Dcook 评论了 “久坐会增加早死风险”被证实 到底多久算久坐？ ...

吓

Dcook 评论了 程序员这段注释差点把领导气吐血，网友：双击溜溜溜...

iOS9 以上的系统失效 别发这种low文了 mmp

lky123456 评论了 ios 关于itunes安装ipa包的方法，一步解...

图呢

290014768 评论了 ios 关于itunes安装ipa包的方法，一步解...

下一个

qq240066285 评论了 中国程序员仅凭借一段劳动法则霸榜GitHub，每个...

## 相关帖子

收到账户被调查的消息

我奉劝大家，不要侵权，不要破坏市场

苹果审核，新账号第一次审核，已经9天了，还是没有反馈

最近Paypal不能用了

苹果电脑卡的问题

更新通过了，但是商店里一直看不到，什么原因？

假如你离开了现在这家公司，你还能潇洒地生活吗？

高德地图

没有¥299的企业账号，可以重签名吗

微博



CocoaChina

加关注

犯困的周三来篇纯干货：《一文掌握iOS开发中的全部web知识》http://t.cn/EJNK90Q 周三过去了，周五还会远吗~[挤眼]



3月27日 13:31

转发 | 评论

100天iOS数据结构与算法实战 Day0 1 [爱你] http://t.cn/EJXAAy1



07:00 15:00

转发 | 评论

```

25     } else {
26         oldTable = nil;
27     }
28     if (HaveNew) {
29         // 更改新值指针, 获得以 newObj 为索引所存储的值地址
30         newTable = &SideTables()[newObj];
31     } else {
32         newTable = nil;
33     }
34     // 加锁操作, 防止多线程中竞争冲突
35     SideTable::lockTwoHaveOld, HaveNew>(oldTable, newTable);
36     // 避免线程冲突重处理
37     // location 应该与 oldObj 保持一致, 如果不同, 说明当前的 location 已经处
38     if (HaveOld && *location != oldObj) {
39         SideTable::unlockTwoHaveOld, HaveNew>(oldTable, newTable);
40         goto retry;
41     }
42     // 防止弱引用间死锁
43     // 并且通过 +initialize 初始化构造器保证所有弱引用的 isa 非空指向
44     if (HaveNew && newObj) {
45         // 获得新对象的 isa 指针
46         Class cls = newObj->getIsa();
47         // 判断 isa 非空且已经初始化
48         if (cls != previouslyInitializedClass &&
49             !((objc_class *)cls)->isInitialized()) {
50             // 解锁
51             SideTable::unlockTwoHaveOld, HaveNew>(oldTable, newTable);
52             // 对其 isa 指针进行初始化
53             class_initialize(class_getNonMetaClass(cls, (id)newObj)
54                             // 如果该类已经完成执行 +initialize 方法是最理想情况
55                             // 如果该类 +initialize 在线程中
56                             // 例如 +initialize 正在调用 storeWeak 方法
57                             // 需要手动对其增加保护策略, 并设置 previouslyInitializedClass
58                             previouslyInitializedClass = cls;
59                             // 重新尝试
60                             goto retry;
61             )
62         }
63         // ② 清除旧值
64         if (HaveOld) {
65             weak_unregister_no_lock(&oldTable->weak_table, oldObj, locati
66         }
67         // ③ 分配新值
68         if (HaveNew) {
69             newObj = (objc_object *)weak_register_no_lock(&newTable->weak
70                                                         (id)newObj, loc
71                                                         CrashIfDeallocat
72             // 如果弱引用被释放 weak_register_no_lock 方法返回 nil
73             // 在引用计数表中设置若引用标记位
74             if (newObj && !newObj->isTaggedPointer()) {
75                 // 弱引用位初始化操作
76                 // 引用计数那张散列表的weak引用对象的引用计数中标识为weak引用
77                 newObj->setWeaklyReferenced_nolock();
78             }
79             // 之前不要设置 location 对象, 这里需要更改指针指向
80             *location = (id)newObj;
81         }
82         else {
83             // 没有新值, 则无需更改
84         }
85         SideTable::unlockTwoHaveOld, HaveNew>(oldTable, newTable);
86         return (id)newObj;
87     }

```

撇开源码中各种锁操作, 来看看这段代码都做了些什么。

## 1)、SideTable

SideTable 这个结构体, 我给他起名引用计数和弱引用依赖表, 因为它主要用于管理对象的引用计数和 weak 表。在 NSObject.mm 中声明其数据结构:

```

1 struct SideTable {
2     // 保证原子操作的自旋锁
3     spinlock_t slock;
4     // 引用计数的 hash 表
5     RefcountMap refcnts;
6     // weak 引用全局 hash 表
7     weak_table_t weak_table;
8 }

```

对于 slock 和 refcnts 两个成员不用多说，第一个是为了防止竞争选择的自旋锁，第二个是协助对象的 isa 指针的 extra\_rc 共同引用计数的变量（对于对象结果，在今后的文中提到）。这里主要看 weak 全局 hash 表的结构与作用。

## 2)、weak表

weak表是一个弱引用表，实现为一个weak\_table\_t结构体，存储了某个对象相关的所有的弱引用信息。其定义如下(具体定义在objc-weak.h中)：

```
1 struct weak_table_t {
2     // 保存了所有指向指定对象的 weak 指针
3     weak_entry_t *weak_entries;
4     // 存储空间
5     size_t num_entries;
6     // 参与判断引用计数辅助量
7     uintptr_t mask;
8     // hash key 最大偏移值
9     uintptr_t max_hash_displacement;
10 };
```

这是一个全局弱引用hash表。使用不定类型对象的地址作为 key，用 weak\_entry\_t 类型结构体对象作为 value。其中的 weak\_entries 成员，从字面意思上看，即为弱引用表入口。其实现也是这样的。

其中weak\_entry\_t是存储在弱引用表中的一个内部结构体，它负责维护和存储指向一个对象的所有弱引用hash表。其定义如下：

```
1 typedef objc_object ** weak_referrer_t;
2 struct weak_entry_t {
3     DisguisedPtr<objc_object> referent;
4     union {
5         struct {
6             weak_referrer_t *referrers;
7             uintptr_t out_of_line : 1;
8             uintptr_t num_refs : PTR_MINUS_1;
9             uintptr_t mask;
10            uintptr_t max_hash_displacement;
11        };
12        struct {
13            // out_of_line=0 is LSB of one of these (don't care which)
14            weak_referrer_t inline_referrers[WEAK_INLINE_COUNT];
15        };
16    };
17 };
```

在 weak\_entry\_t 的结构中，DisguisedPtr referent 是对泛型对象的指针做了一个封装，通过这个泛型类来解决内存泄漏的问题。从注释中写 out\_of\_line 成员为最低有效位，当其为0的时候，weak\_referrer\_t 成员将扩展为多行静态 hash table。其实其中的 weak\_referrer\_t 是二维 objc\_object 的别名，通过一个二维指针地址偏移，用下标作为 hash 的 key，做成了一个弱引用散列。

那么在有效位未生效的时候，out\_of\_line、num\_refs、mask、max\_hash\_displacement 有什么作用？以下是笔者自身的猜测：

out\_of\_line：最低有效位，也是标志位。当标志位 0 时，增加引用表指针纬度。

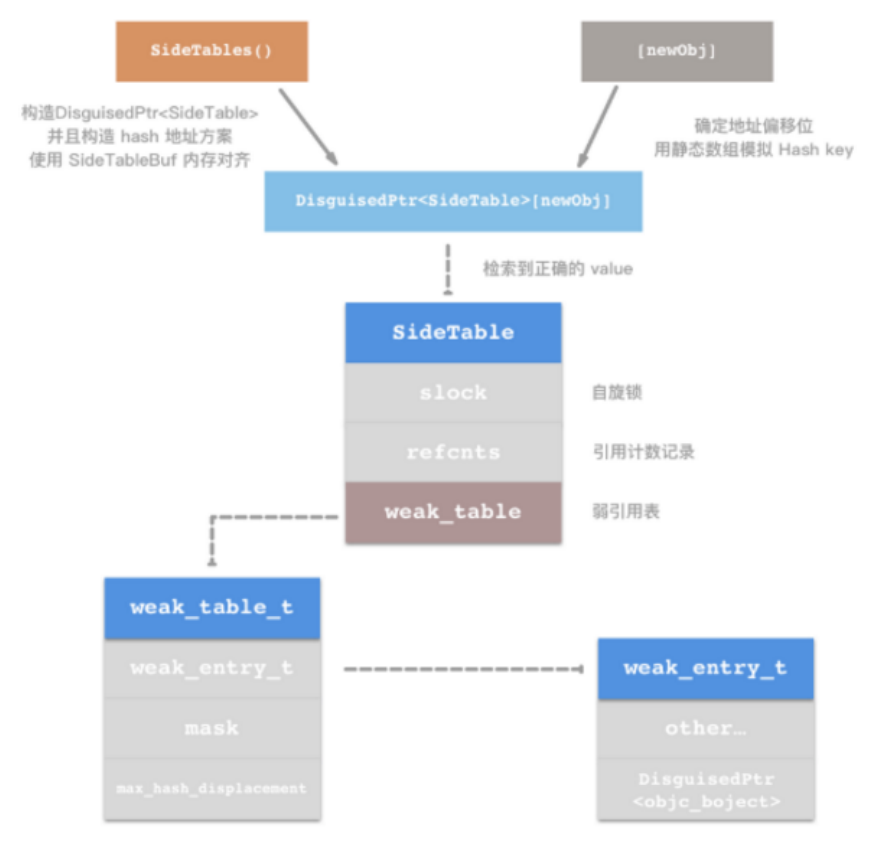
num\_refs：引用数值。这里记录弱引用表中引用有效数字，因为弱引用表使用的是静态 hash 结构，所以需要变量来记录数目。

mask：计数辅助量。

max\_hash\_displacement：hash 元素上限阈值。

其实 out\_of\_line 的值通常情况下是等于零的，所以弱引用表总是一个 objc\_objective 指针二维数组。一维 objc\_objective 指针可构成一张弱引用散列表，通过第三纬度实现了多张散列表，并且表数量为 WEAK\_INLINE\_COUNT。

总结一下 StripedMap[]：StripedMap 是一个模板类，在这个类中有一个 array 成员，用来存储 PaddedT 对象，并且其中对于 [] 符的重载定义中，会返回这个 PaddedT 的 value 成员，这个 value 就是我们传入的 T 泛型成员，也就是 SideTable 对象。在 array 的下标中，这里使用了 indexForPointer 方法通过位运算计算下标，实现了静态的 Hash Table。而在 weak\_table 中，其成员 weak\_entry 会将传入对象的地址加以封装起来，并且其中也有访问全局弱引用表的入口。



### 旧对象解除注册操作 weak\_unregister\_no\_lock

该方法主要作用是将旧对象在 weak\_table 中接触 weak 指针的对应绑定。根据函数名，称之为解除注册操作。从源码中，可以知道其功能就是从 weak\_table 中接触 weak 指针的绑定。而其中的遍历查询，就是针对于 weak\_entry 中的多张弱引用散列表。

### 新对象添加注册操作 weak\_register\_no\_lock

这一步与上一步相反，通过 weak\_register\_no\_lock 函数把新的对象进行注册操作，完成与对应的弱引用表进行绑定操作。

### 初始化弱引用对象流程一览

弱引用的初始化，从上文的分析中可以看出，主要的操作部分就在弱引用表的取键、查询散列、创建弱引用表等操作，可以总结出如下的流程图：



这个图中省略了很多情况的判断，但是当声明一个 weak 会调用上图中的这些方法。当然，storeWeak 方法不仅仅用在 weak 的声明中，在 class 内部的操作中也会常常通过该方法来对 weak 对象进行操作。

3、释放时，调用clearDeallocating函数。clearDeallocating函数首先根据对象地址获取所有weak指针地址的数组，然后遍历这个数组把其中的数据设为nil，最后把这个entry从weak表中删除，最后清理对象的记录。

当weak引用指向的对象被释放时，又是如何去处理weak指针的呢？当释放对象时，其基本流程如下：

- 1、调用objc\_release
- 2、因为对象的引用计数为0，所以执行dealloc
- 3、在dealloc中，调用了\_objc\_rootDealloc函数
- 4、在\_objc\_rootDealloc中，调用了object\_dispose函数
- 5、调用objc\_destructInstance
- 6、最后调用objc\_clear\_deallocating

重点看对象被释放时调用的objc\_clear\_deallocating函数。该函数实现如下：

```

1 void objc_clear_deallocating(id obj)
2 {
3     assert(obj);
4     assert(!UseGC);
5     if (obj->isTaggedPointer()) return;
6     obj->clearDeallocating();
7 }
  
```

也就是调用了clearDeallocating，继续追踪可以发现，它最终是使用了迭代器来取weak表的value，然后调用weak\_clear\_no\_lock,然后查找对应的value，将该weak指针置空，weak\_clear\_no\_lock函数的实



现如下：

```

1  /**
2  * Called by dealloc; nils out all weak pointers that point to the
3  * provided object so that they can no longer be used.
4  *
5  * @param weak_table
6  * @param referent The object being deallocated.
7  */
8  void weak_clear_no_lock(weak_table_t *weak_table, id referent_id)
9  {
10     objc_object *referent = (objc_object *)referent_id;
11     weak_entry_t *entry = weak_entry_for_referent(weak_table, referent);
12     if (entry == nil) {
13         /// XXX shouldn't happen, but does with mismatched CF/objc
14         /// printf("XXX no entry for clear deallocating %p\n", referent);
15         return;
16     }
17     // zero out references
18     weak_referrer_t *referrers;
19     size_t count;
20
21     if (entry->out_of_line) {
22         referrers = entry->referrers;
23         count = TABLE_SIZE(entry);
24     }
25     else {
26         referrers = entry->inline_referrers;
27         count = WEAK_INLINE_COUNT;
28     }
29
30     for (size_t i = 0; i < count; ++i) {
31         objc_object **referrer = referrers[i];
32         if (referrer) {
33             if (*referrer == referent) {
34                 *referrer = nil;
35             }
36             else if (*referrer) {
37                 _objc_inform("__weak variable at %p holds %p instead
38                 \"This is probably incorrect use of \"
39                 \"objc_storeWeak() and objc_loadWeak(). \"
40                 \"Break on objc_weak_error to debug.\n\",
41                 referrer, (void*)*referrer, (void*)referent);
42                 objc_weak_error();
43             }
44         }
45     }
46     weak_entry_remove(weak_table, entry);
47 }

```

objc\_clear\_deallocating该函数的动作如下：

- 1、从weak表中获取废弃对象的地址为键值的记录
- 2、将包含在记录中的所有附有 weak修饰符变量的地址，赋值为nil
- 3、将weak表中该记录删除
- 4、从引用计数表中删除废弃对象的地址为键值的记录

看了objc-weak.mm的源码就明白了：其实Weak表是一个hash（哈希）表，然后里面的key是指向对象的地址，Value是Weak指针的地址的数组。

补充：.m和.mm的区别

.m：源代码文件，这个典型的源代码文件扩展名，可以包含OC和C代码。

.mm：源代码文件，带有这种扩展名的源代码文件，除了可以包含OC和C代码之外，还可以包含C++代码。仅在你的OC代码中确实需要使用C++类或者特性的时候才用这种扩展名。

参考资料：

- [weak 弱引用的实现方式](#)
- [weak的生命周期：具体实现方法](#)