鹏鹏南柯 原创文章
21 原创 | 15万+ 周排名 | 18万+ 总排名 | 47+ 粉丝 | 等级
600 积分 | 3 积分 | 4 获赞 | 0 评论 | 33 收藏
私信 关注

搜博主文章

热门文章
关于-charset的几种编码方式 10379
use_frameworks的作用，静态库与动态库，libd 7977
如何利用当前线程是主线程 6674
判断小数点后几位：整数的取模运算（%）与小数的取模运算（fmod） 3532
ARC的实现原理 2366

最新文章
iOS中的 uncaught exception 主要有哪些
C++文件打包成framework如何给人使用
Cocoapods的相关知识点

2019年 6篇 2019年 23篇

原创 鹏鹏南柯 2018-08-28 09:43:50 2371 收藏
分类专栏：iOS移动开发

iOS移动开发 专栏收录该内容 0 订阅 25 篇文章 订阅专栏

1、ARC的实现原理

> 我们都知道，ARC是编译器特性，程序在编译的时候，编译器帮我们在合适的地方插入retain、release等代码以管理对象的引用计数，从而达

但是只有编译器是无法单独完成这一工作的，还需要OC运行时库的配合协助，因此ARC的实现工具主要包括：

1. LLVM编译器（clang 3.0以上）
2. OC运行时库4.0.390以上

1、__strong修饰符

使用__strong修饰变量的程序究竟是如何运行的

```
{
    id __strong object = [NSObject alloc] init];
}
```

转换后的模拟源代码为

```
{
    /*编译器的模拟代码*/
    id object = objc_msgSend(NSObject,@selector(alloc));
    objc_msgSend(object,@selector(init));
    objc_release(object);
}
```

对象变量生成时，alloc和init分别调用了两个方法调用，对象变量作用域结束时调用objc_release方法释放对象变量，虽然ARC情况下不能使用release方法，但是由此可见编译器编译时在合适的地方插入了release。

在使用alloc、new、copy、mutableCopy以外的方法生成对象变量方法时会有什么不同

```
{
    __strong object = [NSMutableArray array];
}
```

调用array的类方法转换后如下

```
{
    /*编译器的模拟代码*/
    id object = objc_msgSend(NSMutableArray,@selector(array));
    objc_retainAutoreleasedReturnValue(object);
    objc_release(object);
}
```

中间的objc_retainAutoreleasedReturnValue(object)函数与之前的不同，它主要是起何作用的呢？

```
objc_retainAutoreleasedReturnValue(object)函数的作用：最优化程序运行
```

自己持有（retain）对象的函数，但它持有的应为自己注册在autoreleasepool中对象的方法或动态的返回值。

objc_retainAutoreleasedReturnValue函数与objc_autoreleaseReturnValue是成对出现的，现在看看NSMutableArray类的array类方法的编译器实现

```
+ (id)array {
    return [[NSMutableArray alloc] init];
}
```

转换后的源代码

```
+ (id) array {
    /*编译器的模拟代码*/
    id obj = objc_msgSend(NSMutableArray,@selector(alloc));
    objc_msgSend(obj,@selector(init));
    return objc_autoreleaseReturnValue(obj);
}
```

通过objc_autoreleaseReturnValue函数将对象注册在自动释放池autoreleasepool中并返回。但是与objc_autorelease函数不同的是，objc_autoreleaseReturnValue函数一般不仅用于注册对象到autoreleasepool中去。

```
objc_autoreleaseReturnValue与objc_retainAutoreleasedReturnValue的配合使用，可以不将对象注册到autoreleasepool中而直接传递，达到最优化
```

objc_autoreleaseReturnValue函数会检查使用该函数的方法或者函数的调用方的执行命令列表，如果调用方在调用该函数或方法后，紧接着调用了objc_retainAutoreleasedReturnValue函数，那么将不再将对象注册到autoreleasepool中去，而直接将对象传递给调用方。

相比于objc_retain函数来说objc_retainAutoreleasedReturnValue函数在返回一个即使没有注册到autoreleasepool中的对象，也能正确的获取对象。

3、__weak修饰符

weak解析可参考下面这篇链接：
weak属性修饰原解析

这里主要讲解与__weak与__unsafe_unretained的区别

```
{
    id __weak object = [NSObject alloc] init];
}
```

转换后的模拟源代码为

```
{
    /* 编译器的模拟代码 */
    id object;
    id tmp = objc_msgSend(NSObject, @selector(alloc));
    objc_msgSend(tmp, @selector(init));
    objc_initWeak(&object, tmp);
    objc_release(tmp);
    objc_destoryWeak(&object);
}
```

自己生成并且持有的对象通过objc_initWeak函数赋值给__weak修饰符的变量，但是编译器判断并没有对其进行持有，因此该对象通过objc_release函数被释放和废弃，
随后通过objc_destoryWeak将引用废弃对象的附有__weak 修饰符的变量置为nil

```
{
    id __unsafe_unretained object = [NSObject alloc] init];
}
```

转换后的模拟源代码为

```
{
    /*编译器的模拟代码*/
    id object = objc_msgSend(NSObject, @selector(alloc));
    objc_msgSend(object, @selector(init));
    objc_release(tmp);
}
```

可见通过__unsafe_unretained修饰的变量引用了对象但是并不持有对象，对象在释放和废弃后，并没有调用被__unsafe_unretained修饰的变量的objc_destoryWeak函数，因此该对象的悬垂指针被赋值给变量object，导致引用变量object打发生非法。

```
带有__weak修饰符的变量，其引用的对象已经注册到autoreleasepool中
```

如果不是直接赋值，而是通过使用__weak修饰符来引用变量时

```
{
    id __weak object = obj;
    NSLog(@"%@",object);
}
```

转换后的模拟源代码为

```
/*编译器的模拟代码*/
{
    id object;
    objc_initWeak(&object, obj);
    id temp = objc_loadWeakRetained(&object);
    objc_autorelease(temp);
    NSLog(@"%@", temp);
    objc_destoryWeak(&object);
}
```

明显增加了objc_loadWeakRetained与objc_autorelease函数调用，他们的主要作用是：

1. objc_loadWeakRetained函数取出__weak修饰符变量引用的对象并且retain
2. objc_autorelease函数将引用的对象注册到autoreleasepool中

因此，使用__weak修饰符引用的对象都通过注册到autoreleasepool中，在@autoreleasepool块结束之前都可以放心使用，大量使用__weak修饰符的话，会因注册到autoreleasepool中的对象也大量地增加，所以在使用__weak修饰符引用的变量时，最好先暂时用__strong修饰符的变量进行引用后再使用，

```
29年前使用__weak修饰符的情况
```

- 重写了retain/release的类，例如NSMachPort类
- 当allowsWeakReference/retainWeakReference实例方法返回NO时

4、__autoreleasing修饰符

```
__autoreleasing修饰符变量引用的对象，相当于在MRC情况下调用对象的autorelease方法
```

对于alloc、new、copy、mutableCopy如何实现的

```
@autoreleasepool{
    id __autoreleasing object = [NSObject alloc] init];
}
```

转换后的模拟源代码为

```
{
    /*编译器的模拟代码*/
    id pool = objc_autoreleasePoolPush();
    id object = objc_msgSend(NSObject, @selector(alloc));
    objc_msgSend(object, @selector(init));
    // 调用autorelease方法
    objc_autorelease(object);
    id pool = objc_autoreleasePoolPop();
}
```

NSMutableArray类种的array方法如何实现autorelease功能

```
@autoreleasepool{
    id __autoreleasing object = [NSMutableArray array];
}
```

转化后的模拟源代码为：

```
{
    /*编译器的模拟代码*/
    id pool = objc_autoreleasePoolPush();
    id object = objc_msgSend(NSMutableArray, @selector(array));
    objc_retainAutoreleasedReturnValue(object);
    // 调用autorelease方法
    objc_autorelease(object);
    id pool = objc_autoreleasePoolPop();
}
```

除了持有对象的方法从alloc变成了objc_retainAutoreleasedReturnValue函数，但是注册到autoreleasepool的方法没有变化，都是调用了objc_autorelease函数。

5、引用计数

```
extern uintptr_t _objc_rootRetainCount();可获取ARC情况下对象的引用计数
```

仅用于调试，对于已释放的对象或者不正确的对象地址，有时也返回“1”，此外在多线程情况下使用对象的引用计数数值，因为竞态条件的问题，取得的数值不一定完全可信。

ARC cpu和系统利 06-16
想了发现有Fairy cpu并考平易，对于新入式开发非常有用
ARC4算法(加密) 每天多一分钟学习，生命就多一份精彩 2268

#include "StdAfx.h" #include "ARc4EnDe.h" CArc4EnDe::CArc4EnDe(void) { } CArc4EnDe::~CArc4EnDe(void) { } void CArc4EnDe::Fz_arcInit...
登录后可以写您作者群更高位置
发表评论 评论

相关推荐
ARC的实现原理 -- 概要_gcs的博客 7-10
ARC的实现原理 -- 概要 自动引用计数(Automatic Reference Count 简称 ARC)，是苹果在 WWDC 2011 年大会上提出的用于内存管理的技术。 引用计...
理解ARC 实现原理 -- 详细自述_gcs的博客_arc原理 7-1
ARC 是 iOS 中管理引用计数的技术 简称 iOS 实现过程自动回收，具体实现的原理是由编译器进行管理的，同时运行时库协助编译器辅助完成，主要涉及...
理解 ARC 实现原理 weixin_33963504的博客 457
ARC 是 iOS 中管理引用计数的技术，简称 iOS 实现过程自动回收，具体实现的原理是由编译器进行管理的，同时运行时库协助编译器辅助完成，主要涉...
ARC到底帮我们做了哪些工作? DCSmart-林牛 5051
从iOS开始，苹果开始引入自动引用计数(Automatic Reference Counting,ARC)了，所以就变得更为简单了，ARC几乎把所有内存管理事宜交给编译器来决定...
简单介绍 ARC 以及 ARC 实现的原理_dianzanni3748的博客 7-2
简单介绍 ARC 以及 ARC 实现的原理，考虑点 ARC 是苹果在 WWDC 2011 提出来的技术,因此很多新人行的同学可能对此技术陌生不熟悉。但是,虽然...
【iOS】理解 ARC 实现原理(经典)_Harrovey Blog 5-15
理解 ARC 实现原理 iOS字典和数组底层实现原理 罗宁笔 4330 ### NSMutableArray的底层原理 _size是堆区指针，size是堆区的...
ARC介绍 weixin_34016740的博客 85
ARC也是想修讲的一部分内容，很多iOS的程序员到现在也没有完全转到ARC，当然有有各的理由，这里只是转载了一篇介绍性的文字，简单介绍了ARC...
ARC基本原理 explorer_freedom的博客 695
基本原理 ARC是Automatic Reference Counting自动引用计数器的简称，ARC是ios5.0引入的新特性，完全消除手动管理内存的管理，编译器会自动在...
iOS底层原理 浅谈ARC 以及 ARC 实现的原理。_cchoudingwan... 7-11
ARC 的工作原理大致是这样:当我们编译器运行的时候,编译器会分析源码中每个对象的生命周期,然后基于这些对象的生命周期,来添加相应的引用计数操作...
如何理解iOS的ARC和ARC的实现原理_Harrovey Blog 7-11
iOSARC完全指南 10-30 iOS基础篇 浅谈ARC以及ARC实现的原理。 0_猴贡的博客 143 自动的引用计数(Automatic Reference Count 简称ARC)是苹果...
iOS面试题：简述ARC 以及 ARC 实现的原理。 0_猴贡的博客 144
自动的引用计数（Automatic Reference Count 简称 ARC），是苹果在 WWDC 2011 年大会上提出的用于内存管理的技术。 引用计数（Reference Coun...
iOS 底层解析weak的实现原理 3198847的博客 169
参考地址：------------ http://www.cocoachina.com/ios/20170326/18962.html 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...
ARC的原理详解 weixin_34216190的博客 212
1、ARC的本质 ARC本质是NSAutoreleasePool的直接应用。 @autoreleaseat[return UIApplicationMainargc, argv, nil, NSStringFromClass(AppDelegat...
oc语言 ARC机制的简单学习 lipengz123的专栏 3593
先用Xcode新建一个命令行工程
ARC实现 夜流脑袋的博客 208
1、__strongid原理 (id __strong obj) { [NSObject alloc] init]; } //编译器模拟代码/ id obj = objc_msgSend(NSObject,@selector(alloc); objc_msgSend(o...
【黑马程序员】ARC概念及原理 m_25269224的博客 1700
---------- Java语言、Android语言、.Net语言、期待与您交流！ ---------- 1、指针分类 （1）强指针：默认的情况下，所有的指针都是强指针...
浅析ARC 1240
浅析ARC
iOS面试专题(三)：底层原理相关问题 kyb2828995430的博客 306
分享目录iOS面试专题(三)： 底层原理相关问题iOS面试专题(三)：底层原理相关问题
阿里、字节：一套高效的iOS面试题（修正版） iOS_开发 620
作者 | monkey来源 | 码上work （codework168） 简述笔者最近收集地理了一些iOS相关的问题，其中大部分都是大厂面试或者面试其他人用到的，能会...

©2020 CSDN 皮肤主题: 精致技术 设计师:CSDN官方博客 返回首页
关于我们 招聘纳才 广告服务 开发助手 400-660-0108 kefu@csdn.net 在线客服 工作时间 8:30-22:00
公安备案号11010502030143 京ICP备19034160号 京网文（2020）1039-165号 经营性网站备案信息
北京互联网违法和不良信息举报中心 网络110报警服务 中国互联网举报中心 家长监护 Chrome浏览下载
网络谣言专区 数字证书管理服务平台 权威认证 北京市公安局网安总队在线备案 京公网安备11010502030143 ICP证书 ICP证书查询 ICP证书信息公示