delphi090902的专栏 Just do what i what

个人资料

安子就是安子

访问： 171118次
积分： 2414
等级： BLOG 5
排名： 第8711名

原创： 67篇 转载： 24篇
译文： 3篇 评论： 24条

Objective-C 内存管理之dealloc方法中变量释放处理

标签： variables crash application xcode object debugging

2011-10-31 16:57 4068人阅读 评论(0) 收藏 举报

分类： objective-c

(一).关于nil

http://cocoadevcentral.com/d/learn_objectivec/

Calling Methods on Nil

In Objective-C, the *nil* object is the functional equivalent to the *NULL*pointer in many other languages.

The difference is that you can call methods on nil without crashing or throwing an exception.

在Objective-C中，nil对象与许多其他语言中的NULL指针有同样的功能。

区别在于，你可以向一个nil对象发送任何信息，即让一个nil的对象调用任何方法，而不会导致程序的崩溃或者抛出异常。

这是一个有用的基础知识

另外在cocoachina （http://www.cocoachina.com/bbs/read.php?tid-70288.html ）论坛有人指出

对象被release后 系统会将该块内存标记为可用（可重新分配） nil就是起到个重置指针的作用 对内存的释放无意义

防止出现调用错误。

(二).结论：

不同的环境下使用不同的代码处理是中庸之道，也就是

如果你在开发和调试，测试阶段

使用可以直接暴露问题代码的dealloc处理方式：

在dealloc中即仅仅release你的变量。

如果你把程序发布给用户使用，使用先release，后赋值nil的方式

这样，不会暴露crash给用户

(三)..以下是参考：

首先，这是一篇非常好的博客，详细说明了关于dealloc中的几种处理方式的论证

http://iphonedevelopment.blogspot.com/2010/09/dealloc.html

英文好的看原文，不要被我的翻译误解。

另外：

stackoverflow上几个有价值的参考，上面讨论了开发者的在这方面的经验:

http://stackoverflow.com/questions/4124049/dealloc-use-release-or-set-to-nil-for-properties

http://stackoverflow.com/questions/192721/why-shouldnt-i-use-objective-c-2-0-accessors-in-init-dealloc

(四)..现在，正式对这篇博客翻译：http://iphonedevelopment.blogspot.com/2010/09/dealloc.html

声明：如果有错误翻译或者错字等请指出，但是本人仅是翻译供学习探讨所用，任何问题与纠纷居于本人无关，谢谢合作!

Dealloc

Last week there was a bit of a Twitter in-fight in the iOS community over the "right" way to release your instance variables in dealloc. I think Rob actually started it, to be honest, but I probably shouldn't be bringing that up.

上一周，在Twitter iOS 社区上有一场关于任何正确在dealloc方法中正确释放你的变量的激烈讨论(辩论大战！)，我认为是由Rob引发的，但说实话，我其实不应该透露这一点(好笑，作者不想八卦，却仍然给了Rob的连接~).

Basically, several developers were claiming that there's never a reason to set an instance variable to nil in dealloc, while others were arguing that you should always do so.

基本上，几个开发者都声明到，没有任何理由允许在dealloc方法中把一个实例变量设值为nil，同时其他极为争辩到你应该这么做，即在dealloc中，对实例变量赋值为nil.

To me, there didn't seem to be a clear and compelling winner between the two approaches. I've used both in my career. However, since we're in the process of trying to decide which approach to use in the next edition of Beginning iPhone Development, I reached out to Apple's Developer Tools Evangelist, Michael Jurewitz, to see if there was an official or recommended approach to handling instance variables in dealloc.

对我来说，似乎在这两种方式中，没有出现一个清晰地完全的胜利者。在我的开发过程中，这两种方式我都用过。但是，既然我们在尝试将这两个方法取其一而用到下一版本的 Beginning iPhone Development中，我向苹果的开发者 Michael Jurewitz 求助,来探索在dealloc中如何处理实例变量这件事情上，是否存在一个官方的或者推荐的方法。

Other than the fact that you should never, ever use mutators in dealloc (or init, for that matter), Apple does not have an official recommendation on the subject.

除了在dealloc或者init方法中，你最好不要使用设值方法（eg:[self.xxxx message];），苹果没有提供过关于dealloc中释放实例变量的官方推荐方法。

However, Michael and Matt Drance of Bookhouse Software and a former Evangelist himself, had discussed this issue extensively last week. They kindly shared their conclusions with me and said it was okay for me to turn it into a blog post. So, here it is. Hopefully, I've captured everything correctly.

然而，Michael and Matt Drance of Bookhouse Software 和一个前 Evangelist 他自己（Evangelist 本意是传播福音的人，我猜是那些宣讲苹果官方技术的人员，就是普及苹果开发技术的一些专家吧）关于此问题在上周进行了一场较广泛的讨论。他们非常有好的与我分享了他们的结论结果并且同意我在博客上登载出来，所以，结果是这样的，希望我已经正确地搜集到每个方面了。

The Two Major Approachs

Just to make sure we're all on the same page, let's look at the two approaches that made up the two different sides of the argument last week.

两种主要的方法

首先来确认我们达成共识，让我们先看看上周两个不同支持者的两种方法。

Just Release

仅仅release

The more traditional approach is to simply release your instance variables and leave them pointing to the released (and potentially deallocated) object, like so:

最传统的方法是仅仅release掉你的实例变量并且任其只想一个已经release的对象上，就像这样:

```
[cpp] view plain  copy  print  ?
- (void)dealloc
{
    [Sneezy release];
    [Sleepy release];
    [Dopey release];
    [Doc release];
    [Happy release];
    [Bashful release];
    [Grumpy release];
    [super dealloc];
}
```

In this approach, each of the pointers will be pointing to a potentialy invalid object for a very short period of time — until the method returns — at which point the instance variable will disappear along with its owning object. In a single-threaded application (the pointer would have to be accessed by something triggered by code in either this object's implementation of dealloc or in the dealloc of one of its superclasses, there's very little chance that the instance variables will be used before they go away , which is probably what has led to the proclamations made by several that there's "no value in setting instance variables to nil" in dealloc.

在这个方法上，每一个指针都会在非常短的时间上的指向一块可能非法的对象上，知道方法执行完毕，在那时，实例变量会随着它的拥有者一同消失。在一个单线程的应用中（指针会不得不被或者是对象的自己实现dealloc又或者在它父类的dealloc中一些启动的东西访问到，一个实例变量在它小事之前会被使用到是一个小概率事件。但如果发生了，就正中下怀了，也就是那些宣告说需要在dealloc方法中赋值给实例变量nil值观点的人所顾虑的那样。

In a multi-threaded environment, however, there is a very real possibility that the pointer will be accessed between the time that it is deallocated and the time that its object is done being deallocated. Generally speaking, this is only going to happen if you've got a bug elsewhere in your code, but let's face it, you may very well. Anyone who codes on the assumption that all of their code is perfect is begging for a smackdown, and Xcode's just biding its time waiting for the opportunity.

然而，在一个多线程的环境里，一个指针在本身被dealloc以后与它的拥有者对象被dealloc之前被访问到，却是一个很有可能发生的事情的。一般而言，如果你在别处代码中遇到bug，这里成为了问题的源头，但是让我们面对它，你会变得更好。任何建立在自己的代码是完美的这个假设上写代码的人都是在掩耳盗铃，Xcode仅仅是在伺机而动，等待机会报错给你。

Release and nil

In the last few years, another approach to `dealloc` has become more common. In this approach, you release your instance variable and then immediately set them to `nil` before releasing the next instance variable. It's common to actually put the `release` and the assignment to `nil` on the same line separated by a comma rather than on consecutive lines separated by semicolons, though that's purely stylistic and has no affect on the way the code is compiled. Here's what our previous `dealloc` method might look like using this approach:

在过去的几年，另一个在dealloc中处理的方法变得越来越普及，在这个方法中，你release你的实例变量并且在releasing下一个实例变量前立即把它赋值为nil.比较普遍的做法是，将release语句与赋值nil语句用一个都好隔开放到同一行而不是用分号隔开为多行，尽管这样做仅仅是表层代码的长相上，在编译时是无任何差别的。下面是我们之前的dealloc用这种方法后的样子：

```
[cpp] view plain copy print ?
- (void)dealloc
{
    [sneezy release], sneezy = nil;
    [sleepy release], sleepy = nil;
    [dopey release], dopey = nil;
    [doc release], doc = nil;
    [happy release], happy = nil;
    [bashful release], bashful = nil;
    [grumpy release], grumpy = nil;
    [super dealloc];
}
```

In this case, if some piece of code accesses a pointer between the time that `dealloc` begins and the object is actually deallocated, it will almost certainly fail gracefully because sending messages to `nil` is perfectly okay in Objective-C. However, you're doing a tiny bit of extra work by assigning `nil` to a bunch of pointers that are going to go away momentarily, and you're creating a little bit of extra typing for yourself in every class.

在这种情况下如果有代码访问一个开始被dealloc但是还没被完全dealloc间的指针时，这样的操作是无效失败的，因为在Objective-C中，想一个nil值的对象发送消息是没有问题，不会引发错误的。但是，如果你赋值给一些会立即消失的对象nil值，你却做了一些额外的工作，在每一个你的类中都会制造这些额外的输入开支。

The Showdown 最后一战

So, here's the real truth of the matter: The vast majority of the time, it's not going to make any noticeable difference whatsoever. If you're not accessing instance variables that have been released, there's simply not going to be any difference in the behavior between the two approaches. If you are, however, then the question is: what do you want to happen when your code does that bad thing?

那么，接下来是最重要的地方。绝大多数的时候，没有什么明显的区别。如果你不去访问那些被释放掉的实例变量，在这两种方法间没有任何的功能方面结果的区别。但如果你访问了，那么问题是，你想你的代码如何针对这种情况做出反应?

In the first approach, your application will usually crash with an `EXC_BAD_ACCESS`, though you could also end up with any manner of odd behavior (which we call a "heisenbug") if the released object is deallocated and its memory is then reused for another object owned by your application. In those cases, you may get a `selector not recognized` exception when the message is sent to the deallocated object, or you may simply get unexpected behavior from the same method being called on the wrong object.

第一种方法下，你的程序通常下会崩溃，警告给你EXC_BAD_ACCESS,虽然你还可能最后得到一些诡异的结果（通常我们都称它为heisenbug，这种bug是诡异的出现，并且通常不好重现，以致难以修补）像是一个released对象被dealloced并且它的内存之后被你的冲虚中另一个对象利用到。在这种情况下，你可能得到一个selector not recognized异常，当消息被发送给一个被delloced完的对象，或者在一个错误的对象执行你调用的方法后，你可能仅仅得到一个异常结果。

In the other approach, your application will quietly send a message to `nil` and go about its merry

way, generally without a crash or any other immediately identifiable problem.
在另一个方法下，你的程序会给一个nil值的对象发送消息并且就这么不了了之了~什么都没有发生，没有程序崩溃（no crash！！），没有其他任何实时可分辨的问题出现。
The former approach is actually good when you're developing, debugging, and doing unit testing, because it makes it easier to find your problematic code. On the other hand, that approach is really, really bad in a shipping application because you really don't want to crash on your users if you can avoid it.

前一种方法在你正在开发，调试或者做单元测试时会更有优势。因为它会让你很容易找到问题代码。另一方面，这个方法会非常非常的糟糕，在你发布你的程序后，因为你不想你的用户在使用时遇到程序crash并且这是你可以避免的crash！。

The latter approach, conversely, can hide bugs during development, but handles those bugs more gracefully when they happen, and you're far less likely to have your application go up in a big ball of fire in front of your users.

后一种方法，相反地会在开发过程中掩盖住你的bugs，但是在bugs发生的时候更温和的处理掉了，但同时你会让或多或少可能让你的程序被火球包裹，然后呈现给你的用户。

The Winner? 胜利者?
There really isn't a clear cut winner, which is probably why Apple doesn't have an official recommendation or stance. During their discussion, Matt and Michael came up with a "best of both worlds" solution, but it requires a fair bit of extra code over either of the common approaches.
其实并没有一个明显的胜利者，或许这就是为什么苹果没有给出一个官方的推荐或者例子的原因。在他们讨论期间，Matt and Michael 得出一个权衡之计，既两种情况都会使程序更好更稳固的解决方案，但是这需要一些额外的代码，在这两种通常的方法基础上。
If you want your application to crash when a released pointer is accessed during development and debugging, the solution is to use the traditional approach in your debug configuration. If you want your application to degrade gracefully for your users, the solution is to use the newer approach in your release and ad hoc configurations.
在你开发和调试期间，当一个指针被访问时如果你想要你的程序直接crash，解决方案就是用最传统的方法在你的调试环境。如果你想你的程序友好的呈现给用户，解决方法是用新的方法，在你的release并且加上额外配置。
One somewhat pedantic implementation of this approach would be this:
一种比较死板的实现这个理论的方法会像是这样：

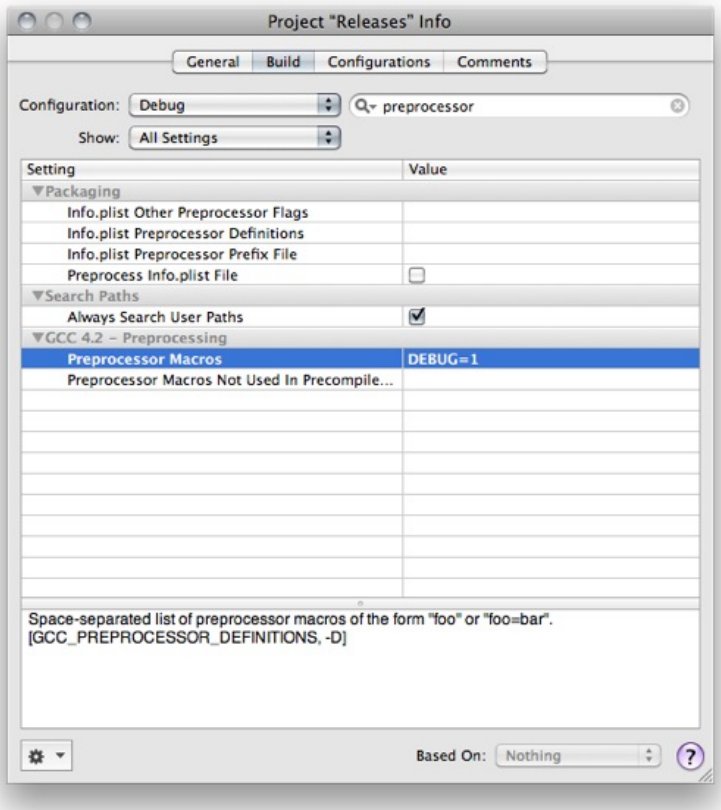
```
- (void)dealloc
{
#if DEBUG
    [Sneezy release];
    [Sleepy release];
    [Dopey release];
    [Doc release];
    [Happy release];
    [Bashful release];
    [Grumpy release];

    [super dealloc];
#else
    [sneezy release], sneezy = nil;
    [sleepy release], sleepy = nil;
    [dopey release], dopey = nil;
    [doc release], doc = nil;
    [happy release], happy = nil;
    [bashful release], bashful = nil;
    [grumpy release], grumpy = nil;
```

```
    [super dealloc];
#endif
}
```

That code assumes that your debug configuration has a precompiler definition of `DEBUG`, which you usually have to add to your Xcode project - most Xcode project templates do not provide it for you. There are several ways you can add it, but I typically just use the *Preprocessor Macros* setting in the project's Build

这个代码假设你的调试配置中有一个预编译宏定义，你必须将它添加到你的Xcode 工程中，大多数Xcode工程模板不会提供给你这个。有多种方式添加，但是代表性的，我只使用在工程Build下的预编译宏设值

配置：



Although the code above does, indeed, give us the best of both worlds - a crash during development and debugging and graceful degrading for customers - it at least doubles the amount of code we have to write in every class. We can do better than that, though. How about a little macro magic? If we add the following macro to our project's .pch file:
尽管上面的代码给了我们权衡的答案，也就是在开发和调试时程序会crash，但是在发布给用户时会平稳的处理掉。可是它至少在我们每一个类中都多使用照原来两倍的代码。我们可以做的更好，用一些宏怎么样？如果我们在.pch文件中加入下面的宏变量：

```
[cpp] view plain  copy  print  ?
#if DEBUG
#define MCRelease(x) [x release]
#else
#define MCRelease(x) [x release], x = nil
#endif
```

We can then use that macro in `dealloc`, and our best-of-both-worlds code becomes much shorter and more readable:
我们可以在dealloc中使用它们，并且我们的“权衡方法”变得更短，更加易读:

```
- (void)dealloc
{

    MCRelease(sneezy);
    MCRelease(sleepy);
    MCRelease(dopey);
    MCRelease(doc);
    MCRelease(happy);
    MCRelease(bashful);
    MCRelease(grumpy);

    [super dealloc];
}
```

Once you've got the macro in your project, this option is actually no more work or typing than either of the other `dealloc` methods.
一旦你在你的工程中写了宏，这种选择不会比之前的两种dealloc方法花费额外的工作。
But, you know what? If you want to keep doing it the way you've always done it, it's really fine, regardless of which way you do it. If you're consistent in your use and are aware of the tradeoffs, there's really no compelling reason to use one over the other outside of personal preference.
但是你知道么？如果你想保持你之前在dealloc处理对象的方式，其实也是不错的。无论你现则哪种方式，如果你对你使用并且意识到其中利弊，真的什么强制的理由让一个人改变他的个人选择。
So, in other words, it's kind of a silly thing for us all to argue over, especially when there's already politics, religions, and sports to fill that role.

那么，也就是说，我们没有必要争论下去了，需要争论的东西太多了，像是政策，种族，体育那些...

The Garbage Collection Angle
There's one last point I want to address. I've heard a few times from different people that setting an instance variable to `nil` in`dealloc` acts as a hint to the garbage collector when you're using the allowed-not-required GC option (when the required option is being used, `dealloc` isn't even called, `finalize` is). If this were true, forward compatibility would be another possible argument for preferring the newer approach to `dealloc` over the traditional approach.
还有最后一点我想要指出。好几次，我从不同的人听到过，在dealloc方法中赋值给实例变量nil值会起到给垃圾回收器一个钩子的作用，当你用allowed-not-required GC 选项（当required-option被使用，dealloc不会被调用，finalize会）。如果这是真的，向前兼容会变成新的争论，针对在dealloc里新的方法与传统的方法之间。
While it is true that in Java and some other languages with garbage collection, `null`ing out a pointer that you're done with helps the garbage collector know that you're done with that object, so it's not altogether unlikely that Objective-C's garbage collector does the same thing, however any benefit to `nil`'ing out instance variables once we get garbage collection in iOS would be marginal at best. I haven't been able to find anything authoritative that supports this claim, but even if it's 100% true, it seems likely that when the owning object is deallocated a few instructions later, the garbage collector will realize that the deallocated object is done with anything it was using.

在java和一些其他的自带垃圾回收的机制的语言中，给一个指针赋值null会帮助你让垃圾回收器知道

你已经用完这个对象了，它们不等同于Objective-C的垃圾回收器做同样的操作，但是使用nil赋值给实例变量的益处，一旦是iOS的垃圾回收机制它都变得相当的边缘化了。我没能找到任何权威性的论据来支持这个声明，但是即使它是百分百真的，似乎当一个被拥有的对象被指令dealloc执行后，垃圾回收器会意识到那个被dealloc的对象已经做完所有该做的事情。

If there is a benefit in this scenario, which seems unlikely, it seems like it would be such a tiny difference that it wouldn't even make sense to factor it into your decision. That being said, I have no firm evidence one way or the other on this issue and would welcome being enlightened.

在这种情境下有没有好处，貌似没有...似乎它都不能成为你做决定使用那种方法的一个参考因素。也就是说，我没有有力的证据，在这个问题上欢迎大家来讨论互相启发。

上一篇 Objective-c 数据类型之间的转换(NSNumber NSDate...)

下一篇 iPhone代码碎片:处理objective- c 里html特殊字符显示问题的一个函数