25个增强iOS应用程序性能的提示和技巧 — 初级篇

- 作者 BeyondVincent

- 6 四月, 2013
- 6条评论

本文由破船译自：raywenderlich
转载请注明出处：BeyondVincent的博客

在开发iOS应用程序时，让程序具有良好的性能是非常关键的。这也是用户所期望的，如果你的程序运行迟钝或缓慢，会招致用户的差评。

然而由于iOS设备的局限性，有时候要想获得良好的性能，是很困难的。在开发过程中，有许多事项需要记住，并且关于性能影响很容易就忘记。

这就是为什么我要写这篇文章！本文收集了25个关于可以提升程序性能的提示和技巧。

目录

我把性能优化技巧分为3个不同的等级：初级、中级和高级：

初级

在开发过程中，下面这些初级技巧需要时刻注意：

初级性能提升

本部分内容介绍几本的程序性能提升技巧。其实所有级别的开发者都能从中获益。

1) 使用ARC进行内存管理

ARC是在iOS 5中发布的，它解决了最常见的内存泄露问题——也是开发者最容易健忘的。

ARC的全称是“Automatic Reference Counting”——自动引用计数，它会自动的在代码中做retain/release工作，开发者不用再手动处理。

下面是创建一个View通用的一些代码块：

```
1. UIView *view = [[UIView alloc] init];
2. // ...
3. [self.view addSubview:view];
4. [view release];
```

在上面代码结束的地方很容易会忘记调用release。不过当使用ARC时，ARC会在后台自动的帮你调用release。

ARC除了能避免内存泄露外，还有助于程序性能的提升：当程序中的对象不再需要的时候，ARC会自动销毁对象。所以，你应该在工程中使用ARC。

下面是一些学习ARC很棒的一些资源：

> 值得注意的是，ARC并不能避免所有的内存泄露。使用ARC之后，工程中可能还会有内存泄露，不过引起这些内存泄露的主要原因是：block，retain循环，对CoreFoundation对象（通常是C结构）管理不善，以及真的是代码没写好。

这里有一篇文章是介绍哪些问题是ARC不能解决的 — 以及如何处理这些问题。

2) 在适当的情况下使用reuseIdentifier

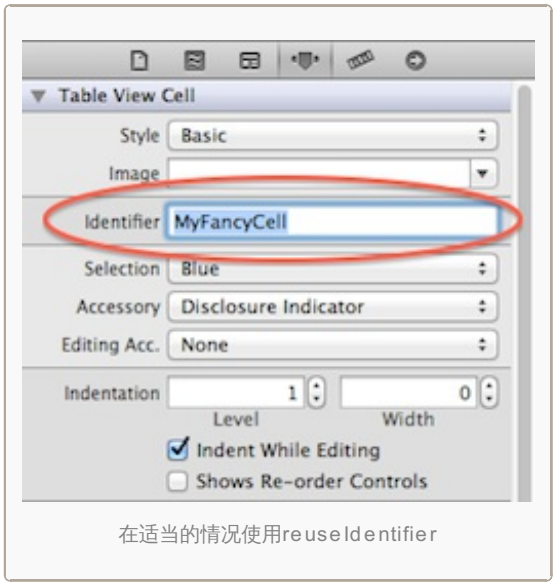
在iOS程序开发中一个普遍性的错误就是没有正确的为UITableViewCells、UICollectionViewCells和UITableViewHeaderFooterViews设置reuseIdentifier。

为了获得最佳性能，当在tableView:cellForRowAtIndexPath:方法中返回cell时，table view的数据源一般会重用UITableViewCell对象。table view维护着UITableViewCell对象的一个队列或者列表，这些数据源已经被标记为重用了。

如果没有使用reuseIdentifier会发生什么？

如果你在程序中没有使用reuseIdentifier，table view每次显示一个row时，都会配置一个全新的cell。这其实是一个非常消耗资源的操作，并且会影响程序中table view滚动的效率。

自iOS 6以来，你可能还希望header和footer views，以及UICollectionView的cell和supplementary views。



在适当的情况使用reuseIdentifier

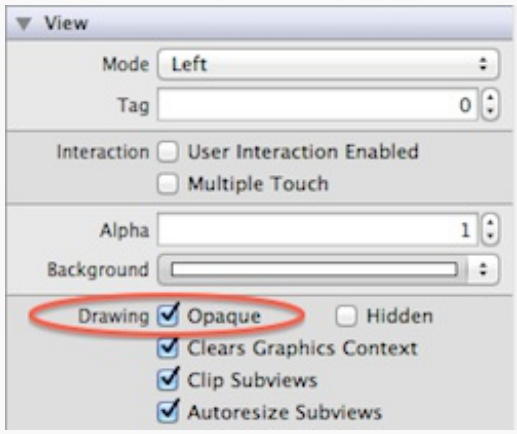
为了使用reuseIdentifiers，在table view请求一个新的cell时，在数据源中调用下面的方法：

```
1. static NSString *CellIdentifier = @"Cell";
2. UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier
   forIndexPath:indexPath];
```

如果table view维护的UITableViewCell队列或列表中有可用的cell，则从队列从移除一个已经存在的cell，如果没有的话，就从之前注册的nib文件或类中创建一个新的cell。如果没有可以重用的cell，并且没有注册nib文件或类，tableview的dequeueReusableCellWithIdentifier:方法会返回一个nil。

3) 尽可能将View设置为不透明（Opaque）

如果view是不透明的，那么应该将其opaque属性设置为YES。



尽量将view设置为Opaque

为什么要这样做呢？这样设置可以让系统以最优的方式来绘制view。opaque属性可以在Interface Builder或代码中设置。

苹果的官方文档对opaque属性有如下解释：

This property provides a hint to the drawing system as to how it should treat the view. If set to YES, the drawing system treats the view as fully opaque, which allows the drawing system to optimize some drawing operations and improve performance. If set to NO, the drawing system composites the view normally with other content. The default value of this property is YES.

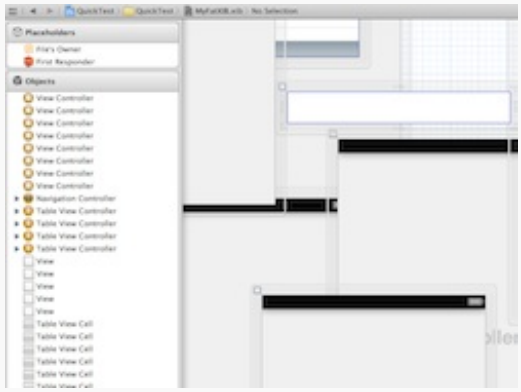
（opaque属性提示绘制系统如何处理view。如果opaque设置为YES，绘图系统会将view看为完全不透明，这样绘图系统就可以优化一些绘制操作以提升性能。如果设置为NO，那么绘图系统结合其它内容来处理view。默认情况下，这个属性是YES。）

如果屏幕是静止的，那么这个opaque属性的设置与否不是一个大问题。但是，如果view是嵌入到scroll view中的，或者是复杂动画的一部分，不将设置这个属性的话肯定会影响程序的性能！

可以通过模拟器的Debug\Color Blended Layers选项来查看哪些view没有设置为不透明。为了程序的性能，尽可能的将view设置为不透明！

4) 避免臃肿的XIBs

在iOS 5中开始使用Storyboards，并且将替代XIBs。不过在有些情况下XIBs仍然有用。如果你的程序需要运行在装有iOS 5之前版本的设备上，或者要自定义可重用的view，那么是避免不了要使用XIBs的。


避免臃肿的XIBs

如果必须要使用XIBs的话，尽量让XIBs文件简单。并且每个view controller对于一个XIB文件，如果可以的话，把一个view controller的view不同的层次单独分到一个XIBs文件中。

注意：当把一个XIB文件加载到内存时，XIB文件中的所有内容都将被加载到内存中，包括图片。如果有一个view还不立即使用的话，就会造成内存的浪费。而这在storyboard中是不会发生的，因为storyboard还在需要的时候才实例化一个view controller。

当加载XIB时，所有涉及到的图片都将被缓存，并且如果是开发的程序是针对OS X的话，声音文件也会被加载。苹果的官方文档这样说：

When you load a nib file that contains references to image or sound resources, the nib-loading code reads the actual image or sound file into memory and and caches it. In OS X, image and sound resources are stored in named caches so that you can access them later if needed. In iOS, only image resources are stored in named caches. To access images, you use the imageNamed: method of NSImage or UIImage, depending on your platform.

（当加载一个nib文件时，也会将nib文件涉及到的图片或声音资源加载到内存中，nib-loading代码会将实际的图片或声音文件读取到内存中，并一直缓存着。在OS X中，图片和声音资源都存储在命名缓存中，这样之后如果需要的话，可以对其进行访问。在iOS中，只有图片资源被存储到命名缓存中。要访问图片的话，使用NSImage或UIImage（根据不同的系统）的imageNamed:方法即可。）

显然，在使用storyboard时也会发生类似的缓存操作；不过我没有找到相关内容的任何资料。如果你知道的话，

可以告诉我哦！

想要学习storyboard的更多知识吗？可以看看Matthijs Hollemans写的iOS 5中：初级Storyboard Part 1和Part2。

5) 不要阻塞主线程

永远都不要在主线程做繁重的任务。因为UIKit的任务都在主线程中进行，例如绘制、触摸管理和输入响应。

在主线程做所有任务的风险是：如果你的代码阻塞了主线程，那么程序将出现反应迟钝。这回招致用户在App Store上对程序的差评！

在执行I/O操作中，大多数情况下都会祖塞主线程，这些操作需要从读写外部资源，例如磁盘或者网络。

关于网络操作可以使用NSURLConnection的如下方法，以异步的方式来执行：

不要阻塞主线程

```
1. + (void)sendAsynchronousRequest:(NSURLRequest *)request queue:(NSOperationQueue *)queue completionHandler:(void (^)(NSURLResponse*, NSData*, NSError*))handler
```

或者使用第三方框架，例如AFNetworking。

如果你需要做一些其它类型开销很大的操作（例如执行一个时间密集型的计算或者对磁盘进行读写），那么就使用GCD（Grand Central Dispatch），或NSOperations 和 NSOperationQueues。

下面的代码是使用GCD的一个模板：

```
1. dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{
2.     // switch to a background thread and perform your expensive operation
3.
4.     dispatch_async(dispatch_get_main_queue(), ^{
5.         // switch back to the main thread to update your UI
6.
7.     });
8. });
```

如上代码，为什么在第一个**dispatch_async**里面还嵌套了一个dispatch_async呢？这是因为关于UIKit相关的代码需要在主线程里面执行。

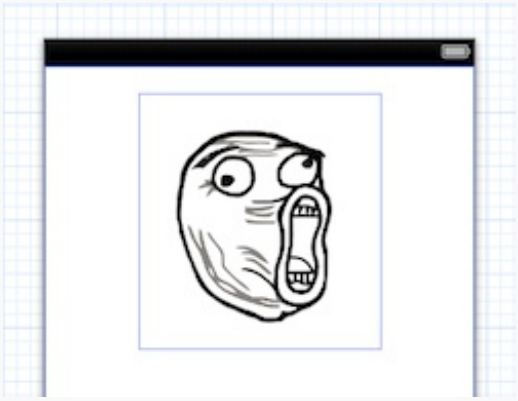
对**NSOperation**和**GCD**感到好奇吗？可以看看Ray Wenderlich中的教程：iOS中多线程和GCD—初级，以及Soheil Azarpour的

如何使用NSOperations和NSOperationQueues教程。

6) 让图片的大小跟UIImageView一样

如果需要将程序bundle中的图片显示到UIImageView中，请确保图片和UIImageView的大小是一样的。因为图片的缩放非常耗费资源，特别是将UIImageView嵌入到UIScrollView中。

如果是从远程服务中下载图片，有时候你控制不了图片的尺寸，或者在下载之前无法在服务器上进行图片的缩放。这种情况，当图片下载完之后，你可以手动进行图片的缩放——做好是在后台线程中！——然后再在UIImageView中使用缩放过的图片。



确保图片和UIImageView大小一致

7) 选择正确的集合

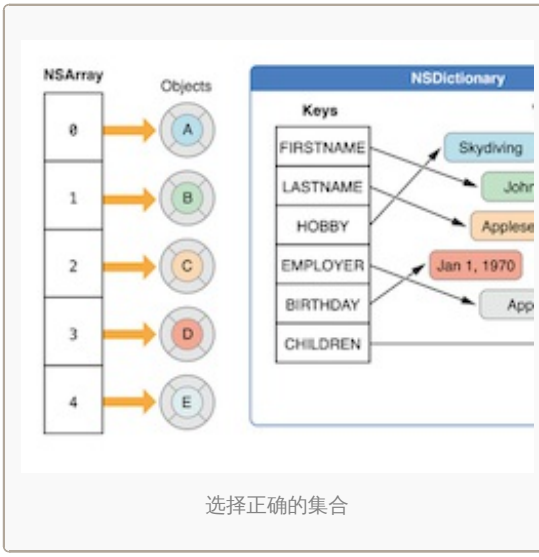
学习使用最适合的类或对象是编写高效代码的基础。特别是在处理集合数据时，尤为重要。

苹果的官网上有一篇文章：

集合编程主题(Collections Programming Topics)——详细的介绍了在集合数据中可以使用的类，以及什么情况下使用哪个类。在使用集合时，每个开发者都应该阅读一下这个文档。

太长，不想阅读(TLDR)？下面是常见集合类型的一个简介：

- 数组：是一个值按顺序排列的一个列表。根据索引可以快速查找，不过根据值进行查找就比较慢，另外插入和删除也比较慢。
- 字典: 存储键/值对。根据键可以快速查找。
- **Sets**: 是一个值无序排列的列表，根据值可以快速查找，另外插入和删除也比较快。



选择正确的集合

8) 使用GZIP压缩

越来越多的程序依赖于外部数据，这些数据一般来自远程服务器或者其它的外部APIs。有时候你需要开发一个程序来下载一些数据，这些数据可以是XML，JSON，HTML或者其它一些文本格式。

问题是在移动设备上的网络是不确定的。用户的设备可能在EDGE网络一分钟，然后接着又在3G网络中。不管在什么情况下，都不要让用户等待。

有一个可以优化的选择：使用GZIP对网络传输中的数据进行压缩，这样可以减小文件的大小，并加快下载的速度。压缩对于文本数据特别有用，因为文本具有很高的压缩比。

使用GZIP压缩

iOS中，如果使用NSURLConnection，那么默认情况下已经支持GZIP压缩了，并且基于NSURLConnection的框架页支持GZIP压缩，如AFNetworking。甚至有些云服务提供商已经提供发送经压缩过的响应内容，例如 Google App Engine。

这里有一篇关于GZIP压缩很好的文章，介绍了如何在Apache活IIS服务器中开启支持GZIP压缩。

25个增强iOS应用程序性能的提示和技巧 — 中级篇
25个增强iOS应用程序性能的提示和技巧 — 高级篇

4

- 支持
- 反对