25个增强iOS应用程序性能的提示和技巧 — 中级篇

- 作者 BeyondVincent
- 11 四月, 2013
- 暂无评论

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：

中级

在性能优化时，当你碰到一些复杂的问题，应该注意和使用如下技巧：

9. 重用和延迟加载View
10. 缓存、缓存、缓存
11. 考虑绘制
12. 处理内存警告
13. 重用花销很大的对象
14. 使用Sprite Sheets
15. 避免重新处理数据
16. 选择正确的数据格式
17. 设置适当的背景图片
18. 降低Web内容的影响
19. 设置阴影路径
20. 优化TableView
21. 选择正确的数据存储方式

中级性能提升

现在，在进行代码优化时，你已经能够完成一些初级性能优化了。但是下面还有另外一些优化方案，虽然可能不太明显（取决于程序的架构和相关代码），但是，如果能够正确的利用好这些方案，那么它们对性能的优化将非常明显！

9) 重用和延迟加载View

程序界面中包含更多的view，意味着界面在显示的时候，需要进行更多的绘制任务；也就意味着需要消耗更多的CPU和内存资源。特别是在一个UIScrollView里面加入了许多view。

这种情况的管理技巧可以参考UITableView和UICollectionView的行为：不要一次性创建所有的subview，而是在需要的时候在创建view，并且当view使用完毕时候将它们添加到重用队列中。

这样就可以仅在UIScrollView滚动的时候才配置view，以此可以避免分配创建view的带来的成本——这可能是非常耗资源的。

现在有这样的一个问题：在程序中需要显示的view在什么时机创建（比如说，当用户点击某个按钮，需要显示某个view）。这里有两种可选方法：

1. 在屏幕第一次加载以及隐藏的时候，创建view；然后在需要的时候，再把view显示出来。
2. 直到需要显示view的时候，才创建并显示view。

每种方法都有各自的优点和确定。

使用第一种方法，需要消耗更多的内容，因为创建出来的view一直占据着内存，直到view被release掉。不过，使用这种方法，当用户点击按钮时，程序会很快的显示出view，因为只需要修改一下view的可见性即可。

而使用第二种方法则产生相反的效果；当需要的时候猜创建view，这会消耗更少的内存；不过，当用户点击按钮的时候，不会立即显示出view。

10) 缓存、缓存、缓存

在开发程序时，一个重要的规则就是“缓存重要的内容”——这些内容一般不会改变，并且访问的频率比较高。

可以缓存写什么内容呢？比如远程服务器的响应内容，图片，甚至是计算结果，比如UITableView的行高。

NSURLConnection根据HTTP头的处理过程，已经把一些资源缓存到磁盘和内存中了。你甚至可以手动创建一个NSURLRequest ，让其只加载缓存的值。

下面的代码片段一般用在为图片创建一个NSURLRequest：

```
1. + (NSMutableURLRequest *)imageRequestWithURL:(NSURL *)url {
2.     NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:url];
3.
4.     request.cachePolicy = NSURLRequestReturnCacheDataElseLoad; // this will make sure the request always returns the cached image
5.     request.HTTPShouldHandleCookies = NO;
6.     request.HTTPShouldUsePipelining = YES;
7.     [request addValue:@"image/*" forHTTPHeaderField:@"Accept"];
8.
9.     return request;
10. }
```

注意：你可以使用NSURLConnection抓取一个URL请求，但是同样可以使用AFNetworking来抓取，这种方法不用修改所有网络相关的代码——这是一个技巧！:]

如果你要直到更多关于HTTP 缓存, NSURLCache, NSURLConnection 以及相关的内容, 那么看一下NSHipster中的the NSURLCache entry。

如果你需要缓存的内容没涉及到HTTP请求，那么使用NSCache。

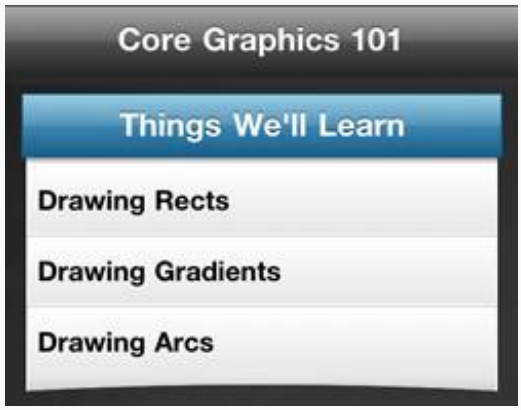
NSCache的外观和行为与NSDictionary类似, 但是，当系统需要回收内存时，NSCache会自动的里面存储的内容。Mattt Thompson 在NSHipster上写了一篇关于NSCache非常不错的文章。

如果还想知道关于HTTP缓存更多的内容，那么建议阅读一下Google的这篇文章：

best-practices document on HTTP caching。

11) 考虑绘制

在iOS中制作漂亮的按钮有多种方法。可以使用全尺寸图片，可缩放图片，或者使用CALayer, CoreGraphics， 甚至是OpenGL来手动测量和绘制按钮。



考虑绘制

当然，这些方法的复杂程度也不同，并且性能也有所区别。这里有一篇相关文章值得阅读一下：关于iOS中图形的性能。其中Andy Matuschak（曾经是苹果的UIKit小组的组员）对这篇文章的评论中，对于不同的方法及其性能权衡有非常好的一个见解。

简单来说，使用预渲染图片技术是最快的，因为iOS中不用等到在屏幕上显示的时候才创建图形和对形状进行绘制（图片已经创建好了!）。这样带来的问题是需要把所有的图片都放到程序bundle中，从而增加了程序的大小。因此使用可伸缩图片在这里将排上用场了：可以移除“浪费”空间的图片——iOS可以重复利用。并且针对不同的元素（例如按钮）不需要创建不同的图片。

不过，使用图片的话会失去代码对图片的控制能力，进而针对不同的程序，就需要重复的生成每一个需要的图片，并反复的放到每个程序中。这个处理过程一般会比较慢。另外一点就是如果你需要一个动画，或者许多图片都要进行轻微的调整（比如多个颜色的覆盖），那么需要在程序中加入许多图片，进而增加了程序bundle的大小。

总的来说，你需要考虑一下什么才是最重要的：绘制性能还是程序大小。一般来说都重要，所以在同一个工程中，应该两种都应考虑。

12) 处理内存警告

当系统内存偏低时，iOS会通知所有在运行的程序。苹果的官方文档中介绍了如何处理低内存警告：

If your app receives this warning, it must free up as much memory as possible. The best way to do this is to remove strong references to caches, image objects, and other data objects that can be recreated later. 如果程序收到了低内存警告，在程序中必须尽量释放内存。最佳方法就是移除强引用的涉及到的缓存，图片对象，以及其它可以在之后使用时还可以重新创建的数据对象。

UIKit中提供了如下几种方法来接收低内存（low-memory）警告：

- 实现app delegate中的applicationDidReceiveMemoryWarning: 方法。
- 在UIViewController子类中重写(Override)didReceiveMemoryWarning方法。
- 在通知中心里面注册UIApplicationDidReceiveMemoryWarningNotificatio通知。

在收到以上任意的警告时，需要立即释放任何不需要的内存。

例如，UIViewController的默认情况是清除掉当前不可见的view；在UIViewController的子类中，可以清除一些额

外的数据。程序中不没有显示在当前屏幕中的图片也可以release掉。

当收到低内存警告时，尽量释放内存是非常重要的。否则，运行中的程序有可能会被系统杀掉。

不过，在清除内存时要注意一下：确保被清除的对象之后还可以被创建出来。另外，在开发程序的时候，请使用iOS模拟器中的模拟内存警告功能对程序进行测试！

13) 重用花销很大的对象

有些对象的初始化非常慢——比如NSDateFormatter和NSCalendar。不过有时候可以避免使用这些对象，例如在解析JSON/XML中的日期时。

当使用这些对象时，为了避免性能上的瓶颈，可以尝试尽量重用这些对象——在类中添加一个属性或者创建一个静态变量。

注意，如果使用静态变量的话，对象会在程序运行的时候一直存在，就像单例一样。

下面的代码演示创建一个延迟加载的日期格式属性。第一次调用属性的时候，会创建一个新的日期格式。之后再调用的话，会返回已经创建好的实例对象：

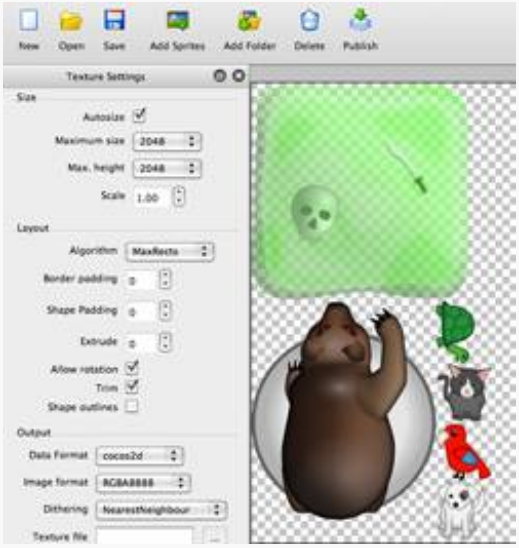
```
// in your .h or inside a class extension
@property (nonatomic, strong) NSDateFormatter *formatter;

// inside the implementation (.m)
// When you need, just use self.formatter
- (NSDateFormatter *)formatter {
    if (! _formatter) {
        _formatter = [[NSDateFormatter alloc] init];
        _formatter.dateFormat = @"EEE MMM dd HH:mm:ss Z yyyy"; // twitter date format
    }
    return _formatter;
}
```

另外，还需要记住的是在设置NSDateFormatter的日期格式时，同样跟创建新的一个NSDateFormatter实例对象时一样慢！因此，在程序中如果需要频繁的处理日期格式，那么对NSDateFormatter进行重用是非常好的。

14) 使用Sprite Sheets

你是一个游戏开发者吗？是的话那么sprite sheets是最佳选择之一。使用Sprite sheets跟常用的绘制方法比起来，绘制更快，并且消耗更少的内存。



使用sprite sheets

下面是两个非常不错的sprite sheets教程：

1. 如何在Cocos2D中使用动画和Sprite Sheets
2. 如何在Cocos2D中使用纹理包（Texture Packer）和像素格式来创建并优化Sprite Sheets

第二个教程详细的介绍了像素格式——在游戏中可以衡量性能的影响。

如果你还不熟悉sprite sheets，那么可以看看这里的介绍：SpriteSheets – 视频, Part 1and Part 2. 这两个视频的作者是Andreas Löw, 他是纹理包(Texture Packer)的创建者, 纹理包是创建sprite sheets的重要工具。

除了使用sprite sheets外，这里还介绍了一些用于游戏开发中的技巧，例如，如果你有很多sprite（比如射击类游戏中），那么可以重用sprite，而不用每次都创建sprite。

15) 避免重新处理数据

许多程序都需要从远程服务器中获取数据，以满足程序的需求。这些数据一般是JSON或XML格式。在请求和接收数据时，使用相同的数据结构非常重要。

为什么呢？在内存中把数据转换为适合程序的数据格式是需要付出额外代价的。

例如，如果你需要在table view中显示一些数据，那么请求和接收的数据格式最好是数组格式的，这样可以避免一些中间操作——将数据转换为适合程序使用的数据结构。

类似的，如果程序是根据键来访问具体的值，那么最好请求和接收一个键/值对字典。

在第一时间获得的数据就是所需要格式的，可以避免将数据转换为适合程序的数据格式带来的额外代价。

16) 选择正确的数据格式

将数据从程序传到网络服务器中有多种方法，其中使用的数据格式基本都是JSON和XML。你需要做的就是在程序中选择正确的数据格式。

选择正确的数据格式

JSON的解析速度非常快，并且要比XML小得多，也就意味着只需要传输更少数据。并且在iOS5之后，已经有内置的JSON反序列化API了，所以使用JSON是很容易的。

不过XML也有它自己的优势：如果使用SAX方法来解析XML，那么可以边读XML边解析，并不用等到全部的XML获取到了才开始解析，这与JSON是不同的。当处理大量数据时，这种方法可以提升性能并减少内存的消耗。

17) 设置适当的背景图片

在iOS编码中，跟别的许多东西类似，这里也有两种方法来给view设置一个背景图片：

1. 可以使用UIColor的colorWithPatternImge方法来创建一个颜色，并将这个颜色设置为view的背景颜色。
2. 可以给view添加一个UIImageView子视图。

如果你有一个全尺寸的背景图片，那么应该使用UIImageView，因为UIColor的colorWithPatternImge方法是用来创建小图片的——该图片会被重复使用。此时使用UIImageView会节省很多内存。

```
1. // You could also achieve the same result in Interface Builder
2. UIImageView *backgroundView = [[UIImageView alloc] initWithImage:[UIImage
   imageNamed:@"background"]];
3. [self.view addSubview:backgroundView];
```

不过，如果你计划用小图片当做背景，那么应该使用UIColor的colorWithPatternImge方法。这种情况下绘制速度会很快，并且不会消耗大量的内存。

```
1. self.view.backgroundColor = [UIColor colorWithPatternImage:[UIImage
   imageNamed:@"background"]];
```

18) 降低Web内容的影响

UIWebView非常有用。用它可以很容易的显示web内容，甚至可以构建UIKit空间难以显示的内容。

不过，你可以能已经注意到程序中使用的UIWebView组建没有苹果的Safari程序快。这是因为JIT编译限制了WebKit的Nitro引擎的使用。

因此为了获得更加的性能，需要调整一下HTML的大小。首先就是尽量的摆脱JavaScript，并避免使用大的矿建，例如jQuery。有时候使用原始的JavaScript要比别的框架快。

另外，尽量的异步加载JavaScript文件——特别是不直接影响到页面行为时，例如分析脚本。

最后——让使用到的图片，跟实际需要的一样大小。如之前提到的，尽量使用sprite sheets，以此节省内存和提升速度。

更多相关信息，可以看一下：

WWDC 2012 session #601 – 在iOS中优化UIWebView和网站中的Web内容。

19) 设置阴影路径

如果需要在view活layer中添加一个阴影，该如何处理呢？

大多数开发者首先将QuartzCore框架添加到工程中，然后添加如下代码：

```
#import &lt;QuartzCore/QuartzCore.h&gt;

// Somewhere later ...
UIView *view = [[UIView alloc] init];

// Setup the shadow ...
view.layer.shadowOffset = CGSizeMake(-1.0f, 1.0f);
view.layer.shadowRadius = 5.0f;
view.layer.shadowOpacity = 0.6;
```

看起来非常容易，不是吗？

然而不幸的是上面这种方法有一个问题。Core Animation在渲染阴影效果之前，必须通过做一个离屏(offscreen)才能确定view的形状，而这个离屏操作非常耗费资源。

下面有一种方法可以更容易的让系统进行阴影渲染：设置阴影路径！

```
view.layer.shadowPath = [[UIBezierPath bezierPathWithRect:view.bounds] CGPath];
```

通过设置阴影路径，iOS就不用总是再计算该如何绘制阴影了。只需要使用你预先计算好的路径即可。有一点不好的是，根据view的格式，自己可能很难计算出路径。另外一个问题就是当view的frame改变时，必须每次都更新一下阴影路径。

如果你想了解更多相关信息，Mark Pospesel写了一篇很棒的文章：shadowPath。

20) 优化TableView

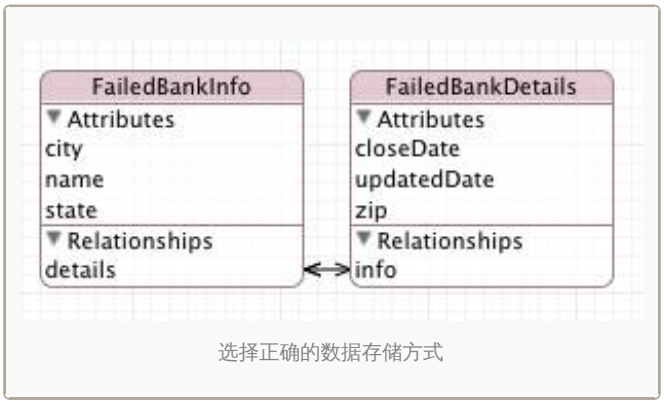
Table views需要快速的滚动——如果不能的话，用户会感觉到停顿。

为了让table view平滑的滚动，确保遵循了如下建议：

- 设置正确的reuseIdentifer以重用cell。
- 尽量将view设置为不透明，包括cell本身。
- 避免渐变，图像缩放以及离屏绘制。
- 如果row的高度不相同，那么将其缓存下来。
- 如果cell显示的内容来此网络，那么确保这些内容是通过异步来获取的。
- 使用shadowPath来设置阴影。
- 减少subview的数量。
- 在cellForRowAtIndexPath:中尽量做更少的操作。如果需要做一些处理，那么最好做过一次之后，就将结果缓存起来。
- 使用适当的数据结构来保存需要的信息。不同的结构会带来不同的操作代价。
- 使用rowHeight, sectionFooterHeight 和 sectionHeaderHeight 来设置一个恒定 高度，而不要从delegate中获取。

21) 选择正确的数据存储方式

当需要存储和读取大量的数据时，该如何选择存储方式呢？



选择正确的数据存储方式

有如下选择：

- 使用*NSUserDefaults进行存储*
- 保存为XML，JSON或Plist格式的文件
- 利用NSCoding进行归档
- 存储到一个本地数据库，例如SQLite。
- 使用*Core Data*.

使用*NSUserDefaults*有什么问题呢? 虽然NSUserDefaults很好并且容易，不过只只针对于存储小量数据（比如你的级别，或者声音是开或关）。如果要存储大量的数据，最好选择别的存储方式。

大量数据保存为结构化的文件也可能会带来问题。一般，在解析这些结构数据之前，需要将内容全部加载到内存中，这是很消耗资源的。虽然可以使用SAX来处理XML文件，但是这有点复杂。另外，加载到内存中的所有对象，不一定全部都需要用到。

那么使用NSCoding来保存大量数据怎么样呢？因为它同样是对文件进行读写，因此依然存在上面说的问题。

要保存大量的数据，最好使用SQLite或Core Data。通过SQLite或Core Data可以进行具体的查询——只需要获取并加载需要的数据对象——避免对数据进行不合理的搜索。在性能方面，SQLite和Core Data差不大。

SQLite和Core Data最大的区别实际上就是用法上。Core Data代表一个对象模型，而SQLite只是一个DBMS。一般，苹果建议使用Core Data，不过如果你有特殊的原因不能使用Core Data的话，可以使用低级别的SQLite。

在程序中，如果选择使用SQLite，这里有个方便的库FMDB ：可以利用该库操作SQLite数据库，而不用深入使用SQLite C API。

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

1

- 支持
- 反对