**优化知识集合**

1. **用ARC管理内存**

它自动为你管理retain和release的过程，所以你就不必去手动干预了;

**2.** **在正确的地方使用reuseIdentifier**

这对性能的影响可是相当大的，尤其会使app的滚动体验大打折扣;

1. **尽量把views设置为完全不透明**

设置它们的opaque(不透明)属性为YES, 例如一个黑色半透明的可以设置为一个灰色不透明的View替代,因为子视图的半透明和父视图的透明,会导致GPU有一个Blending混合像素计算,而这个Blending是会导致性能损失的;

**4.** **避免庞大的XIB**

当你加载一个XIB的时候所有内容都被放在了内存里，包括任何图片。如果有一个不会即刻用到的view，你这就是在浪费宝贵的内存资源了;

**5. 不要block主线程**

永远不要使主线程承担过多。因为UIKit在主线程上做所有工作，渲染，管理触摸反应，回应输入等都需要在它上面完成。一直使用主线程的风险就是如果你的代码真的block了主线程，你的app会失去反应, 大部分阻碍主进程的情形是你的app在做一些牵涉到读写外部资源的I/O操作，比如存储或者网络。或者使用像 AFNetworking这样的框架来异步地做这些操作;

**6. 在Image Views中调整图片大小**

如果要在UIImageView中显示一个来自bundle的图片，你应保证图片的大小和UIImageView的大小相同。在运行中缩放图片是很耗费资源的，特别是UIImageView嵌套在UIScrollView中的情况下。

如果图片是从远端服务加载的你不能控制图片大小，比如在下载前调整到合适大小的话，你可以在下载完成后，最好是用后台线程缩放一次，然后在UIImageView中使用缩放后的图片

**7. 选择正确的Collection**

就是说，如果搜索一个元素，NSSet的效率会比NSArray高。NSSet中元素的存储和访问都是一个hash的过程。比如你要存储元素A，一个hash算法直接就能直接找到A应该存储的位置；同样，当你要访问A时，一个hash过程就能找到A存储的位置。而对于NSArray，若想知道A到底在不在数组中，则需要一个一个元素比较，显然效率不高;

**8. 打开gzip压缩**

减小文档的一个方式就是在服务端和你的app中打开gzip。这对于文字这种能有更高压缩率的数据来说会有更显著的效用。好消息是，iOS已经在NSURLConnection中默认支持了gzip压缩

**9. 重用和延迟加载Views**

这里我们用到的技巧就是模仿UITableView和UICollectionView的操作: 不要一次创建所有的subview，而是当需要时才创建;有时候需要依据实际情况而定,当显示一个view的时候,可以有两种方法,第一种就是先创建,然后隐藏,需要显示的时候再显示,第二种就是根据需要的时候,再创建,再显示,第一中方法更加消耗内存,但速度快,第二种方法,消耗更小内存,速度慢;

**10. Cache, Cache, 还是Cache！**

缓存所需要的，也就是那些不大可能改变但是需要经常读取的东西,比如远端服务器的响应，图片，甚至计算结果，比如UITableView的行高。

NSURLConnection默认会缓存资源在内存或者存储中根据它所加载的HTTP Headers。你甚至可以手动创建一个NSURLRequest然后使它只加载缓存的值

+ (NSMutableURLRequest\*)imageRequestWithURL:(NSURL\*)url {

NSMutableURLRequest\*request = [NSMutableURLRequest requestWithURL:url];

request.cachePolicy = NSURLRequestReturnCacheDataElseLoad;//this will make sure the request always returns the cachedimage

request.HTTPShouldHandleCookies =NO;

request.HTTPShouldUsePipelining =YES;

[request addValue:@"image/\*" forHTTPHeaderField:@"Accept"];

return request;

}

**11. 权衡渲染方法**

在iOS中可以有很多方法做出漂亮的按钮。你可以用整幅的图片，可调大小的图片，或者可以用CALayer，甚至OpenGL来画它们, 每个不同的解决方法都有不同的复杂程度和相应的性能, 简单来说，就是用事先渲染好的图片更快一些，因为如此一来iOS就免去了创建一个图片再画东西上去然后显示在屏幕上的程序, 但是静态图片会造成bundle大小的不断增大;折中的办法是使用可变大小的静态小图片;

**12. 处理内存警告**

如果你的app收到了内存警告，它就需要尽可能释放更多的内存。最佳方式是移除对缓存，图片object和其他一些可以重创建的objects的strong references., 在app delegate中使用applicationDidReceiveMemoryWarning: 的方法在你的自定义UIViewController的子类(subclass)中覆盖didReceiveMemoryWarning注册并接收 UIApplicationDidReceiveMemoryWarningNotification 的通知一旦收到这类通知，你就需要释放任何不必要的内存使用

**13. 重用大开销的对象**

一些objects的初始化很慢，比如NSDateFormatter和NSCalendar。然而，你又不可避免地需要使用它们，比如从JSON或者XML中解析数据。

想要避免使用这个对象的瓶颈你就需要重用他们，可以通过添加属性到你的class里或者创建静态变量来实现

- (NSDateFormatter\*)formatter {

if(!\_formatter) {

\_formatter = [[NSDateFormatter alloc] init];

\_formatter.dateFormat =@"EEE MMM dd HH:mm:ss Z yyyy";

// twitter date format

}

return\_formatter;

}

**14. 使用Sprite Sheets**

Sprite sheet可以让渲染速度加快，甚至比标准的屏幕渲染方法节省内存。

**15. 避免反复处理数据**

在内存中操作数据使它们满足你的数据结构是开销很大的。比如你需要数据来展示一个table view,最好直接从服务器取array结构的数据以避免额外的中间数据结构改变;

**16. 选择正确的数据格式**

解析JSON会比XML更快一些，JSON也通常更小更便于传输。从iOS5起有了官方内建的JSON deserialization 就更加方便使用了, 但是XML也有XML的好处,当解析大量数据的时候,xml比json的效率高;

**17. 正确地设定Background Images**

使用UIColor的 colorWithPatternImage来设置背景,或者在view中添加一个UIImageView作为一个子View, 如果你使用全画幅的背景图，你就必须使用UIImageView因为UIColor的colorWithPatternImage是用来创建小的重复的图片作为背景的, 如果你用小图平铺来创建背景，你就需要用UIColor的colorWithPatternImage来做了，它会更快地渲染也不会花费很多内存

**18. 减少使用Web特性**

UIWebView很有用，用它来展示网页内容或者创建UIKit很难做到的动画效果是很简单的一件事。但是你可能有注意到UIWebView并不像驱动Safari的那么快;

**19. 设定Shadow Path**

Core Animation不得不先在后台得出你的图形并加好阴影然后才渲染，这开销是很大的。使用shadowPath的话就避免了这个问题：使用shadow path的话iOS就不必每次都计算如何渲染，它使用一个预先计算好的路径

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

**20. 优化你的Table View**

正确使用reuseIdentifier来重用cells

尽量使所有的view opaque，包括cell自身避免渐变，图片缩放，后台选人

缓存行高如果cell内现实的内容来自web，使用异步加载，缓存请求结果

使用shadowPath来画阴影

减少subviews的数量尽量不适用cellForRowAtIndexPath:，如果你需要用到它，只用一次然后缓存结果

使用正确的数据结构来存储数据

尽量使用rowHeight, sectionFooterHeight 和 sectionHeaderHeight来设定固定的高，不要请求delegate

**21. 选择正确的数据存储选项**

当做本地数据存储时你会怎么做？

你有很多选择，比如：

使用NSUerDefaults

使用XML, JSON, 或者 plist

使用NSCoding存档

使用类似SQLite的本地SQL数据库

使用 Core Data

NSUserDefaults的问题是什么？虽然它很nice也很便捷，但是它只适用于小数据，比如一些简单的布尔型的设置选项，再大点你就要考虑其它方式了XML这种结构化档案呢？总体来说，你需要读取整个文件到内存里去解析，这样是很不经济的。使用SAX又是一个很麻烦的事情。

NSCoding？不幸的是，它也需要读写文件，所以也有以上问题。

当存储大块数据时,以上的方法都不适用. 在这种应用场景下，使用SQLite 或者 Core Data比较好。使用这些技术你用特定的查询语句就能只加载你需要的对象。

在性能层面来讲，SQLite和Core Data是很相似的。

他们的不同在于具体使用方法。

Core Data代表一个对象的graph model，但SQLite就是一个DBMS。

Apple在一般情况下建议使用Core Data，但是如果你有理由不使用它，那么就去使用更加底层的SQLite吧。

**22. 加速启动时间**

能做的就是使它尽可能做更多的异步任务,比如在首页需要请求的接口,就先在启动页加载时,异步请求;这样app在加载启动页的时候,其实就已经在请求网络了;

**23. 使用Autorelease Pool**

假如你创建很多临时对象，你会发现内存一直在减少直到这些对象被release的时候。这是因为只有当UIKit用光了autorelease pool的时候memory才会被释放。

好消息是你可以在你自己的@autoreleasepool里创建临时的对象来避免这个行为

**24. 选择是否缓存图片**

imageNamed的优点是当加载时会缓存图片。相反的，imageWithContentsOfFile仅加载图片。如果你要加载一个大图片而且是一次性使用，那么就没必要缓存这个图片，用imageWithContentsOfFile足矣，这样不会浪费内存来缓存它。

然而，在图片反复重用的情况下imageNamed是一个好得多的选择;

**25. 尽量避免日期格式转换**

你可以控制你所处理的日期格式，尽量选择Unix时间戳。你可以方便地从时间戳转换到NSDate