iOS 开发 UITableView 加载图片的内存管理①

黄天柱, 涂时亮

(复旦大学 计算机科学技术学院, 上海 200433)

摘 要: 首先对移动操作系统 iOS 开发过程的内存泄露、缓存等相关知识作了详细叙述, 介绍了使用 iOS 集成开发环境 Xcode 进行内存调试的方法. 针对视图 UITableView 中含有大量网络图片时的内存问题, 为了达到内存最优化和良好用户体验的目标, 提出异步下载网络图片和本地缓存的解决方法. 最后测试结果表明 UITableView 在下载图片和滑动的过程中内存占用量趋于稳定, app 的运行速度较为顺畅, 拥有较好的用户体验.

关键词: iOS; UITableView; 内存管理; 内存优化; 引用计数; Instruments

Memory Management and Optimization of iOS Development

HUANG Tian-Zhu, TU Shi-Liang

(School of Computer Science, Fudan University, Shanghai 200433, China)

Abstract: This paper describes the knowledge of memory management and cache of iOS development in detail, and introduces methods of debugging memory problems using tools which is provided by Apple IDE Xcode, for the memory problems when UITableView contains large amounts of network pictures, with the goal of memory optimization and best user experience. It proposed network of asynchronous-download pictures and the local cache solution. Finally, test results show that the memory allocation is safe and stable, the app runs more smoothly, and have a better user experience.

Key words: iOS; UITableView; memory management; memory optimization; reference count; Instruments

1 概述

iOS 是由苹果公司开发的手持设备操作系统, 最新版的 iOS 系统[1](iOS5)中, 系统操作占用大概 774.4MB 的内存空间. iOS 与苹果的 Mac OS X 操作系统一样, 它也是以 Darwin 为基础的, 因此同样属于类 Unix 的商业操作系统. Cocoa 是苹果公司为 Mac OS X 所创建的原生面向对象的编程环境. Cocoa 应用程序一般在苹果公司的开发工具 Xcode 和 Interface Builder 上用编程语言 Objective-C 写成. iOS 对开发者进行内存管理的要求很严格, 对于程序 crash、有 memory leak、以及内存占用量过大的程序, 都不能通过审核. 这样需要开发者在开发过程中对程序中的每个变量都需要细心地进行内存管理方面的工作, Cocoa 环境的一个特点是它可以管理动态分配的内存. 不仅需要开发者在开发过程中对每个对象的引用计数进行维护, 防止 memeory leak(内存泄露). 而且如果程序一次加载太多的数据, 那么也会导致内存短缺. 当使用占用内存较大的图像、音频时, 在内存中保留所有东西会导致问题. 在 iOS 应用中, UITableView 应该是使用率最高的视图之一了. iPod、时钟、日历、备忘录、Mail、天气、照片、电话、短信、Safari、App Store、iTunes、Game Center 几乎所有自带的应用中都能看到它, 可见它的重要性. 然而在使用第三方应用时, 却经常遇到性能上的问题. 为了使表格滑动较为流畅, UITableView 以标识 UITableViewCell 达到不开销内存只改变 cell 内容而重用, 是苹果为了实现大量数据显示而采用的一种节省内存的机制. 由于 UITableView 自身的重用机制、可能照成每个表格 cell 的内存错乱.

① 收稿时间:2012-01-19;收到修改稿时间:2012-03-04

图片占用内存过大会造成程序 crash. 所以动态分配内存，推迟真正需要时再加载资源，并在系统需要的时候释放内存， 以及通过一种被成为缓存的策略，才能管理内存达到最优化.

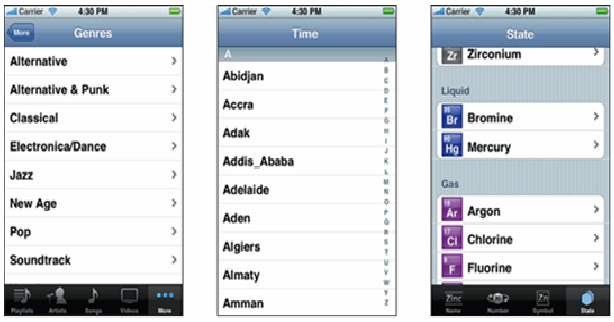


图 1　UITableView 的常见形式

2　内存管理基本原理

iOS 没有提供垃圾收集机制,它依赖于一个引用计数的内存管理系统. 这意味着开发人员必须控制何时创建对象、保留对象和从内存中释放对象. 不仅如此，一个开发者是否对 app 进行过内存优化对整个 app 运行时的速度、流畅度等都有根本性的作用.

首先，内存管理有两条简单的规则. 创建时，每个对象都有一个初始值为 1 的保留计数. 释放时，每个对象的保留计数都为 0. 开发人员自行管理对象在整个生命周期中的保留操作. 应该确保对象从头至尾不会被过早释放，而到了正确的时候一定要释放它. 每个 Cocoa 对象携带一个整数用来指示对其存在感兴趣的其它对象的数目. 这个整数被称为对象的保留数(retain count). 当创建一个对象时，或者通过一个类工厂方法或者使用 alloc 或 allocWithZone: 类方法，Cocoa 做了一些很重要的事情: 它设置对象的 isa 指针- NSObject 类[2]的唯一公共成员变量-以指向这个对象的类，这样把这个对象集成到运行时视图类层次. 它设置对象的保留数(retain count)- 一种由运行时管理的隐藏的成员变量为 1. 在对象分配后，一般会设置它的成员变量为一个合理的初始值. (NSObject 声明 init 方法作为这个目的的原形). 这个对象现在已经可以使用了，可以发送消息给它，把它传递给其他对象等等.

当释放一个对象，发送一个 release 消息给它 - NSObject 减少其保留数. 如果这个保留数从 1 变成 0，这个对象会被释放. 释放分成两个步骤. 首先，对象的 dealloc 方法被调用来释放成员变量并动态释放分配的内存. 然后操作系统销毁对象自身并回收该对象曾经占用的内存. 如果在从别处接收到一个对象时给它发送了一个 retain 消息，这个对象的保留数(retain count)被增加为 2. 现在在释放之前需要两个 release 消息. 图 2 图示了这个相对简化的场景 .

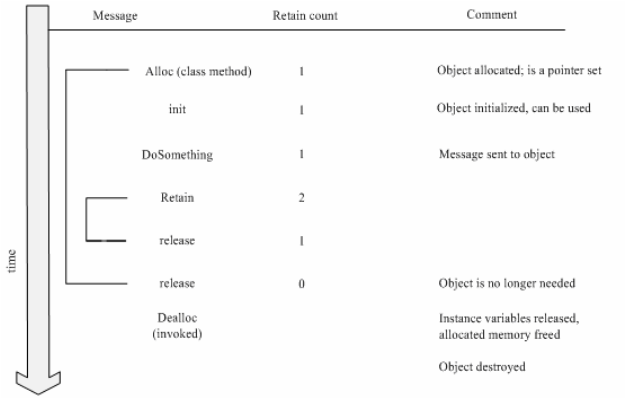


图 2　引用保留计数的基本原理

3　iOS开发内存调试工具

在程序编写直至发布的过程中，内存调试的方法也由 IDE 的工具功能不同而各自不同. 总的来说，较为常用的调试方法有设置断点、使用文本消息、Log 等. UITableView 作为使用率最高的视图，在开发过程中 Xcode 提供了一系列的调试工具帮助开发者进行内存管理和优化的相关工作.

在调试应用程序时，Instrument[3]用于监控性能和泄露检测，跟踪、识别和解决程序中的内存泄露问题. 如图显示了一个存在两处泄露问题的应用程序，一个是用 malloc()构建了一个字符串，但是没有调用相应的 free()，使这个字符串的 retain count 为 1，而不能释放.

可以点击各个 leak 查看各处的泄露列表，包括泄露显示的泄露内存量、泄露开始的地址以及泄露对象的类型. 当内存泄露检测开始时，会看到列表中出现了泄露的对象，如果希望看到内存泄露的细节，点击列表中的对象，然后点击屏幕下方的 Extended Detail 按钮.

在此视图中，可以发现一个栈跟踪，它将泄露追溯到它的创建处. 如图 3 所示，当前的内存泄露是在已经分配内存的 leakCString 中分配的. 发现对象的起源有助于追踪在对象的生命周期中何时会发生泄露，

解决内存问题.

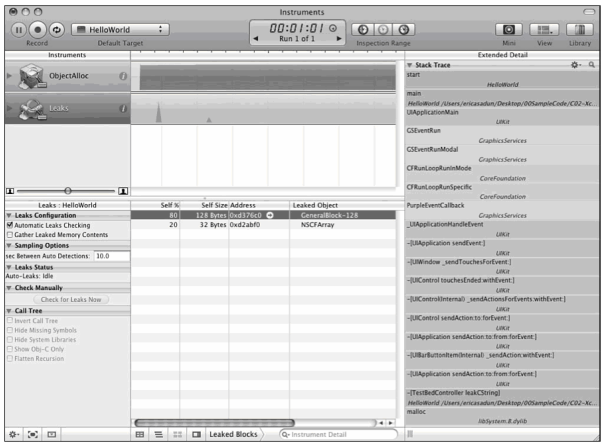


图 3 Memory leak 时的栈跟踪

4 UITableView加载图片的分析和实现

由于 UITableView 自身的重用机制 4、可能照成每个表格 cell 的内存错乱. 图片占用内存过大会造成程序 crash. 为了使 UITableView 在滑动过程中, 不仅能够进行流畅的进行浏览, 还能得到图片下载的相关信息, 就需要对图片下载中和下载完成时进行内存管理. 在优化 UITableView 的内存问题之前, 必须充分了解 tableview 的特点, 内存占用方式, 以达到内存最优化.

4.1 UITableView 特点

特点 1: 视图布局: UITableView 类继承自 UIScroll View 类[5], 像其他视图一样, 其实例通过窗体定义自己的边界, 还可以是 其他视图的子类或父类. UITableViewController 负责处理布局, 并会使用一个 UITableView 进行填充.

特点 2: 指定数据源: UITableView 实例依赖外部资源按需为新表格单元或现有表格单元提供内容, 数据源根据索引路径提 供表格单元格, 索引路径是 NSIndexPath 类的对象, 描述通过数据树到达特定节点的路径, 即它们的分段和它们的行.

myIndexPath=[NSIndexPath indexPathForRow:5 inSection:0];

特点3: 指定委托:UITableView实例使用委托响应用户交互, 并实现有意义的响应, 委托告知表格将响应这些交互的责任移交给指定对象, 委托必须实现 UITableViewDelegate 协议.

特点 4: 重用机制: 重用机制是苹果为了实现大量数据显示滚动流畅而采用的一种节省内存的机制, 为了有较好的用户体验, 苹果对 UITableView 中的每个 UITableViewCell(构成表格 UITableView 的基本元素, 即是单元格)使用了重用, 函数 dequeueReusable CellWithIdentifier 它返回的是一个受 identifier 管理定位的可重用的 UITableViewCell, 重复使用以节约内存使用量. UITableView 是 UIScrollView 的子类, 因此它可以自动响应滚动事件(一般为上下滚动). 它内部包含 0 到多个 UITableViewCell 对象, 每个 table cell 展示各自的内容. 当新 cell 需要被显示时, 就会调用 tableView:cellForRowAtIndexPath:方法来获取或创建一个 cell; 而不可视时, 它又会被释放. 由此可见, 同一时间其实只需要存在一屏幕的 cell 对象即可, 不需要为每一行创建一个 cell. 举例来说, 在系统刚启动时, tableview 可以显示假定 10 个 cell, , 在刚开始的时候 tableview 会生成 10 个 tableviewcell, 并且对应有自己的 tag 值, 假定为 0－9. 苹果为了尽量避免频繁的 add / remove view 或者控件之类, 采用下面的方法来实现: 在 tableview 向上滚动的时候, tag 为 0 的 cell 将不再显示; 然后我们把 tag 为 0 的 cell 移动到 tag 为 9 的 cell 下面, 重新设置相关的属性, 然后将 tag 为 1 的 cell 移动到 tag 为 0 的 cell 下面.

4.2 UITableView 加载网络图片带来的问题

问题 1: UITableViewCell 中下载图片阻滞系统 UI 主线程:

iOS 程序和传统的桌面程序的最大不同在于内存有限, 管理内存成了 iPhone 开发中时时刻刻需要谨记的事情. 类似的功能在桌面程序上无非是将 down 下来的数据缓存于内存中, 需要的时候画出来即可. 此法在 iOS 上却不可行, 尤其是在 UITableView 视图中, 当每个 cell 中都需要下载一张很大的图片时, cell 需要为新的图片绘在界面上, 但系统 UI 主线程会等待图片下载完毕, 然后绘制在相应的 cell 中. 即使图片已经下载完毕, 如果图片过大, 也会占用很大的内存, 每次用户滑动表格出现新的 table cell 的时候, uitableview 就会遇到性能上的问题, 普遍表现在滚动时比较卡. 如果一个 tableview 有几百个 cell, 在滑动过快的时候, 图片占有内存较大, 绘制图片占有系统资源过高, 会使阻滞滚动速度. 这样很容易出现 memory warning 甚至 crash 掉.

问题 2: UITableViewCell 的重用机制导致子视图重叠错乱

重用机制带来的问题就是 cell 中填充内容的错乱, 子视图重叠, 不能正确显示信息. , 如果一屏只能显示 10 行的图片, 在所有图片都加载完后, 滚动 TableView, 让隐藏在下面的行显示在屏幕上, 而这些行(比如 11 行)的图像会先显示第 1 行的图片, 12 行显示第 2 行的图片, 然后在显示属于它自己的图片. 以此类推, 后面的行都会出现重用上次屏幕中此位置 cell 的内容.

问题 3: 图片过多造成系统内存占用过大:

图片如果使用太多的内存, iphone 会警告应用程序委托和 UIViewController. 委托收到 applicationDidRecieveMemoryWarning: 回调, 视图控制器收到 didRecieveMemoryWarning. 如果使用太多内存, iPhone 将终止应用程序, 使用户回到 springboard. 正如苹果公司反复指出的那样, 这也许不是希望用户得到的用户体验. 必须小心地管理内存, 在低内存的情况下释放内存. 导致低内存的原因通常有两个: 一是内存泄露, 即分配了不能被访问或重用的内存块; 二是一次存储了太多的数据.

4.3 解决方案

为了解决以上问题, 能够流畅地浏览 table view 中的内容, 对内存进行控制是必不可少的. 解决方案如下:

首先, 在 UITableView 重用 cell 时, 为了防止 cell 中填充内容的错乱, 需要对重用机制稍作修改, 如果 cell== nil, 通过分配并初始化来创建(create)一个对象(比如[[UITableViewCell alloc] init]将其分配到内存中, 如果 cell 被重用(即 cell != nil),需要主动先将图片从重用的 cell 中去除, ([oldImage removeFromSuperview]), 再将在 table view 这个位置的图片加载到 cell 中. 同上所述一致, 如果图片没有下载完成, 就以 loading 的界面方式展示给用户.

其次, 对于每个 cell 中下载图片的任务, 不能占用程序主线程(主线程处理程序 UI 任务), 使用图片的 url 地址作为图片的标识, 新开线程下载图片, 没有下载到图片的时候 cell 中图片使用一个 loading 图或者 UIActivityView(转动的滑轮, iphone 上一种展示 loading 的方式)展示给用户, 如图 4 所示. 这时对于每个 cell 对象, 唯一的 url 就代表了这个图片.

在线程函数中把所有的图片下载到本地, connectionDidFinishLoading 函数下载完成后, 将图片缓存到本地, 通过类似于字典一样的键值对数据结构存储图片(键就是图片唯一的 url,值就是本地在内存或者文件目录的图片). 同时通过 performSelectorOnMainThread 通知主线程, 主线程 reload UITableView, 将下载完成的本地图片显示在对应的 cell 中. 当用户滑动 table 到需要新的区域, 需要图片显示时, loadImageFromURL:(NSURL*)url 载入图片时如果图片在字典中找到, 则自动加载到 cell 显示, 如果尚未下载完成, 则继续下载图片. 这样, UITableView 下载图片不会占用或阻塞主线程, 保证用户操作时的界面显示的流畅性.

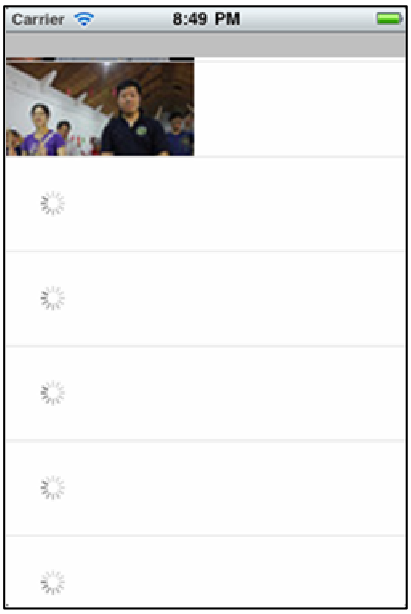


图 4 图片未下载完成时候的 loading 展示

表格中 cell 过多, 会有过多的图片下载线程, NSOperationQueue 可以对各个线程任务进行管理和调度, 加入、移除下载队列, 设置并发线程最大数、调整下载任务优先级, 比如用户滑动到的 visible cell 的下载任务优先级最高.

缓存: 如果一次加载太多的数据, 那么也会导致内存短缺. 当使用内存密集型资源, 在内存中保留所有东西会导致问题. 通过一种被成为缓存的策略, 可以推迟真正需要时再加载资源, 并在系统需要的时候释放内存.

最简单的一种方法是从一个 NSMutableDictionary 对象构建缓存. 基本对象缓存的工作原理是这样的: 当被请求时, 缓存检查被请求对象是否已经加载. 如

果尚未加载, 缓存发出一条基于对象名称的加载请求. 对象加载方法可能从本地检索数据, 也可能从 web 检索数据. 加载后, 它将新信息存储在内存中, 以便下次能快速调用.

下面代码执行缓存职责的第一部分. 它直到数据被请求时才将新数据加载到内存.

```
-(id) retrieveObjectNamed: (NSString*) someKey
{
  id object = [self.myCach objectForKey: someKey];
  if (!object)
  {
    object = [self loadObjectNamed: someKey];
    [self.myCache setObject:object forKey: someKey];
  }
  return object;
}
```

缓存的第二个职责是当应用程序遇到低内存状况时能够自我清除. 对于基于字典的缓存, 只需清除对象. 当下一次检索请求到达时, 缓存可以重新加载被请求的对象.

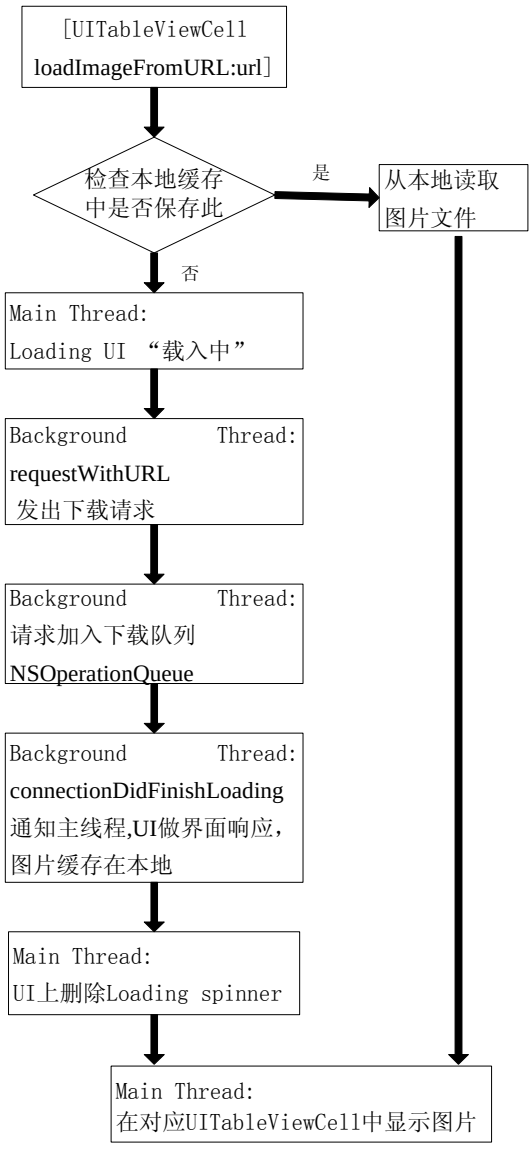
```
-(void) respondToMemoryWarning
{[self.myCache removeAllObjects];}
```

通过将推迟加载与内存触发的清除相结合, 缓存可以对内存有好的方式运行. 对象被加载到内存后, 可以被使用和重用, 而不存在加载延时. 但是, 当内存紧缺时, 缓存会释放对象, 以确保程序能够运行.

比如, 自动读取文件载入更新数据对用户来说也很友好, 这减少了用户等待下载的时间. 例如载入 50 条信息时, 那就可以在滚动到倒数第 10 条以内时, 将 table 最前面 10 条的图片从 UI 中移除, 并作为缓存保存在 app 的本地目录下, 当用户重看这几条信息的时候, cell 自动从文件加载图片.

```
-(void)tableView:(UITableView*)tableView
willDisplayCell:(UITableViewCell*)cell
forRowAtIndexPath:(NSIndexPath *)indexPath {
  if (count - indexPath.row >50 && !updating) {
    updating = YES;
    [self update]; // remove 10 photos in the head and save the 10 photos as files
  }
}// update 方法获取到结果后, 设置 updating 为 NO.
```

相关流程图如下:



5 测试结果

调试程序确定较好的用户体验和非 crash 的前提下, 在发布程序之前, 还可以使用开发集成环境 Xcode 自带的内存监视工具, 可以较为直接地查看程序内存分配情况.

5.1 使用 Instruments 监视缓存对象的内存分配

可以使用模拟器中的模拟内存警告, 发送对应用程序委托和视图控制器的调用, 请求它们释放不需要

的内存. 用于实时查看内存分配的 Instruments 可以监视那些释放. 确保在遇到内存警告时, 应用程序能够妥善应对. 当遇到内存警告时, 缓存做出响应, 即释放数据. 如图 5 所示.

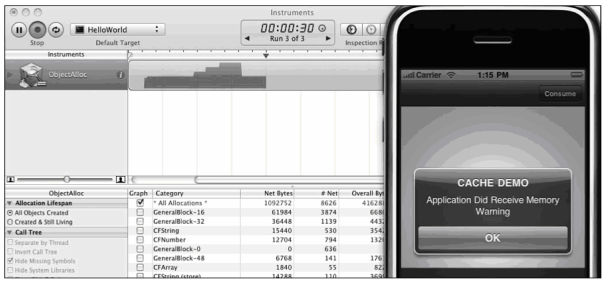


图 5　使用 Instrument 监视内存分配

从图中看出, 在正常的重用机制的 uitableview 中, 随着用户滚动, 程序下载图片占用程序内存不断陡增, 当内存过高时候, 系统会收到 memory warning,自动释放内存(图中内存占用图降到初始值). 以至于用户浏览相同图片时需要重新下载, 等待时间增加.

使用异步下载图片以及本地缓存的机制后, 内存占用如图 6 所示:

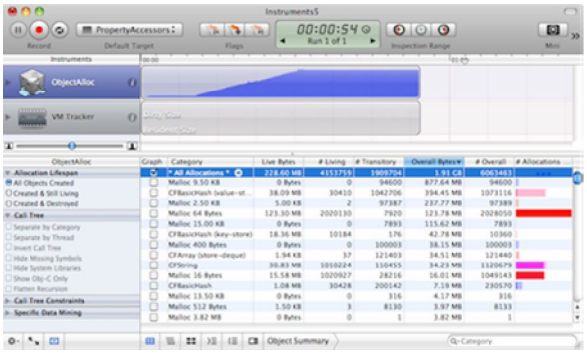


图 6　内存优化后 tableview 的内存占用情况

在收到 memory warning 之前将较大的图片从内存中移除, 保存在文件中, 使用 Instruments 查看程序使用内存占用量情况趋于稳定, 如图 6 所示, 在初期, 随着用户浏览图片量的增加, 内容占用量逐渐增加(趋于线性), 到达某个阈值之后, 内存不在增加, 趋于稳定.

6　总结

至今, 1.6 亿 iPhone、iPod、iPad 用户, 从苹果应用程序商店 App Store(下称苹果 App Store)下载应用程序的次数已经超过了 100 亿次. 随着移动通信技术的不断发展, 苹果产品的风靡, 越来越多的开发者加入到了 iOS app 开发的阵营中. 而由于苹果公司对开发者进行内存管理的严格要求和为了是 app 有最佳的用户体验, 在开发过程中注重内存管理就成了每个开发者都需要注意的问题. 苹果公司自身也在对内存管理方面进行优化, 在最新发布的 iOS5 sdk 中的 Automatic Reference Counting (ARC) 编译期技术, 可以简化 Objective-C 编程在内存管理方面的工作量, 程序人员可以在变量维护、内存的 retain, 释放等方面减少很多精力. 总的来说, 对于内存要求严格的苹果开发者审核来说, 内存优化是一个好的 app 的基础, 只有做到最佳的内存优化, 才能做到较好的用户体验.

参考文献

1 Sadun E. The iPhone Developer's Cookbook. 2nd Edition. Addison-Wesley Publishing Company,2010.

2 NSObject Protocol Reference. general concepts, iPhone Library Documentation, Apple Developer Center (ADP). http://developer.apple.com/iphone/library/documentation

3 Introduction to Instruments User Guide. MACOS X Reference Library, ADP.

4 Xiaodong P. Research of Iphone Application VI Design Based on Children Cognition Feature. IEEE 11th International Conference on Computer-Aided , 2010,1:293–296.

5 Iulia-Maria T, Ciocarlie H. Best practices in iPhone programming Model-View-Controller architecture–Carousel component development. International Conference on Computer as a Tool (EUROCON), 2011:1–4.