# 小议 iOS 内存管理

----小议 iphone 内存管理与属性

注：网上看到这篇关于 iPhone 的内存管理文章，觉得很好，转成 PDF 让大家学习学习。
来源：http://www.cnblogs.com/ET-Union/archive/2011/08/17/2143774.html

## 一、 前言

对于大多数从 C++或者 JAVA 转过来学习 Object-C（以下简称 OC）的人来说，OC 这门语言看起来非常奇怪，用起来也有点麻烦。

OC 没有像 JAVA 一样的垃圾回收机制，也就是说，OC 编程需要程序员手动去管理内存。这就是为什么它烦的原因，苹果却一直推崇开发者在有限硬件资源内写出最优化的代码，使用 CPU 最少，占用内存最小。

## 二、 基本原理

对象的创建：

OC 在创建对象时，不会直接返回该对象，而是返回一个指向对象的指针，因此出来基本类型以外，我们在 OC 中基本上都在使用指针。

```
ClassA  *a = [[ClassA   alloc]  init];
```

在[ClassA   alloc]的时候，已经发送消息通知系统给 ClassA 的对象分配内存空间，并且返回了指向未初始化的对象的一个指针。

未初始化的 ClassA 对象接手到 init 消息，init 返回指向已初始化后的 ClassA 对象的一个指针，然后将其赋值给变量 a。

在创建并使用完一个对象的时候，用户需要手动地去释放该对象。

```
[a   dealloc];
```

如果指针 a 和 b 同时指向堆中同一块内存地址

```
ClassA  *a = [[ClassA   alloc]  init];
ClassA  *b = a;
[a   dealloc];
```

当执行到第三行的时候，指针 b 就成了无头指针。这是一个在 C++中也是常见的错误，我们需要避免这类错误，因为无头指针是危险的。

引用计数：

OC 在内存管理上采用了引用计数（retain count），在对象内部保存一个数字，用来表示被引用的次数。init、new 和 copy 都会让 retain count 加 1。当销毁对象的时候，系统不会直接调用 dealloc 方法，而是先调用 release，让 retain count 减 1，当 retain count 等于 0 的时候，系统才会调用 dealloc 方法来销毁对象。

在指针赋值的时候，retain count 是不会自动增加的，为了避免上面所说的错误，我们需要在赋值的时候手动 retain 一次，让 retain count 增加 1。

```
ClassA  *a = [[ClassA   alloc]  init];  // retain count = 1
ClassA  *b = a;
[b   retain];  // retain count = 2
[a   dealloc];
```

这样在执行到第四行的时候，对象的 retain count 只是减了 1，并没有被销毁，指针 b 仍然有效。

内存泄露：

就如上面列子所示，当生成 ClassA 对象时，指针 a 拥有对该对象的访问权。如果失去了对一个对象的访问权，而又没有将 retain count 减至 0，就会造成内存泄露。也就是说，分配出去的内存无法回收。

```
ClassA  *a = [[ClassA   alloc]  init];
a  =  nil;
```

# 三、 Autorelease Pool

为了方便程序员管理内存，苹果在 OC 中引入了自动释放池(Autorelease Pool)。在遵守一些规则的情况下，可以自动释放对象。但即使有这么一个工具，OC 的内存仍需要程序员时刻关注（这个自动释放池跟 JAVA 的垃圾回收机制不是一回事，或者说，骑马都追不上 JAVA 的机制，可能连尘都吃不到）。

```
ClassA  *a = [[[ClassA   alloc]  init]  autorelease];
//retain count = 1，但无需 release
```

Autorelease Pool 的原理：

autorelease pool 全名叫做 NSAutoreleasePool，是 OC 中的一个类。autorelease pool 并不是天生就有的，你需要手动的去创建它

```
NSAutoreleasePool  *pool = [[NSAutoreleasePool  alloc]  init];
```

一般地，在新建一个 iphone 项目的时候，xcode 会自动地为你创建一个 autorelease pool，这个 pool 就写在 Main 函数里面。

在 NSAutoreleasePool 中包含了一个 可变数组，用来存储被声明为 autorelease 的对象。当 NSAutoreleasePool 自身被销毁的时候，它会遍历这个数组，release 数组中的每一个成员（注意，这里只是 release，并没有直接销毁对象）。若成员的 retain count 大于 1，那么对象没有被销毁，造成内存泄露。

默认的 NSAutoreleasePool 只有一个，你可以在你的程序中创建 NSAutoreleasePool，被标记为 autorelease 的对象会跟最近的 NSAutoreleasePool 匹配。

```
NSAutoreleasePool  *pool = [[NSAutoreleasePool  alloc]  init];
//Create some objects
//do something…
[pool  release];
```

你也可以嵌套使用 NSAutoreleasePool ，就像你嵌套使用 for 一样。

即使 NSAutoreleasePool 看起来没有手动 release 那么繁琐，但是使用 NSAutoreleasePool 来管理内存的方法还是不推荐的。因为在一个 NSAutoreleasePool 里面，如果有大量对象被标记为 autorelease，在程序运行的时候，内存会剧增，直到 NSAutoreleasePool 被销毁的时候才会释放。如果其中的对象足够的多，在运行过程中你可能会收到系统的低内存警告，或者直接 crash。

Autorelease Pool 扩展：

如果你极具好奇心，把 Main 函数中的 NSAutoreleasePool 代码删除掉，然后再自己的代码中把对象声明为 autorelease，你会 发现系统并不会给你发出错误信息或者警告。用内存检测工具去检测内存的话，你可能会惊奇的发现你的对象仍然被销毁了。

其实在新生成一个 Run Loop 的时候，系统会自动的创建一个 NSAutoreleasePool ，这个 NSAutoreleasePool 无法被删除。

在做内存测试的时候，请不要用 NSString。OC 对字符串作了特殊处理

```
NSString *str =[ [NSString alloc] stringWithString:@”123”];
```

在输出 str 的 retain count 的时候，你会发现 retain count 大于 1。

# 四、 手动管理内存

使用 alloc、new、copy 创建一个对象，该对象的 retain count 都等于 1，需要用 release 来释放该对象。谁创建，谁去释放。在这 3 钟方法以外的方法创建的对象，都被系统默认的声明为 autorelease。

```
ClassA *a = [[ClassA alloc] init];
```

```
ClassA  *b = a;
[b   retain];
//do smoething
[b release];
b  =  nil;
```

把一个指针赋值给另外一个指针的时候，a 指针所指向的对象的引用次数并没有增加，也就是说，对象的 retain count 依然等于 1。只有在 retain 了之后，retain count 才会加 1。那么，如果这时候执行[a  release]，只是 a 指针放弃了对对象的访问权，对象的 retain count 减 1，对象没有被销毁。只有当 b 也执行了 release 方法之后，才会将对象销毁掉。因此，谁 retain 了，谁就要 release。

在对象被销毁之后，指针依然是存在的。所以在 release 了之后，最好把指针赋为空，防止无头指针的出现。顺便一说，release 一个空指针是合法的，但是不会发生任何事情。

如果你在一个函数中创建并返回一个对象，那么你需要把这个对象声明为 autorelease

```
(ClassA  *)Function()
{
    ClassA *a = [[[ClassA   alloc]  init]  autorelease];
    return a;
}
```

不这样做的话，会造成内存泄露。

## 五、 属性与内存管理

苹果一直没有强调的一点是，关于属性中的 retain。事实上，属性中带有 retain 的，在赋值的时候可能已经在合成的 setter 中 retain 了一次，因此，这里也需要 release。

@property 实际上是 getter 和 setter，@synthesize 是合成这 2 个方法。为什么在声明了属性之后可以用 “.” 来直接调用成员变量呢？那是因为声明属性以后系统根据你给的属性合成

了一个 set 方法和一个 get 方法。使用 “.” 与属性并没有直接关联，如果你不嫌麻烦，在你的程序里面多写一个 set 和 get 方法，你也可以使用 “.” 来调用变量。

@property()，如果你里面什么都不写，那么系统会默认的把你的属性设置为：

@property(atomic, assign)……

关于 nonatomic：

这个属性没有对应的 atomic 关键字，即使我上面是这么写，但 atomic 只是在你没有声明这个特性的时候系统默认，你无法主动去声明这一特性。

如果你的程序只有一个主线程，或者你确定你的程序不会在 2 个或者以上线程运作的时候访问同一个变量，那么你可以声明为 nonatomic。指定 nonatomic 特性，编译器合成访问器的时候不会去考虑线程安全问题。如果你的多个线程在同一时间会访问到这个变量的话，可以将特性声明为 atomic(通过省略关键字 nonatomic)。在这种特性的状态下，编辑器在合成访问器的时候就会在访问器里面加一个锁（@synchronized），在同一时间只能有一个线程访问该变量。

但是使用锁是需要付出代价的，一个声明为 atomic 的属性，在设置和获取这个变量的时候都要比声明为 nonatomic 的慢。所以如果你不打算编写多线程代码，最好把变量的属性特性声明为 nonatomic。

关于 assign、retain 和 copy：

assign 是系统默认的属性特性，它几乎适用于 OC 的所有变量类型。对于非对象类型的变量，assign 是唯一可选的 特性。但是如果你在引用计数下给一个对象类型的变量声明为 assign，那么你会在编译的时候收到一条来自编译器的警告。因为 assign 对于在引用计数下的对象特性，只创建了一个弱引用（也就是平时说的浅复制）。这样使用变量会很危险。当你 release 了前一个对象的时候，被赋值的对象指针就成了无头指针了。因此在为对象类型的变量声明属性的时候，尽量少（或者不要）使用 assign。

关于 assign 合成的 setter，看起来是这样的：

```
-(void)setObjA:(ClassA  *)a
{
    objA  =  a;
}
```

在深入 retain 之前，先把声明为 retain 特性的 setter 写出来：

```
-(void)setObjA:(ClassA  *)a
   {
    If(objA != a)
        {
         [objA  release];
         objA  =  a;
         [objA  retain];  //对象的 retain count 加 1
        }
   }
```

明显的，在 retain 的 setter 中，变量 retain 了一次，那么，即使你在程序中

```
self.objA  =  a;
```

只写了这么一句，objA 仍然需要 release，才能保证对象的 retain count 是正确的。但是如果你的代码

```
objA  =  a；
```

只写了这么一句，那么这里只是进行了一次浅复制，对象的 retain count 并没有增加，因此这样写的话，你不需要在后面 release objA。

这 2 句话的区别是，第一句使用了编译器生成的 setter 来设置 objA 的值，而第二句只是一个简单的指针赋值。

copy 的 setter 看起来是这样的:

```
-(void)setObjA:(ClassA  *)a
    {
     ClassA  * temp  =  objA;
     objA  =  [a   copyWithZone:nil];
     [temp  release];
    }
```

复制必须通过实现 copyWithZone 这个方法 ,因次 copy 这个特性只适用于拥有这个方法的类型，也就是说，必须这个类支持复制。复制是把原来的对象 release 掉，然后让指针指向一个新的对象的副本。因此即使在 setter 里面 release 了原来的对象，你仍然需要在后面 release 新指向的对象（副本）。

## 六、 尾声

IOS 开发现在唯一能用的内存管理方式就是引用计数，无论你喜欢还是不喜欢。在一个内存紧缺的机器上，你编写程序的时候也只能步步为营，尽可能的让你的程序腾出内存空间，并保证系统不会给你一个警告。即使苹果在 Mac OS X 雪豹(v10.5)系统里面添加了另外一种内存管理方式(垃圾收集)，但目前不适用于 IOS。

以上，仅个人对 IOS 开发的内存管理的认知。