iPhone/Mac Objective-C 内存管理教程和原理剖析

Vince Yuan (vince.yuan@gmail.com)

前言

初学 objectice-C 的朋友都有一个困惑，总觉得对 objective-C 的内存管理机制琢磨不透，程序经常内存泄漏或莫名其妙的崩溃。我在这里总结了自己对 objective-C 内存管理机制的研究成果和经验，写了这么一个由浅入深的教程。希望对大家有所帮助，也欢迎大家一起探讨。

此文涉及的内存管理是针对于继承于 NSObject 的 Class。

一 基本原理

Objective-C 的内存管理机制与.Net/Java 那种全自动的垃圾回收机制是不同的，它本质上还是 C 语言中的手动管理方式，只不过稍微加了一些自动方法。

1、Objective-C 的对象生成于堆之上，生成之后，需要一个指针来指向它。

```
ClassA *obj1 = [[ClassA alloc] init];
```

2、Objective-C 的对象在使用完成之后不会自动销毁。需要执行 dealloc 来释放空间（销毁），否则内存泄露。

```
[obj1 dealloc];
```

这带来了一个问题。下面代码中 obj2 是否需要调用 dealloc？

```
ClassA *obj1 = [[ClassA alloc] init];
ClassA *obj2 = obj1;
[obj1 hello];   //输出 hello
[obj1 dealloc];
[obj2 hello];   //能够执行这一行和下一行吗？
[obj2 dealloc];
```

不能，因为 obj1 和 obj2 只是指针，它们指向同一个对象，[obj1 dealloc]已经销毁这个对象了，不能再调用[obj2 hello]和[obj2 dealloc]。obj2 实际上是个无效指针。

如何避免无效指针？请看下一条。

3、Objective-C 采用了引用计数(ref count 或者 retain count)。

对象的内部保存一个数字，表示被引用的次数。例如，某个对象被两个指针所指向（引用）那么它的 retain count 为 2。需要销毁对象的时候，不直接调用 dealloc，而是调用 release。release 会让 retain count 减 1，只有 retain count 等于 0，系统才会调用 dealloc 真正销毁这个对象。

```
ClassA *obj1 = [[ClassA alloc] init];
 //对象生成时，retain count = 1
[obj1 release];
```

```
//release 使 retain count 减 1，retain count = 0，dealloc 自动被调用,对象被销毁
```

我们回头看看刚刚那个无效指针的问题，把 dealloc 改成 release 解决了吗？

```
ClassA *obj1 = [[ClassA alloc] init];       //retain count = 1
ClassA *obj2 = obj1;    //retain count = 1
[obj1 hello];           //输出 hello
[obj1 release];         //retain count = 0，对象被销毁
[obj2 hello];
[obj2 release];
```

[obj1 release]之后，obj2 依然是个无效指针。问题依然没有解决。解决方法见下一条。

4、Objective-C 指针赋值时，retain count 不会自动增加，需要手动 retain。

```
ClassA *obj1 = [[ClassA alloc] init];  //retain count = 1
ClassA *obj2 = obj1;    //retain count = 1
[obj2 retain];          //retain count = 2
[obj1 hello];           //输出 hello
[obj1 release];         //retain count = 2 – 1 = 1
[obj2 hello];           //输出 hello
[obj2 release];         //retain count = 0，对象被销毁
```

问题解决！注意，如果没有调用[obj2 release]，这个对象的 retain count 始终为 1，不会被销毁，内存泄露。

这样的确不会内存泄露，但似乎有点麻烦，有没有简单点的方法？见下一条。

5、Objective-C 中引入了 autorelease pool（自动释放对象池），在遵守一些规则的情况下，可以自动释放对象。

（autorelease pool 依然不是.Net/Java 那种全自动的垃圾回收机制）

5.1 新生成的对象，只要调用 autorelease 就行了，无需再调用 release！

```
ClassA *obj1 = [[[ClassA alloc] init] autorelease];
//retain count = 1 但无需调用 release
```

5.2 对于存在指针赋值的情况，代码与前面类似。

```
ClassA *obj1 = [[[ClassA alloc] init] autorelease];    //retain count = 1
ClassA *obj2 = obj1;        //retain count = 1
[obj2 retain];              //retain count = 2
[obj1 hello];               //输出 hello
//对于 obj1，无需调用（实际上不能调用）release
[obj2 hello];               //输出 hello
[obj2 release];             //retain count = 2-1 = 1
```

细心的读者肯定能发现这个对象没有被销毁，何时销毁呢？谁去销毁它？请看下一条。

6、autorelease pool 原理剖析。

6.1 autorelease pool 不是天生的，需要手动创立。只不过在新建一个 iphone 项目时，xcode 会自动帮你写好。autorelease pool 的真名是 NSAutoreleasePool。

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

6.2 NSAutoreleasePool 内部包含一个数组（NSMutableArray），用来保存声明为 autorelease 的所有对象。如果一个对象声明为 autorelease，系统所做的工作就是把这个对象加入到这个数组中去。

```
ClassA *obj1 = [[[ClassA alloc] init] autorelease];
//retain count = 1，把此对象加入 autorelease pool 中
```

6.3 NSAutoreleasePool 自身在销毁的时候，会遍历一遍这个数组，release 数组中的每个成员。如果此时数组中成员的 retain count 为 1，那么 release 之后，retain count 为 0，对象正式被销毁。如果此时数组中成员的 retain count 大于 1，那么 release 之后，retain count 大于 0，此对象依然没有被销毁，内存泄露。

6.4 默认只有一个 autorelease pool，通常类似于下面这个例子。

```
int main (int argc, const char *argv[])
{
    NSAutoreleasePool *pool;
    pool = [[NSAutoreleasePool alloc] init];

    // do something

     [pool release];
    return (0);
} // main
```

所有标记为 autorelease 的对象都只有在这个 pool 销毁时才被销毁。如果你有大量的对象标记为 autorelease，这显然不能很好的利用内存，在 iphone 这种内存受限的程序中是很容易造成内存不足的。例如：

```
int main (int argc, const char *argv[])
{
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
    int i, j;
    for (i = 0; i < 100; i++ )
    {
        for (j = 0; j < 100000; j++ )
            [NSString stringWithFormat:@"1234567890"];
            //产生的对象是 autorelease 的。
    }
     [pool release];
    return (0);
} // main
```

运行时通过监控工具可以发现使用的内存在急剧增加，直到 pool 销毁时才被释放，你需要考虑下一条。

7、Objective-C 程序中可以嵌套创建多个 autorelease pool。

在需要大量创建局部变量的时候，可以创建内嵌的 autorelease pool 来及时释放内存。

```
int main (int argc, const char *argv[])
{
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
    int i, j;
    for (i = 0; i < 100; i++ )
    {
        NSAutoreleasePool *loopPool = [[NSAutoreleasePool alloc] init];
        for (j = 0; j < 100000; j++ )
            [NSString stringWithFormat:@"1234567890"];
            //产生的对象是 autorelease 的
        [loopPool release];
    }
    [pool release];
    return (0);
} // main
```

运行时通过监控工具可以发现使用占用内存的变化极小。

二 口诀与范式

1、口诀。

1.1 谁创建，谁释放（类似于“谁污染，谁治理”）。如果你通过 alloc、new 或 copy 来创建一个对象，那么你必须调用 release 或 autorelease。换句话说，不是你创建的，就不用你去释放。

例如，你在一个函数中 alloc 生成了一个对象，且这个对象只在这个函数中被使用，那么你必须在这个函数中调用 release 或 autorelease。如果你在一个 class 的某个方法中 alloc 一个成员对象，且没有调用 autorelease，那么你需要在这个类的 dealloc 方法中调用 release；如果调用了 autorelease，那么在 dealloc 方法中什么都不需要做。

1.2 除了 alloc、new 或 copy 之外的方法创建的对象都被声明了 autorelease。

1.3 谁 retain，谁 release。只要你调用了 retain，无论这个对象是如何生成的，你都要调用 release。有时候你的代码中明明没有 retain，可是系统会在默认实现中加入 retain。不知道为什么苹果公司的文档没有强调这个非常重要的一点，请参考范式 2.7 和第三章。

2、范式。

范式就是模板，就是依葫芦画瓢。由于不同人有不同的理解和习惯，我总结的范式不一定适合所有人，但我能保证照着这样做不会出问题。

2.1 创建一个对象。

```
ClassA *obj1 = [[ClassA alloc] init];
```

2.2 创建一个 autorelease 的对象。

```
ClassA *obj1 = [[[ClassA alloc] init] autorelease];
```

2.3 release 一个对象后，立即把指针清空。（顺便说一句，release 一个空指针是合法的，

但不会发生任何事情）

```
[obj1 release];
obj1 = nil;
```

2.4 指针赋值给另一个指针。

```
ClassA *obj2 = obj1;
[obj2 retain];
//do something
[obj2 release];
obj2 = nil;
```

2.5 在一个函数中创建并返回对象，需要把这个对象设置为 autorelease。

```
ClassA *Func1()
{
   ClassA *obj = [[[ClassA alloc]init]autorelease];
   return obj;
}
```

2.6 在子类的 dealloc 方法中调用基类的 dealloc 方法。

```
-(void) dealloc
{
    …
    [super dealloc];
}
```

2.7 在一个 class 中创建和使用 property。

2.7.1 声明一个成员变量。

```
ClassB *objB;
```

2.7.2 声明 property，加上 retain 参数。

```
@property (retain) ClassB* objB;
```

2.7.3 定义 property。（property 的默认实现请看第三章）

```
@synthesize objB;
```

2.7.4 除了 dealloc 方法以外，始终用.操作符的方式来调用 property。

```
self.objB 或者 objA.objB
```

2.7.5 在 dealloc 方法中 release 这个成员变量。

```
[objB release];
```

示例代码如下，你需要特别留意对象是在何时被销毁的。

```
@interface ClassA : NSObject
{
    ClassB* objB;
}
@property (retain) ClassB* objB;
@end
```

```
@implementation ClassA
@synthesize objB;
-(void) dealloc
{
    [objB release];
    [super dealloc];
}
@end
```

2.7.6 给这个 property 赋值时，有手动 release 和 autorelease 两种方式。

```
void funcNoAutorelease()
{
        ClassB *objB1 = [[ClassB alloc]init];
        ClassA *objA = [[ClassA alloc]init];
        objA.objB = objB1;
        [objB1 release];
        [objA release];
}
void funcAutorelease()
{
        ClassB *objB1 = [[[ClassB alloc]init] autorelease];
        ClassA *objA = [[[ClassA alloc]init] autorelease];
        objA.objB = objB1;
}
```

三 @property (retain)和@synthesize 的默认实现

在这里解释一下@property (retain) ClassB* objB;和@synthesize objB;背后到底发生了什么(retain property 的默认实现)。property 实际上是 getter 和 setter，针对有 retain 参数的 property，背后的实现如下：

```
@interface ClassA : NSObject
{
    ClassB *objB;
}
-(ClassB *) getObjB;
-(void) setObjB:(ClassB *) value;
@end
```

```
@implementation ClassA
-(ClassB*) getObjB
{
    return objB;
}
```

```
-(void) setObjB:(ClassB*) value
{
    if (objB != value)
    {
         [objB release];
         objB = [value retain];
    }
}
```

在 setObjB 中，如果新设定的值和原值不同的话，必须要把原值对象 release 一次，这样才能保证 retain count 是正确的。

由于我们在 class 内部 retain 了一次（虽然是默认实现的），所以我们要在 dealloc 方法中 release 这个成员变量。

```
-(void) dealloc
{
    [objB release];
    [super dealloc];
}
```

四 系统自动创建新的 autorelease pool

在生成新的 Run Loop 的时候，系统会自动创建新的 autorelease pool。注意，此处不同于 xcode 在新建项目时自动生成的代码中加入的 autorelease pool，xcode 生成的代码可以被删除，但系统自动创建的新的 autorelease pool 是无法删除的（对于无 Garbage Collection 的环境来说）。Objective-C 没有给出实现代码，官方文档也没有说明，但我们可以通过小程序来证明。

在这个小程序中，我们先生成了一个 autorelease pool，然后生成一个 autorelease 的 ClassA 的实例，再在一个新的 run loop 中生成一个 autorelease 的 ClassB 的对象（注意，我们并没有手动在新 run loop 中生成 autorelease pool）。

```
int main(int argc, char**argv)
{
    NSLog(@"create an autorelasePool\n");
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];

    NSLog(@"create an instance of ClassA and autorelease\n");
    ClassA *obj1 = [[[ClassA alloc] init] autorelease];
    NSDate *now = [[NSDate alloc] init];
    NSTimer *timer = [[NSTimer alloc] initWithFireDate:now
                         interval:0.0
                         target:obj1
                         selector:@selector(createClassB)
                         userInfo:nil
```

```
                            repeats:NO];
    NSRunLoop *runLoop = [NSRunLoop currentRunLoop];
    [runLoop addTimer:timer forMode:NSDefaultRunLoopMode];
    [timer release];
    [now release];
    [runLoop run]; //在新 loop 中调用一函数，生成 ClassB 的 autorelease 实例

    NSLog(@"releasing autorelasePool\n");
    [pool release];
    NSLog(@"autorelasePool is released\n");
    return 0;
}
```

输出如下：

```
create an autorelasePool
create an instance of ClassA and autorelease
create an instance of ClassB and autorelease
ClassB destroyed
releasing autorelasePool
ClassA destroyed
autorelasePool is released
```

注意在我们销毁 autorelease pool 之前，ClassB 的 autorelease 实例就已经被销毁了。

有人可能会说，这并不能说明新的 run loop 自动生成了一个新的 autorelease pool，说不定还只是用了老的 autorelease pool，只不过后来 drain 了一次而已。我们可以在 main 函数中不生成 autorelease pool。

```
int main(int argc, char**argv)
{
    NSLog(@"No autorelasePool created\n");
    NSLog(@"create an instance of ClassA\n");
    ClassA *obj1 = [[ClassA alloc] init];
    NSDate *now = [[NSDate alloc] init];
    NSTimer *timer = [[NSTimer alloc] initWithFireDate:now
                interval:0.0
                target:obj1
                selector:@selector(createClassB)
                userInfo:nil
                repeats:NO];
    NSRunLoop *runLoop = [NSRunLoop currentRunLoop];
    [runLoop addTimer:timer forMode:NSDefaultRunLoopMode];
    [timer release];
    [now release];
    [runLoop run]; //在新 loop 中调用一函数，生成 ClassB 的 autorelease 实例
    NSLog(@"Manually release the instance of ClassA\n");
```

```
        [obj1 release];
        return 0;
    }
```

输出如下：

```
No autorelasePool created
create an instance of ClassA
create an instance of ClassB and autorelease
ClassB destroyed
Manually release the instance of ClassA
ClassA destroyed
```

我们可以看出来，我们并没有创建任何 autorelease pool，可是 ClassB 的实例依然被自动销毁了，这说明新的 run loop 自动创建了一个 autorelease pool，这个 pool 在新的 run loop 结束的时候会销毁自己（并自动 release 所包含的对象）。

补充说明

在研究 retain count 的时候，我不建议用 NSString。因为在下面的语句中，

NSString *str1 = @”constant string”;

str1 的 retain count 是个很大的数字。Objective-C 对常量字符串做了特殊处理。

当然，如果你这样创建 NSString，得到的 retain count 依然为 1

NSString *str2 = [NSString stringWithFormat:@”123”];