**内存管理**

**类扩展和分类的区别**

类扩展 （Class Extension也有人称为匿名分类）##

能为某个类附加额外的属性，成员变量，方法声明

一般的类扩展写到.m文件中

一般的私有属性写到类扩展

@interface Mitchell()

//属性

//方法

@end

**分类**

@interface 类名（分类名字）

/方法声明/

@end

@implementation类名（分类名字）

/方法实现/

@end

分类只能扩充方法，不能扩展属性和成员变量（如果包含成员变量会直接报错）。

- 如果分类中声明了一个属性，那么分类只会生成这个属性的set、get方法声明，也就是不会有实现。

无论父类是在@interface还是@implementation声明的成员变量子类都能拥有；但是子类能不能直接通过变量名来访问父类中定义的成员变量是需要看父类中定义的成员变量是由什么修饰符来修饰的。

无论在MRC还是ARC情况下，Objective-C采用的是引用计数式的内存管理方式

特点:

* 自己生成的对象，自己持有。例如：NSObject \* \_\_strong obj = [[NSObject alloc]init];。
* 非自己生成的对象，自己也能持有。例如：NSMutableArray \* \_\_strong array = [NSMutableArray array];。
* 不再需要自己持有对象时释放。
* 无法释放非自己持有的对象。

生成并持有对象 alloc/new/copy/mutableCopy等方法

持有对象 retain方法

释放对象 release方法

废弃对象 dealloc方法

**自己生成的对象，自己持有**

在iOS内存管理中有四个关键字，alloc、new、copy、mutableCopy，自身使用这些关键字产生对象,那么自身就持有了对象

// 使用了alloc分配了内存，obj指向了对象，该对象本身引用计数为1,不需要retain

id obj = [[NSObject alloc] init];

// 使用了new分配了内存,objc指向了对象，该对象本身引用计数为1，不需要retain

id obj = [NSObject new];

**非自己生成的对象，自己也能持有**

// NSMutableArray通过类方法array产生了对象(并没有使用alloc、new、copy、mutableCopt来产生对象),因此该对象不属于obj自身产生的

// 因此，需要使用retain方法让对象计数器+1,从而obj可以持有该对象(尽管该对象不是他产生的)

id obj = [NSMutableArray array];

[obj retain];

autorelease

顾名思义：autorelease就是自动释放，它和C语言的局部变量类似，C语言局部变量在程序执行时，超出其作用域时，会被自动废弃。autorelease会像C语言局部变量那样对待对象实例，当超出作用域时，对象实例的release实例方法会被调用。我们可以设定autorelease的作用域。

autorelease具体使用方法

生成并持有NSAutoreleasePool对象

调用已分配对象的autorelease实例方法

废弃NSAutoreleasePool对象

**所有权修饰符及其原理**

在 ARC 特性下有 4 种与内存管理息息相关的变量所有权修饰符值得我们关注：

* \_\_strong
* \_\_weak
* \_\_unsafe\_unretaied
* \_\_autoreleasing

说到变量所有权修饰符，有人可能会跟属性修饰符搞混，这里做一个对照关系小结：

* assign 对应的所有权类型是 \_\_unsafe\_unretained。
* copy 对应的所有权类型是 \_\_strong。
* retain 对应的所有权类型是 \_\_strong。
* strong 对应的所有权类型是 \_\_strong。
* unsafe\_unretained对应的所有权类型是\_\_unsafe\_unretained。
* weak 对应的所有权类型是 \_\_weak。

**\_\_strong**

\_\_strong 表示强引用，对应定义 property 时用到的 strong。当对象没有任何一个强引用指向它时，它才会被释放。如果在声明引用时不加修饰符，那么引用将默认是强引用, \_\_strong 修饰符是 id 类型和对象类型默认的所有权修饰符

**\_\_weak**

\_\_weak 表示弱引用，对应定义 property 时用到的 weak。弱引用不会影响对象的释放，而当对象被释放时，所有指向它的弱引用都会自定被置为 nil，这样可以防止野指针

\_\_weak 的几个使用场景：

在 Delegate 关系中防止循环引用。

在 Block 中防止循环引用。

用来修饰指向由 Interface Builder 创建的控件。比如：@property (weak, nonatomic) IBOutlet UIButton \*testButton;。

**为什么@property声明（NString，NSArray，NSDictionary)时需要使用copy，使用strong有什么问题。**

因为NString，NSArray，NSDictionary都有自己对应的子类：NSMutableString，NSMutableArray，NSMutableDictionary，而父类指针可以指向子类对象，使用copy可以让本对象不受外界（子对象）影响，无论给我传入的是一个可变对象还是一个不可变对象，都能保证自身持有的是一个不可变副本。

使用strong时，如果这个属性指向一个可变对象，修改可变对象时，这个属性值也会被修改。使用copy能保证属性不被子类对象修改时同时被修改。

**深拷贝和浅拷贝**

深拷贝：内容的拷贝

浅拷贝：地址的拷贝

**集合对象的copy和mutableCopy和非集合对象相同的结果：**

[immutableObject copy]是浅拷贝

[immutableObject mutableCopy]是深拷贝

[mutableObject copy]是深拷贝

[mutableObject mutableCopy]是深拷贝

**block为什么要使用copy**

block使用copy是在MRC中延续下来的，在MRC下，方法内部的block是存放在栈区，使用copy会将block拷贝到堆区。

在ARC下编译器会自动对block进行copy，因此我们使用copy或者strong的效果是一样的。但是我们在ARC下继续使用copy可以提醒我们编译器会自动帮我们实现copy的操作

集合对象的深拷贝只是对象本身，而集合对象的元素还是地址的拷贝，即单层深拷贝：

僵尸对象：一个OC对象引用计数为0被释放后就变成僵尸对象了，僵尸对象的内存已经被系统回收，虽然可能该对象还存在，数据依然在内存中，但僵尸对象已经是不稳定对象了，不可以再访问或者使用，它的内存是随时可能被别的对象申请而占用的；

野指针：野指针出现的原因是指针没有赋值，或者指针指向的对象已经被释放掉了，野指针指向一块随机的垃圾内存，向他们发送消息会报EXC\_BAD\_ACCESS错误导致程序崩溃；

空指针：空指针不同于野指针，它是一个没有指向任何东西的指针，空指针是有效指针，值为nil、NULL、Nil或0等，给空指针发送消息不会报错，只是不响应消息而已，应该给野指针及时赋予零值变成有效的空指针，避免内存报错

OC是通过RC引用计数，通过引用计数的方式来管理内存的分配与释放，从而防止内存泄漏。

引用计数是局部性的，开发者要管理控制每个对象的引用计数，单个对象引用计数为0后会马上被释放掉

**Objective-C是如何实现内存管理的？autorealease pool自动释放池是什么？autorelease的对象是在什么时候被release的？autorelease和release有什么区别？**

Objective-C的内存管理本质上是通过引用计数实现的，每次RunLoop都会检查对象的引用计数，如果引用计数为0，说明该对象已经没人用了，可以对其进行释放了。其中引用计数可以大体分为三种：MRC(手动内存计数)、ARC(自动内存计数，iOS5以后)和内存池。

其中引用计数是如何操作的呢？不论哪种引用计数方式，本质都是在合适的时机将对象的引用计数加1或者减1。

这里简单归结一下：

使对象引用计数加1的常见操作有：alloc、copy、retain

使对象引用计数减1的常见操作有：release、autorealease

dealloc是在对象引用计数为0以后系统自动调用的，dealloc没有使对象引用计数减1的作用，只是在对象引用计数为0后被系统调用进行内存回收的收尾工作。

Objective-c对象的保存在堆之上,指向它的指针保存在栈之上;

Iphone的内存管理方法采用保留计数retainCount机制;当一个对象被创建时,保留计数为1;每增加一个所有者则对象的保留计数增加1;同时,当对象被release,保留计数减1,失去一个所有者.当保留计数为0时,调用dealloc方法,对象被销毁;

引用计数的主要接口;

1,alloc,allocWithZone,new(带初始化)为对象分配内存,retainCount为‘1’,并返回此实例;

2,retain

retainCount 加1;

3,copy,mutableCopy 复制一个实例,retainCount数为1,返回此实例;所得到的对象是与它上下文无关,独立的对象;

4,release,引用计数减1,当为0时,调用dealloc方法;

5,autorelease,延时释放

NSAutoreleasePool \* pool = [[NSAutorelease alloc] init];

NSObject \* o = [[NSObject alloc] init]; //保留计数为1

[o retain];//保留计数为2

[o release];//保留计数为1

[o autorelease];//保留计数为1

[pool release]; //保留计数为0,调用dealloc

NSAutoreleasePool 内部包含一个数组,用来保存声明为autorelease的所有对象, 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,如果此时数组中的成员的retainCount为1,那么release之后,retainCount为0,对象正式销毁,如果此时数组中成员的retainCount大于1,那么release之后,retainCount大于0,此对象依然没有被销毁,内存泄露;