1、\_\_weak的功能和作用

**(1)\_\_weak类型的指针指向的对象被释放时，该指针会自动被置成nil，因此\_\_weak关键字修饰的指针又被称为智能指针。那么这个功能是如何实现的呢？**

id \_\_weak obj1 = obj;

会转化为

id obj1;

objc\_initWeak(&obj1, obj);

objc\_destoryWeak(&obj1);

即编译器会通过objc\_initWeak函数初始化\_\_weak修饰的变量，当变量的作用域结束后会通过objc\_destoryWeak函数释放该变量。objc\_initWeak函数实际干的活是：

objc1 = 0;

objc\_storeWeak(&obj1, obj);

这里是先将指针objc1置成0，再调用objc\_storeWeak函数使得obj1指向obj对象。  
接下来的objc\_destoryWeak函数的实际操作如下：

objc\_storeWeak(&obj1, 0);

也就是说，让obj1指针指向的内容变成空。

**\_\_weak实现原理**

实际上，objc\_storeWeak函数会把第二个参数的对象的地址作为key，并将第一个参数（\_\_weak关键字修饰的指针的地址）作为值，注册到weak表中。如果第二个参数为0（说明对应的对象被释放了），则将weak表中将整个key-value键值对删除，这就是\_\_weak关键字的核心思想！

weak表和引用计数表类似，都是通过hash表实现的。如果使用weak表，将被释放的对象地址作为key去检索，就能很高效的获取对应的指向该对象的类型为\_\_weak的指针变量的地址。同时很容易理解，一个对象可能有多个\_\_weak指针指向，因此一个对象地址key可能对应多个值。

在调用对象的release方法时，会在其中一步调用objc\_clear\_deallocating函数，该函数会执行以下操作：以当前对象的地址作为key，从weak表中获取对应的值----指向该对象的\_\_weak类型的指针变量；将取到的所有指针变量的值赋值为nil；从weak表中删除该key对应的整条记录。

如果大量使用附有\_\_weak修饰符的变量会消耗响应的CPU资源，因此，应该尽量少使用\_\_weak修饰符。

二、\_\_block和\_\_weak 的本质区别

**block下循环引用的问题**

\_\_block本身并不能避免循环引用，避免循环引用需要在block内部把\_\_block修饰的obj置为nil

\_\_weak可以避免循环引用，但是其会导致外部对象释放了之后，block 内部也访问不到这个对象的问题，我们可以通过在 block 内部声明一个 \_\_strong  
的变量来指向 weakObj，使外部对象既能在 block 内部保持住，又能避免循环引用的问题

**\_\_block与\_\_weak功能上的区别。**

\_\_block会持有该对象，即使超出了该对象的作用域，该对象还是会存在的，直到block对象从堆上销毁；而\_\_weak仅仅是将该对象赋值给weak对象，当该对象销毁时，weak对象将指向nil；

\_\_block可以让block修改局部变量，而\_\_weak不能。

**另外，MRC中\_\_block是不会引起retain；但在ARC中\_\_block则会引起retain。所以ARC中应该使用\_\_weak。**

三、 assign，weak，strong，copy

**1.assign 与weak区别**

assign适用于基本数据类型，weak是适用于NSObject对象，并且是一个弱引用。

assign其实也可以用来修饰对象。那么我们为什么不用它修饰对象呢？因为被assign修饰的对象（一般编译的时候会产生警告：Assigning retained object to unsafe property; object will be released after assignment）在释放之后，指针的地址还是存在的，也就是说指针并没有被置为nil，造成野指针。对象一般分配在堆上的某块内存，如果在后续的内存分配中，刚好分到了这块地址，程序就会崩溃掉。

那为什么可以用assign修饰基本数据类型？因为基础数据类型一般分配在栈上，栈的内存会由系统自己自动处理，不会造成野指针。

weak修饰的对象在释放之后，指针地址会被置为nil。所以现在一般弱引用就是用weak。weak使用场景：

在ARC下,在有可能出现循环引用的时候，往往要通过让其中一端使用weak来解决，比如: delegate代理属性，通常就会声明为weak。

自身已经对它进行一次强引用，没有必要再强引用一次时也会使用weak。比如：自定义 IBOutlet控件属性一般也使用weak，当然也可以使用strong。

**2.strong 与copy的区别**

strong 与copy都会使引用计数加1，但strong是两个指针指向同一个内存地址，copy会在内存里拷贝一份对象，两个指针指向不同的内存地址

**3.\_\_block与\_\_weak的区别**

\_\_block是用来修饰一个变量，这个变量就可以在block中被修改

\_\_block：使用 \_\_block修饰的变量在block代码块中会被retain（ARC下会retain，MRC下不会retain）

\_\_weak：使用\_\_weak修饰的变量不会在block代码块中被retain

同时，在ARC下，要避免block出现循环引用 \_\_weak typedof(self)weakSelf = self;

**4、原子性（nonatomic、atomic）**

**atomic**（默认）：atomic意为操作是原子的，意味着只有一个线程访问实例变量。atomic是线程安全的，至少在当前的存取器上是安全的。它是一个默认的特性，但是很少使用，因为比较影响效率，这跟ARM平台和内部锁机制有关。

当一个变量声明为atomic时，意味着在多线程中只能有一个线程能对它进行访问

当一个变量声明为atomic时，该变量为线程安全型，但是会影响访问速度，

当一个变量声明为atomic时，在非ARC编译环境下，需要设置访问锁来保证对该变量进行正确的getter/setter。

**nonatomic(常用)**：nonatomic跟atomic刚好相反。表示非原子的，可以被多个线程访问。它的效率比atomic快，但不能保证在多线程环境下的安全性，在单线程和明确只有一个线程访问的情况下广泛使用。

当一个变量声明为nonatomic时，意味着多个线程可以同时对其进行访问

当一个变量声明为nonatomic时，它是非线程安全型，访问速度快；

当一个变量声明为nonatomic时，当两个不同的线程对其访问时，容易失控。

**5、存取方法（readonly、readwrite）**

**readwrite（默认）：readwrite是默认值，表示该属性同时拥有setter和getter。**

**readonly：** readonly表示只有getter没有setter。

有时，在声明属性时，可以指定存取方法的自定义名称，通常在Bool类型的属性的getter方法会采用这种方式。

**4. block变量定义时为什么用copy？block是放在哪里的？**

block的循环引用并不是strong导致的…在ARC环境下，系统底层也会做一次copy操作使block从栈区复制一块内存空间到堆区…所以strong和copy在对block的修饰上是没有本质区别的，只不过copy操作效率高而已

typeof意义：

**typeof 是一个一元运算，放在一个运算数之前，运算数可以是任意类型。**

**它返回值是一个字符串，该字符串说明运算数的类型。对于对象、数组、null 返回的值object 。比如typeof(window)，typeof(document)，typeof(null)返回的值都是object。**

关于Block内部要不要使用weakSelf的几种情况

http://www.jianshu.com/p/c6ca540861d9  
1、block是控制器的属性，如果block内部没有使用weakSelf将会造成内存泄露

2、把block内部抽出一个作为self的方法，当使用weakSelf调用这个方法，并且这个方法里有self的属性， block不会造成内存泄露  
3、当block不是self的属性时，block内部使用self也不会造成内存泄露  
4、当使用类方法有block作为参数使用时，block内部使用self也不会造成内存泄露

建议：在block内如果需要访问self的方法，变量建议使用weakSelf，在block内部需要多次访问self,则需要使用strongSelf

block使用的5点注意事项

（来自网址：http://www.jianshu.com/p/db1c7c3c8cd9）

1、在使用block前需要对block指针做判空处理。

不判空直接使用，一旦指针为空直接产生崩溃。

2、在MRC的编译环境下，block如果作为成员参数要copy一下将栈上的block拷贝到堆上

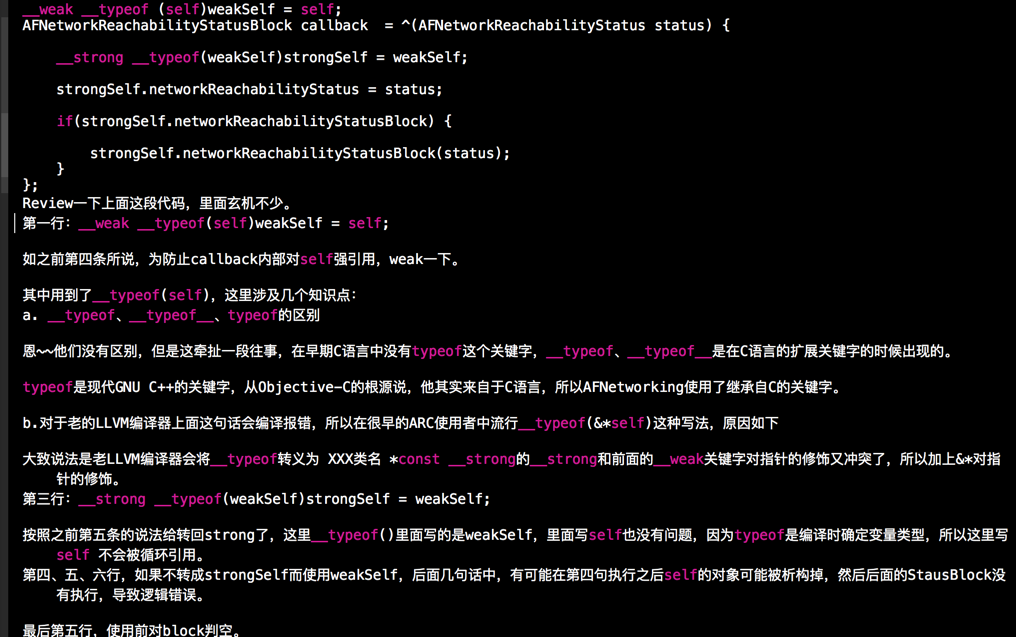
3、在block使用之后要对，block指针做赋空值处理，如果是MRC的编译环境下，要先release掉block对象。

block作为类对象的成员变量，使用block的人有可能用类对象参与block中的运算而产生循环引用。

将block赋值为空，是解掉循环引用的重要方法。（不能只在dealloc里面做赋空值操作，这样已经产生的循环引用不会被破坏掉）

4、使用方将self或成员变量加入block之前要先将self变为\_\_weak

5、在多线程环境下（block中的weakSelf有可能被析构的情况下），需要先将self转为strong指针，避免在运行到某个关键步骤时self对象被析构。

第四、第五条合起来有个名词叫weak–strong dance 堪称使用weak–strong dance的经典。

深究block的实现<http://www.cocoachina.com/ios/20160307/15441.html>

iOS中的栈区和堆区：http://www.jianshu.com/p/f3c1b920e8eb

http://www.jianshu.com/p/bb20a0feffb0