Strong与weak的用法讲解

首先strong和weak这两个关键字是用来修饰变量，表示这个变量是强(strong)引用和弱(weak)引用

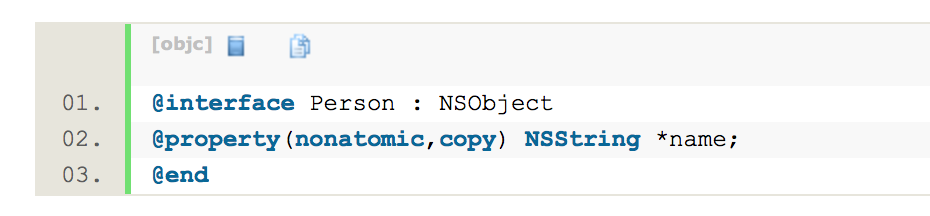
我们在程序中经常会用到“[[类 alloc ] init]” 这样的代码，我想你对它已经很熟。这是在开辟一块内存，并初始化。那么系统开辟了这块内存，我们怎么去拿到它呢？

显然是将刚分配好的内存赋值到一个变量，以后我们就可以利用这个变量直接操作这块内存了。那么把刚分配的内存赋值给一个strong变量和weak变量是有区别的：

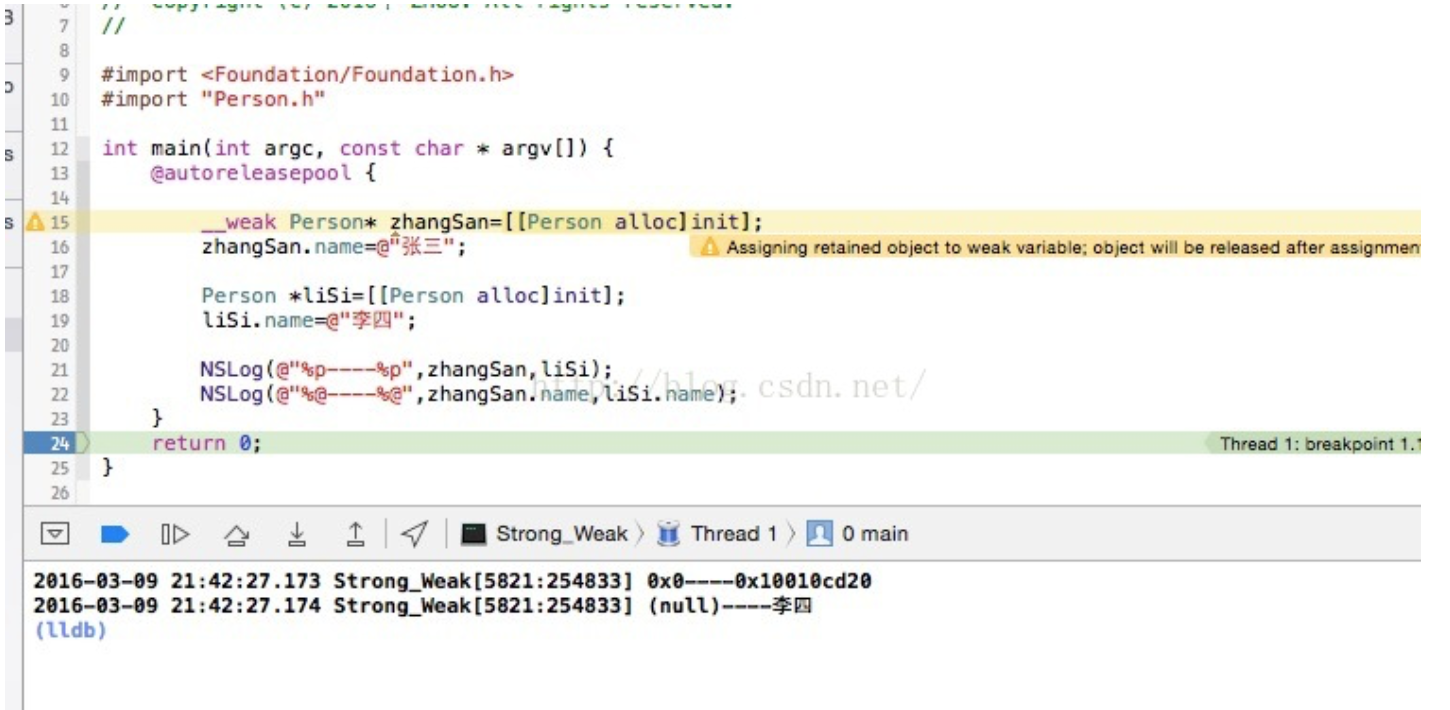
赋值给weak变量后这块内存会马上被释放。而分配给strong变量的会等到这个变量的生命周期结束后，这块内存才被释放(不用关键字weak修饰的变量默认为strong变量)。

看下面的例子：

添加一个Person类，只有一个name属性（例子在代码中）



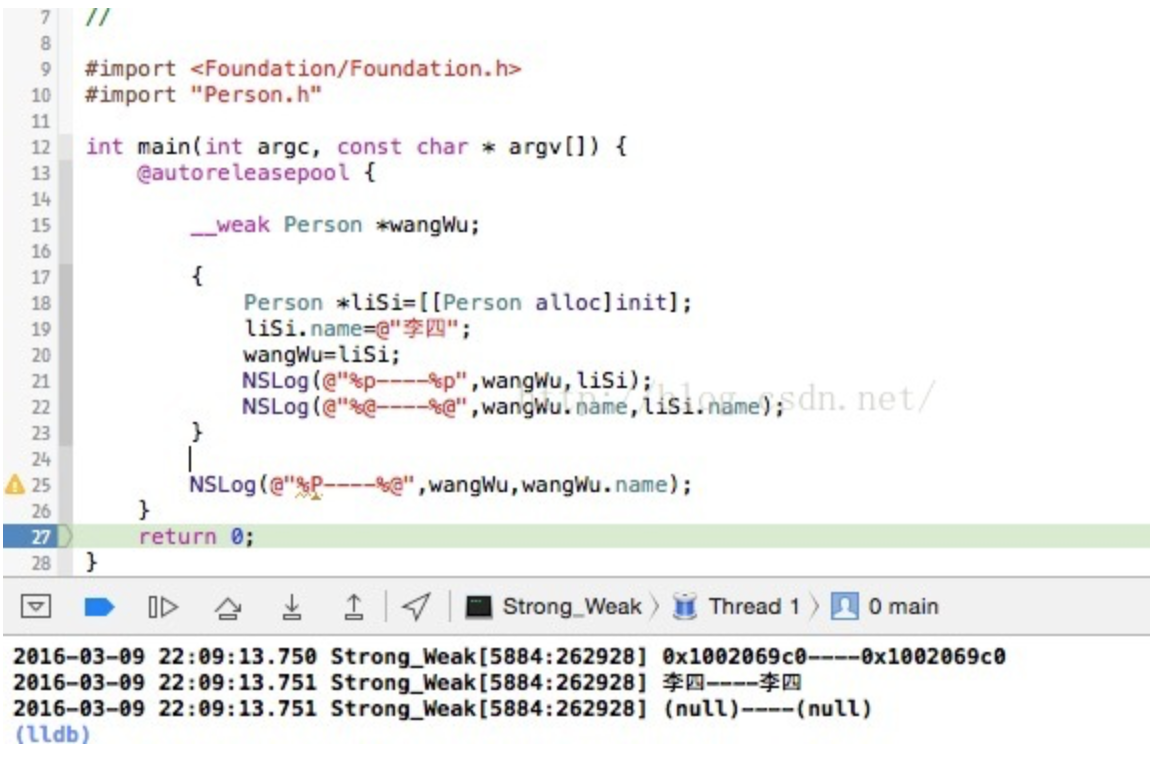
main函数中定义了一个weak的zhangSan和一个strong的李四，很明显zhangSan指定的内存在“zhangSan=[[Person alloc]init]”执行后就立即被释放了。我们分别打印出两个变量的地址和name属性，可以看到zhangSan的确被释放了,而liSi一直到程序的结尾。



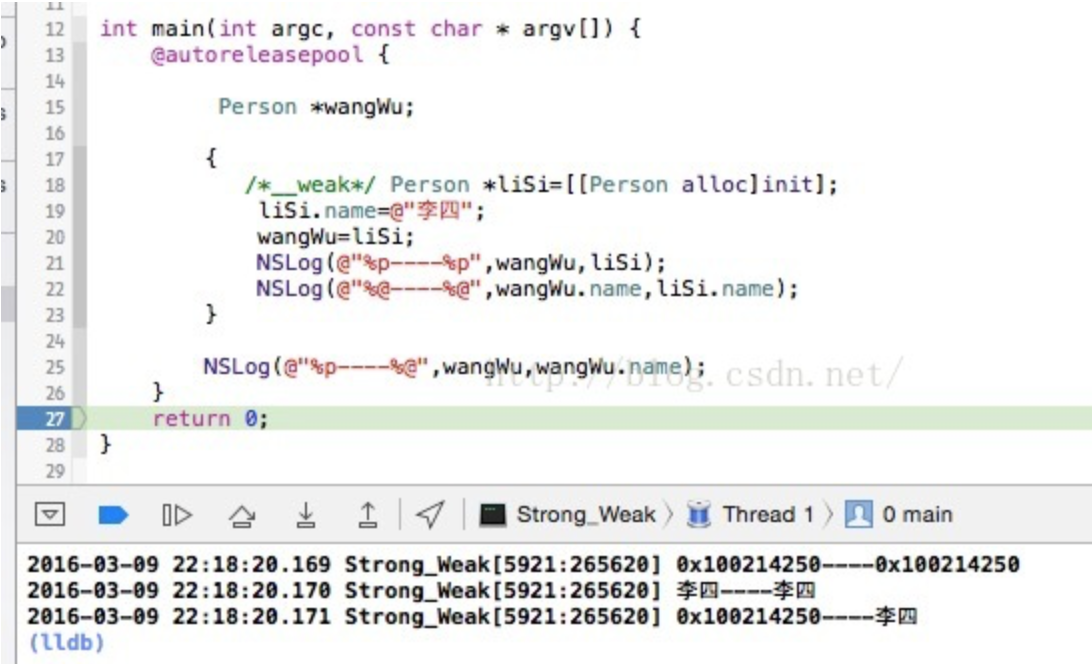
我们可以这样理解，分配出来的内存像一头牛，得用一条结实、强壮（strong）的绳子才能把它牵住，用纤细、弱小（weak）的绳子的话，这头牛随时会把绳子挣断逃脱。

而绳子的另一端是被固定到我们能够看得见够得着的物体（就是我们的变量）上面，我们顺着这个物体上面的绳子摸索过去，你的那头牛还在不在就看你用的上面绳子了。

那既然weak类型的变量内存分配出来就被释放了，它还有什么用呢？我们再看下面的例子：



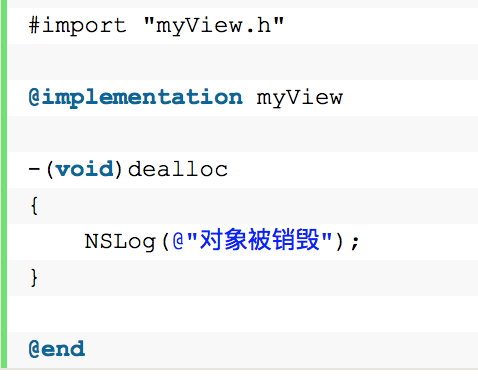
“liSi”声明在一对大括号内，表明它只在大括号内有效，除了大括号，这个拴绳子的物体就不在了，那么牛自然挣脱弱小绳子的束缚跑掉，所以你再拿到“wangWu”想找到那头牛（内存及内存中的值）就不可能了。那如果有多条结实的绳子拴住牛，想必你也知道是怎么回事了。



用两条结实的绳子拴住一头牛,即使一根不在了，利用另一根还是能找到这头牛。

说到这里其实也没有说到weak类型的变量这种机制到底有什么用,我们看下面的例子

1.自定义一个VIew继承自UIView,重写dealloc方法，查看对象什么时候被销毁



2.我们在程序调用主视图的时候，将自定义视图添加到主视图上，并且添加按钮，监听点击事件。





我想要的效果是：点击按钮移除 子视图 ，并且程序以后的运行永远也不会用到这个view。

这里我想到有两种方式拿到这个view，然后从父控件中移除它。

第一种：这种方式显然能实现这样的要求，我们能看到打印结果，在子视图被移除父控制器之后对象也被销毁了。然而这不是我们最常用的方式，有可能父控件上有很多子视图，这样效率很低，而且代码不简洁。



第二种：我们给控制器增加一个属性，指向我们的子视图。这个属性有两种可能，一种是strong，一种是weak。我们先来试试strong。

从结果可以看出，点击按钮子视图是移除了，但是对象没有被销毁。它仍然在内存中（你可以再添加一个按钮打印一下self.testView试试），这不是我们想要的效果。



 我们在来试试weak,只需要将声明变量的地方的strong改为weak即可，其他地方不变。从运行结果可以看出:子视图被移除了，且变量被销毁了。



为什么我们没有添加strong属性的时候分配出来的内存没有被释放，仍然能通过for循环找到它？需要注意的是：当一个视图A被添加到另一个视图B时，A就被B的subViews强引用了（有一个结实的绳子拉着它了），所以我们再用一个强属性去拉着它的话，自然要两条绳子都断了，它才会被释放。