**知识点一：@property中的特性**

**大概可以分为三类：1）原子性2）存取器控制3）内存管理**

* **原子性：**
* **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。
* **nonatomic：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。**
* **存取器控制**
* **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。**
* **自定义访问器的名字：@property (nonatomic,getter = isHidden ) BOOL hidden;**
* **内存管理**
* **assign（默认）：assign用于值类型，如int、float、double和NSInteger，CGFloat等表示单纯的复制。还包括不存在所有权关系的对象，比如常见的delegate。**
* **retain：在setter方法中，需要对传入的对象进行引用计数加1的操作。简单来说，就是对传入的对象拥有所有权，只要对该对象拥有所有权，该对象就不会被释放。**
* **strong是在IOS引入ARC的时候引入的关键字，是retain的一个可选的替代。表示实例变量对传入的对象要有所有权关系，即强引用。strong跟retain的意思相同并产生相同的代码，但是语意上更好更能体现对象的关系。**
* **在setter方法中，需要对传入的对象不进行引用计数加1的操作。**
* **简单来说，就是对传入的对象没有所有权，当该对象引用计数为0时，即该对象被释放后，用weak声明的实例变量指向nil，即实例变量的值为0。**
* **copy：与strong类似，但区别在于实例变量是对传入对象的副本拥有所有权，而非对象本身**

**@synthesized 和@dynamic的却别**

**@dynamic表示不由系统自己生成getter和setter**

**知识点二：category的使用**

* **使用场景**
* 需求变更在整个开发周期是司空见惯的事情，那么我们可能就需要

对某个或某几个类中添加新的方法以满足需求。

* 我们在团队协作开发时候，经常需要多个人来实现一个类中的不同

方法，在这种情况下采用Category是一个较好的选择。

* 当一些基础类库满足不了我们的需求时，我们希望能扩展基础类库，

这时就需要Category。

* **需要注意的问题**
* Category可以访问原始类的实例变量，但不能添加变量，如果想添加变

量，可以考虑通过继承创建子类。

* Category可以重载原始类的方法，但不推荐这么做，这么做的后果是你

再也不能访问原来的方法。如果确实要重载，正确的选择是创建子类。

* 和普通接口有所区别的是，在分类的实现文件中可以不必实现所有

声明的方法，只要你不去调用它。

* **总结**

掌握并用好Category可以充分利用Objective-C的动态特性，编写出灵活简洁的代码。

**知识点三：内存管理**

**在OC中，只有对象才属于内存管理范围，例如int、struce等基本数据类型不存在内存管理的概念。在iOS开发中，对内存的管理实际上就是对引用计数器的管理。**

* **OC内存管理的三种方式**
* **自动垃圾收集(Automatic Garbage Collection)；**
* **手动引用计数器(Manual Reference Counting)和自动释放池；**
* **自动引用计数器(Automatic Reference Counting)。**
* **自动垃圾收集**
* **在OC2.0中，有一种自动垃圾收集的内存管理形式，通过垃圾自动收集，系统能够自动检测出对象是否拥有其他的对象，当程序运行期间，不被引用的对象就会自动释放。**
* **说明：在iOS运行环境中不支持自动垃圾收集，在OS X环境才支持，但是Apple现在不建议使用该方法，而是推荐使用ARC进行替代。**
* **手动引用计数器(MRC)**

**MRC即我们通过人为的方式来控制引用计数器的增减，影响对象RC值得方法有以下几种：**

* **new、alloc、copy、mutableCopy，这几种方法用来创建一个新的对象并且获得对象的所有权，此时RC的值默认为RC=1；**
* **retain,对象调用retain方法，该对象的RC+1;**
* **release，对象调用 release方法，该对象的RC-1;**
* **dealloc，dealloc方法并不会影响RC的值，但是当RC的值为0时，系统会调用dealloc方法来销毁对象。**

关于在MRC中@property关键字如下：

**1. assign和retain和copy**

**这几个关键字用于setter方法的内存管理，如果使用assign（一般用于非OC对象），那么将直接执行赋值操作；如果使用retain（一般用于OC对象）,那么将retain新值，release旧值；如果使用copy，那么将release旧值，copy新值。不显示使用assign为默认值。**

**2. nonatomic和atomic**

**这两个关键字用于多线程管理，nonatomic的性能高，atomic的性能低。不显示使用atomic为默认值。**

**3.readwrite和readonly**

**这两个关键字用于说明是否生成setter方法，readwrite将自动生成setter和getter方法，readonly 只生成getter方法。不显示使用readwrite为默认值。**

**4. getter和setter**

**这两个关键字用于给设值和取值方法另外起一个名字。例如@property(getter=a,setter=b:) int age;相当于取值方法名为a，设值方法名为b:。**

* **Autorelease Pool的使用**

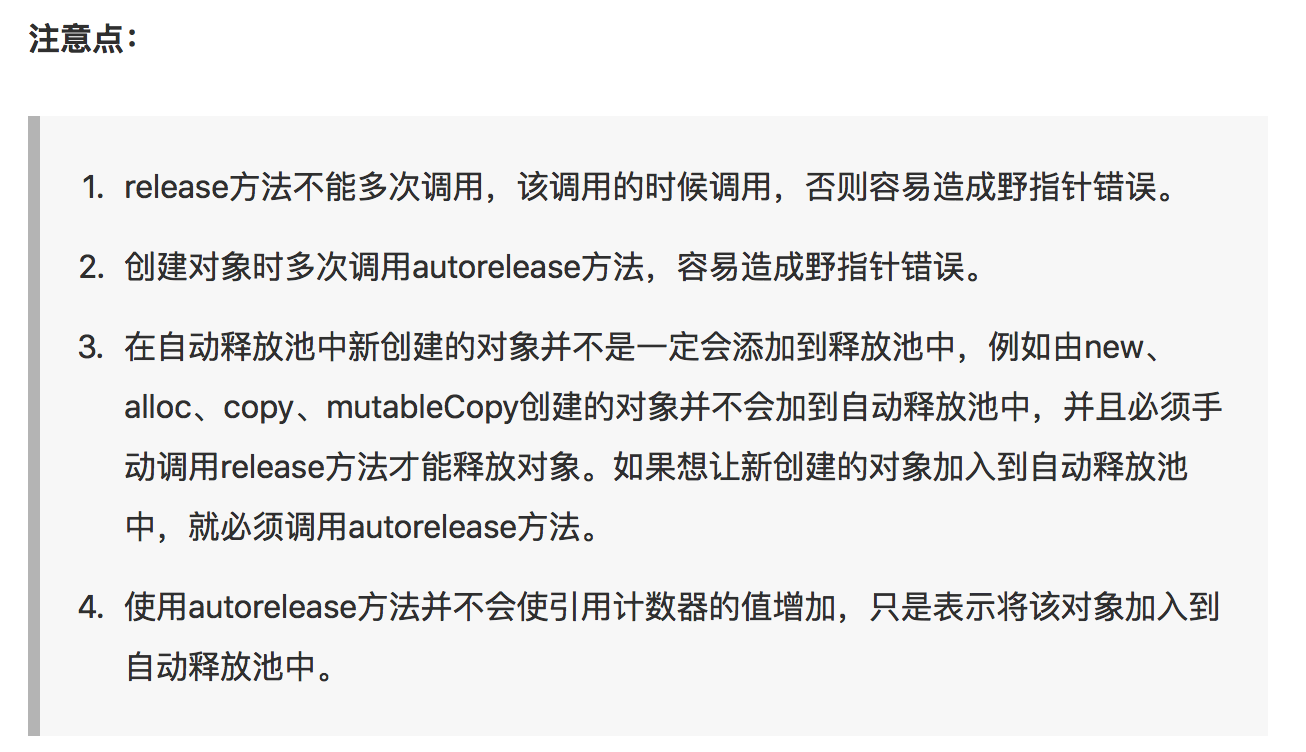
**顾名思义，autorelease即自动释放对象，不需要我们手动释放。从上面代码我们知道，在主函数中，创建对象obj后，总要手动调用[obj release]方法，这样无疑使工作量变大，且对我们的技术增长毫无意义。为了减少这种无意义的工作，可以使用Autorelease Pool方式。**

**Autorelease Pool即自动释放池，在Autorelease Pool内的对象在创建时只要调用了autorelease方法，那么在该池子内的对象的最后的release方法的调用将由编译器完成。**

**创建方式有两种：**



**在返回对象的方法中最好使用自动释放池释放对象，因为如果将新创建的对象作为返回值，由于在返回该对象之前并不能释放该对象，所以可以通过自动释放池来延迟该对象的释放。示例代码如下：**

****

* **自动引用计数器(ARC)**

**ARC将由编译器来自动完成对象引用计数器的控制，不需要手动完成。ARC模式下，创建的新对象通常由以下几种关键字来限定。**

* **\_\_strong(默认值)，由\_\_strong修饰的为强指针，对象只要有强指针指向就不会被销毁；每当一个强指针指向一个对象，该对象的的RC+1；**
* **\_\_weak,由\_\_weak修饰的为弱指针，弱指针所指向的对象并不会改变RC值，弱指针只表示是对对象的引用；当弱指针所指向的对象销毁时，该弱指针的值变为nil；**
* **\_\_unsafe\_unretained,\_\_unsafe\_unretained修饰的对象指针所指向的对象也不会改变RC值，也只表示是对对象的引用；当所指向的对象销毁时，该指针的值不会变为nil，仍是保留原有的地址；**

**在ARC模式下，MRC中的retain、release等方法变的不可用，因为ARC是不需要我们手动管理内存的，一切由编译器完成。**

****

* **总结**

**内存管理的本质是对对象引用计数器的操作，理解MRC模式下内存管理操作有助于我们对OC内存管理的理解。内存管理只针对对象而言，注意MRC和ARC下@property属性关键字的选择，在MRC模式下，OC对象通常使用retain关键字，非OC对象使用assign关键字，但是循环引用是一个例外，通常需要一端使用assign，一端使用retain；在ARC模式下，OC对象通常使用strong关键字，非OC对象使用assign关键字，但是循环引用是一个例外，通常需要一端使用strong，一端使用weak。**