**1、#import和#include的区别，@class代表什么？**

@class一般用于头文件中需要声明该类的某个实例变量的时候用到，在m文件中还是需要使用#import

而#import比起#include的好处就是不会引起重复包含

**2、谈谈Object-C的内存管理方式及过程？**

1.当你使用new,alloc和copy方法创建一个对象时,该对象的保留计数器值为1.当你不再使用该对象时,你要负责向该对象发送一条release或autorelease消息.这样,该对象将在使用寿命结束时被销毁.

2.当你通过任何其他方法获得一个对象时,则假设该对象的保留计数器值为1,而且已经被设置为自动释放,你不需要执行任何操作来确保该对象被清理.如果你打算在一段时间内拥有该对象,则需要保留它并确保在操作完成时释放它.

3.如果你保留了某个对象,你需要(最终)释放或自动释放该对象.必须保持[retain方法和release方法](http://www.cnblogs.com/martin1009/archive/2012/06/18/2553184.html" \t "_blank)的使用次数相等.

**3、Object-C有私有方法吗？私有变量呢？**

objective-c – 类里面的方法只有两种, 静态方法和实例方法. 这似乎就不是完整的面向对象了,按照OO的原则就是一个对象只暴露有用的东西. 如果没有了私有方法的话, 对于一些小范围的代码重用就不那么顺手了. 在类里面声名一个私有方法

@interface Controller : NSObject { NSString \*something; }

+ (void)thisIsAStaticMethod;

– (void)thisIsAnInstanceMethod;

@end

@interface Controller (private) -

(void)thisIsAPrivateMethod;

@end

@private可以用来修饰私有变量

在Objective‐C中，所有实例变量默认都是私有的，所有实例方法默认都是公有的

**4、Object-C有多继承吗？没有的话用什么代替？cocoa 中所有的类都是NSObject 的子类**

多继承在这里是用protocol 委托代理 来实现的

你不用去考虑繁琐的多继承 ,虚基类的概念.

ood的多态特性 在 obj-c 中通过委托来实现.

**5、内存管理 Autorelease、retain、copy、assign的set方法和含义？**

1，你初始化(alloc/init)的对象，你需要释放(release)它。例如：

NSMutableArray aArray = [[NSArray alloc] init]; 后，需要 [aArray release];

2，你retain或copy的，你需要释放它。例如：

[aArray retain] 后，需要 [aArray release];

3，被传递(assign)的对象，你需要斟酌的retain和release。例如：

obj2 = [[obj1 someMethod] autorelease];

对象2接收对象1的一个自动释放的值，或传递一个基本数据类型(NSInteger，NSString)时：你或希望将对象2进行retain，以防止它在被使用之前就被自动释放掉。但是在retain后，一定要在适当的时候进行释放。

关于索引计数(Reference Counting)的问题

retain值 = 索引计数(Reference Counting)

NSArray对象会retain(retain值加一)任何数组中的对象。当NSArray被卸载(dealloc)的时候，所有数组中的对象会 被 执行一次释放(retain值减一)。不仅仅是NSArray，任何收集类(Collection Classes)都执行类似操作。例如 NSDictionary，甚至UINavigationController。

Alloc/init建立的对象，索引计数为1。无需将其再次retain。

[NSArray array]和[NSDate date]等“方法”建立一个索引计数为1的对象，但是也是一个自动释放对象。所以是本地临时对象，那么无所谓了。如果是打算在全Class中使用的变量(iVar)，则必须retain它。

缺省的类方法返回值都被执行了“自动释放”方法。(\*如上中的NSArray)

在类中的卸载方法“dealloc”中，release所有未被平衡的NS对象。(\*所有未被autorelease，而retain值为1的)

**6、浅拷贝和深拷贝区别是什么**

简单的来说就是，在有指针的情况下，浅拷贝只是增加了一个指针指向已经存在的内存，而深拷贝就是增加一个指针并且申请一个新的内存，使这个增加的指针指向这个新的内存，采用深拷贝的情况下，释放内存的时候就不会出现在浅拷贝时重复释放同一内存的错误

**7、C和obj-c 如何混用**

1）obj-c的编译器处理后缀为m的文件时，可以识别obj-c和c的代码，处理mm文件可以识别obj-c,c,c++代码，但cpp文件必须只能用c/c++代码，而且cpp文件include的头文件中，也不能出现obj-c的代码，因为cpp只是cpp

2)在mm文件中混用cpp直接使用即可，所以obj-c混cpp不是问题

3）在cpp中混用obj-c其实就是使用obj-c编写的模块是我们想要的。

如果模块以类实现，那么要按照cpp class的标准写类的定义，头文件中不能出现obj-c的东西，包括#import cocoa的。实现文件中，即类的实现代码中可以使用obj-c的东西，可以import,只是后缀是mm。

如果模块以函数实现，那么头文件要按c的格式声明函数，实现文件中，c++函数内部可以用obj-c，但后缀还是mm或m。

总结：只要cpp文件和cpp include的文件中不包含obj-c的东西就可以用了，cpp混用obj-c的关键是使用接口，而不能直接使用 实现代 码，实际上cpp混用的是obj-c编译后的o文件，这个东西其实是无差别的，所以可以用。obj-c的编译器支持cpp

**8、Objective-C中类别和类扩展的区别。**

答案：category和extensions的不同在于后者可以添加属性。另外后者添加的方法是必须要实现的。

extensions可以认为是一个私有的Category。

**9、我们说的Objective-C是动态运行时语言是什么意思？**

答案：多态。 主要是将数据类型的确定由编译时，推迟到了运行时。

这个问题其实浅涉及到两个概念，运行时和多态。

简单来说，运行时机制使我们直到运行时才去决定一个对象的类别，以及调用该类别对象指定方法。

多态：不同对象以自己的方式响应相同的消息的能力叫做多态。意思就是假设生物类（life）都用有一个相同的方法-eat;

那人类属于生物，猪也属于生物，都继承了life后，实现各自的eat，但是调用是我们只需调用各自的eat方法。

也就是不同的对象以自己的方式响应了相同的消息（响应了eat这个选择器）。

因此也可以说，运行时机制是多态的基础？

**10、Objective-C堆和栈的区别？**

管理方式：对于栈来讲，是由编译器自动管理，无需我们手工控制；对于堆来说，释放工作由程序员控制，容易产生memory leak。

申请大小：

栈：在Windows下,栈是向低地址扩展的数据结构，是一块连续的内存的区域。这句话的意思是栈顶的地址和栈的最大容量是系统预先规定好的，在 WINDOWS下，栈的大小是2M（也有的说是1M，总之是一个编译时就确定的常数），如果申请的空间超过栈的剩余空间时，将提示overflow。因 此，能从栈获得的空间较小。

堆：堆是向高地址扩展的数据结构，是不连续的内存区域。这是由于系统是用链表来存储的空闲内存地址的，自然是不连续的，而链表的遍历方向是由低地址向高地址。堆的大小受限于计算机系统中有效的虚拟内存。由此可见，堆获得的空间比较灵活，也比较大。

碎片问题：对于堆来讲，频繁的new/delete势必会造成内存空间的不连续，从而造成大量的碎片，使程序效率降低。对于栈来讲，则不会存在这个问题，因为栈是先进后出的队列，他们是如此的一一对应，以至于永远都不可能有一个内存块从栈中间弹出

分配方式：堆都是动态分配的，没有静态分配的堆。栈有2种分配方式：静态分配和动态分配。静态分配是编译器完成的，比如局部变量的分配。动态分配由alloca函数进行分配，但是栈的动态分配和堆是不同的，他的动态分配是由编译器进行释放，无需我们手工实现。

分配效率：栈是机器系统提供的数据结构，计算机会在底层对栈提供支持：分配专门的寄存器存放栈的地址，压栈出栈都有专门的指令执行，这就决定了栈的效率比较高。堆则是C/C++函数库提供的，它的机制是很复杂的。

**7、线程和进程的区别？**

进程和线程都是由操作系统所体会的程序运行的基本单元，系统利用该基本单元实现系统对应用的并发性。

进程和线程的主要差别在于它们是不同的操作系统资源管理方式。进程有独立的地址空间，一个进程崩溃后，在保护模式下不会对其它进程产生影响，而线程只是一 个进程中的不同执行路径。线程有自己的堆栈和局部变量，但线程之间没有单独的地址空间，一个线程死掉就等于整个进程死掉，所以多进程的程序要比多线程的程 序健壮，但在进程切换时，耗费资源较大，效率要差一些。但对于一些要求同时进行并且又要共享某些变量的并发操作，只能用线程，不能用进程。

**8、堆和栈的区别？**

管理方式：对于栈来讲，是由编译器自动管理，无需我们手工控制；对于堆来说，释放工作由程序员控制，容易产生memory leak。

申请大小：

栈：在Windows下,栈是向低地址扩展的数据结构，是一块连续的内存的区域。这句话的意思是栈顶的地址和栈的最大容量是系统预先规定好的，在 WINDOWS下，栈的大小是2M（也有的说是1M，总之是一个编译时就确定的常数），如果申请的空间超过栈的剩余空间时，将提示overflow。因 此，能从栈获得的空间较小。

堆：堆是向高地址扩展的数据结构，是不连续的内存区域。这是由于系统是用链表来存储的空闲内存地址的，自然是不连续的，而链表的遍历方向是由低地址向高地址。堆的大小受限于计算机系统中有效的虚拟内存。由此可见，堆获得的空间比较灵活，也比较大。

碎片问题：对于堆来讲，频繁的new/delete势必会造成内存空间的不连续，从而造成大量的碎片，使程序效率降低。对于栈来讲，则不会存在这个问题，因为栈是先进后出的队列，他们是如此的一一对应，以至于永远都不可能有一个内存块从栈中间弹出

分配方式：堆都是动态分配的，没有静态分配的堆。栈有2种分配方式：静态分配和动态分配。静态分配是编译器完成的，比如局部变量的分配。动态分配由alloca函数进行分配，但是栈的动态分配和堆是不同的，他的动态分配是由编译器进行释放，无需我们手工实现。

分配效率：栈是机器系统提供的数据结构，计算机会在底层对栈提供支持：分配专门的寄存器存放栈的地址，压栈出栈都有专门的指令执行，这就决定了栈的效率比较高。堆则是C/C++函数库提供的，它的机制是很复杂的。

**9、Object-C的内存管理？**

1.当你使用new,alloc和copy方法创建一个对象时,该对象的保留计数器值为1.当你不再使用该对象时,你要负责向该对象发送一条release或autorelease消息.这样,该对象将在使用寿命结束时被销毁.

2.当你通过任何其他方法获得一个对象时,则假设该对象的保留计数器值为1,而且已经被设置为自动释放,你不需要执行任何操作来确保该对象被清理.如果你打算在一段时间内拥有该对象,则需要保留它并确保在操作完成时释放它.

3.如果你保留了某个对象,你需要(最终)释放或自动释放该对象.必须保持retain方法和release方法的使用次数相等.

**19、delegate和notification区别，分别在什么情况下使用？**

Delegate:

消息的发送者(sender)告知接收者(receiver)某个事件将要发生，delegate同意然然后发送者响应事件，delegate机制使得接收者可以改变发送者的行为。通常发送者和接收者的关系是直接的一对多的关系。

Notification:

消息的发送者告知接收者事件已经发生或者将要发送，仅此而已，接收者并不能反过来影响发送者的行为。通常发送者和接收者的关系是间接的多对多关系。

1. 效率肯定是delegate比nsnotification高。

2. delegate方法比notification更加直接，最典型的特征是，delegate方法往往需要关注返回值，也就是delegate方法 的结果。比如-windowShouldClose:，需要关心返回的是yes还是no。所以delegate方法往往包含should这个很传神的词。 也就是好比你做我的delegate，我会问你我想关闭窗口你愿意吗？你需要给我一个答案，我根据你的答案来决定如何做下一步。相反 的，notification最大的特色就是不关心接受者的态度，我只管把通告放出来，你接受不接受就是你的事情，同时我也不关心结果。所以 notification往往用did这个词汇，比如NSWindowDidResizeNotification，那么nswindow对象放出这个 notification后就什么都不管了也不会等待接受者的反应。

1）两个模块之间联系不是很紧密，就用notification传值，例如多线程之间传值用notificaiton。

2）delegate只是一种较为简单的回调，且主要用在一个模块中，例如底层功能完成了，需要把一些值传到上层去，就事先把上层的函数通过 delegate传到底层，然后在底层call这个delegate，它们都在一个模块中，完成一个功能，例如 说 NavgationController 从 B 界面到A 点返回按钮 (调用popViewController方法) 可以用delegate 比较好。