- 1.写一个**NSString**类的实现

+ (id)initWithCString:(c***t char *)nullTerminatedCString encoding:**
(NSStringEncoding)encoding;

```
+ (id) stringWithCString: (c*****t char*)nullTerminatedCString
        encoding: (NSStringEncoding)encoding
{
  NSString  *obj;
  obj = [self allocWithZone: NSDefaultMallocZone()];
  obj = [obj initWithCString: nullTerminatedCString encoding: encoding];
  return AUTORELEASE(obj);
}
```

2**static** 关键字的作用：

（1）函数体内 static 变量的作用范围为该函数体，不同于 auto 变量，该变量的内存只被分配一次，因此其值在下次调用时仍维持上次的值；

（2）在模块内的 static 全局变量可以被模块内所用函数访问，但不能被模块外其它函数访问；

（3）在模块内的 static 函数只可被这一模块内的其它函数调用，这个函数的使用范围被限制在声明它的模块内；

（4）在类中的 static 成员变量属于整个类所拥有，对类的所有对象只有一份拷贝；

（5）在类中的 static 成员函数属于整个类所拥有，这个函数不接收 this 指针，因而只能访问类的static成员变量。

3线程与进程的区别和联系**?**

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。

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。

4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。

5什么是键**-**值**,**键路径是什么

模型的性质是通过一个简单的键（通常是个字符串）来指定的。视图和控制器通过键来查找相应的属性值。在一个给定的实体中，同一个属性的所有值具有相同的数据类型。键-值编码技术用于进行这样的查找—它是一种间接访问对象属性的机制。

键路径是一个由用点作分隔符的键组成的字符串，用于指定一个连接在一起的对象性质序列。第一个键的

性质是由先前的性质决定的，接下来每个键的值也是相对于其前面的性质。键路径使您可以以独立于模型

实现的方式指定相关对象的性质。通过键路径，您可以指定对象图中的一个任意深度的路径，使其指向相

关对象的特定属性。

6目标**-**动作机制

目标是动作消息的接收者。一个控件，或者更为常见的是它的单元，以插座变量（参见"插座变量"部分）

的形式保有其动作消息的目标。

动作是控件发送给目标的消息，或者从目标的角度看，它是目标为了响应动作而实现的方法。
程序需要某些机制来进行事件和指令的翻译。这个机制就是目标-动作机制。
7**objc**的内存管理
?? 如果您通过分配和初始化（比如[[MyClass alloc] init]）的方式来创建对象，您就拥
有这个对象，需要负责该对象的释放。这个规则在使用NSObject的便利方法new 时也同样适用。
?? 如果您拷贝一个对象，您也拥有拷贝得到的对象，需要负责该对象的释放。
?? 如果您保持一个对象，您就部分拥有这个对象，需要在不再使用时释放该对象。
反过来，
?? 如果您从其它对象那里接收到一个对象，则您不拥有该对象，也不应该释放它（这个规则有少数
的例外，在参考文档中有显式的说明）。
8 自动释放池是什么,如何工作
当 您向一个对象发送一个autorelease消息时，Cocoa就会将该对象的一个引用放入到最新的自动释放
池。它仍然是个正当的对象，因此自动释放池定 义的作用域内的其它对象可以向它发送消息。当程序
执行到作用域结束的位置时，自动释放池就会被释放，池中的所有对象也就被释放。
1. ojc-c 是 通过一种"referring counting"(引用计数)的方式来管理内存的, 对象在开始分配内存(alloc)的
时候引用计数为一,以后每当碰到有copy,retain的时候引用计数都会加一, 每当碰到release和
autorelease的时候引用计数就会减一,如果此对象的计数变为了0, 就会被系统销毁.
2. NSAutoreleasePool 就是用来做引用计数的管理工作的,这个东西一般不用你管的.
3. autorelease和release没什么区别,只是引用计数减一的时机不同而已,autorelease会在对象的使用真
正结束的时候才做引用计数减一.
9类工厂方法是什么
类工厂方法的实现是为了向客户提供方便，它们将分配和初始化合在一个步骤中，返回被创建的对象，
并
进行自动释放处理。这些方法的形式是+ (type)className...（其中 className不包括任何前缀）。
工厂方法可能不仅仅为了方便使用。它们不但可以将分配和初始化合在一起，还可以为初始化过程提供
对
象的分配信息。
类工厂方法的另一个目的是使类（比如NSWorkspace）提供单件实例。虽然init...方法可以确认一
个类在每次程序运行过程只存在一个实例，但它需要首先分配一个“生的”实例，然后还必须释放该实
例。
工厂方法则可以避免为可能没有用的对象盲目分配内存。
10单件实例是什么
Foundation 和 Application Kit 框架中的一些类只允许创建单件对象，即这些类在当前进程中的唯一实
例。举例来说，NSFileManager 和NSWorkspace 类在使用时都是基于进程进行单件对象的实例化。当
向这些类请求实例的时候，它们会向您传递单一实例的一个引用，如果该实例还不存在，则首先进行实
例的分配 和初始化。单件对象充当控制中心的角色，负责指引或协调类的各种服务。如果类在概念上
只有一个实例（比如
NSWorkspace），就应该产生一个单件实例，而不是多个实例；如果将来某一天可能有多个实例，您
可
以使用单件实例机制，而不是工厂方法或函数。
11动态绑定
—在运行时确定要调用的方法

动 态绑定将调用方法的确定也推迟到运行时。在编译时，方法的调用并不和代码绑定在一起，只有在
消实发送出来之后，才确定被调用的代码。通过动态类型和动态绑 定技术，您的代码每次执行都可以
得到不同的结果。运行时因子负责确定消息的接收者和被调用的方法。运行时的消息分发机制为动态绑
定提供支持。当您向一个动 态类型确定了的对象发送消息时，运行环境系统会通过接收者的isa指针定
位对象的类，并以此为起点确定被调用的方法，方法和消息是动态绑定的。而且，您不 必在Objective-
C 代码中做任何工作，就可以自动获取动态绑定的好处。您在每次发送消息时，

特别是当消息的接收者是动态类型已经确定的对象时，动态绑定就会例行而透明地发生。
12**obj-c**的优缺点
objc优点：
1) Cateogies
2) Posing
3) 动态识别
4) 指标计算
5）弹性讯息传递
6) 不是一个过度复杂的 C 衍生语言
7) Objective-C 与 C++ 可混合编程
缺点:
1) 不支援命名空间
2) 不支持运算符重载

3） 不支持多重继承
4） 使用动态运行时类型，所有的方法都是函数调用，所以很多编译时优化方法都用不到。（如内联函数等），性能低劣。
13**sprintf,strcpy,memcpy**使用上有什么要注意的地方
strcpy是一个字符串拷贝的函数，它的函数原型为strcpy(char *dst, c*****t char *src);
将 src开始的一段字符串拷贝到dst开始的内存中去，结束的标志符号为'\0'，由于拷贝的长度不是由我们自己控制的，所以这个字符串拷贝很容易出错。具 备字符串拷贝功能的函数有memcpy，这是一个内存拷贝函数，它的函数原型为memcpy(char *dst, c*****t char* src, unsigned int len);
将长度为len的一段内存，从src拷贝到dst中去，这个函数的长度可控。但是会有内存叠加的问题。
sprintf是格式化函数。将一段数据通过特定的格式，格式化到一个字符串缓冲区中去。sprintf格式化的函数的长度不可控，有可能格式化后的字符串会超出缓冲区的大小，造成溢出。
14答案是：
a) int a; // An integer
b) int *a; // A pointer to an integer
c) int **a; // A pointer to a pointer to an integer
d) int a[10]; // An array of 10 integers
e) int *a[10]; // An array of 10 pointers to integers
f) int (*a)[10]; // A pointer to an array of 10 integers
g) int (*a)(int); // A pointer to a function a that takes an integer argument and returns an integer
h) int (*a[10])(int); // An array of 10 pointers to functi***** that take an integer argument and return an integer
15.**readwrite**，**readonly**，**assign**，**retain**，**copy**，**nonatomic**属性的作用
@property是一个属性访问声明，扩号内支持以下几个属性：
1，getter=getterName，setter=setterName，设置setter与getter的方法名
2，readwrite,readonly，设置可供访问级别
2，assign，setter方法直接赋值，不进行任何retain操作，为了解决原类型与环循引用问题
3，retain，setter方法对参数进行release旧值再retain新值，所有实现都是这个顺序(CC上有相关资料)
4，copy，setter方法进行Copy操作，与retain处理流程一样，先旧值release，再Copy出新的对象，retainCount为1。这是为了减少对上下文的依赖而引入的机制。
copy是在你不希望a和b共享一块内存时会使用到。a和b各自有自己的内存。
5，nonatomic，非原子性访问，不加同步，多线程并发访问会提高性能。注意，如果不加此属性，则默认是两个访问方法都为原子型事务访问。锁被加到所属对象实例级(我是这么理解的...)。
atomic和nonatomic用来决定编译器生成的getter和setter是否为原子操作。在多线程环境下，原子操作是必要的，否则有可能引起错 误的结果。加了atomic，setter函数会变成下面这样：
16 什么时候用delegate，什么时候用Notification？答：delegate针对one-to-one关系，并且reciever可以返回值 给sender，notification 可以针对one-to-one/many/none,reciever无法返回值给sender.所以，delegate用于sender希望接受到 reciever的某个功能反馈值，notification用于通知多个object某个事件。
17 什么是KVC和KVO？ 答：KVC(Key-Value-Coding)内部的实现：一个对象在调用setValue的时候，（1）首先根据方法名找到运行方法的时候所需要的环 境参数。（2）他会从自己isa指针结合环境参数，找到具体的方法实现的接口。（3）再直接查找得来的具体的方法实现。KVO（Key-Value-Observing）：当观察者为一个对象的属性进行了注册，被观察对象的isa指针被修改的时候，isa指针就会指向一个中间类，而不是真实的类。所以 isa指针其实不需要指向实例对象真实的类。所以我们的程序最好不要依赖于isa指针。在调用类的方法的时候，最好要明确对象实例的类名
18ViewController 的 loadView, viewDidLoad, viewDidUnload 分别是在什么时候调用的？在自定义ViewController的时候这几个函数里面应该做什么工作？答：viewDidLoad在view 从nib文件初始化时调用，loadView在controller的view为nil时调用。此方法在编程实现view时调用,view 控制器默认会注册memory warning notification,当view controller的任何view 没有用的时候，viewDidUnload会被调用，在这里实现将retain的view release,如果是retain的IBOutlet view 属性则不要在这里release,IBOutlet会负责release 。
19
"NSMutableString *"这个数据类型则是代表"NSMutableString"对象本身，这两者是有区别的。
而NSString只是对象的指针而已。
面向过程就是分析出解决问题所需要的步骤，然后用函数把这些步骤一步一步实现，使用的时候一个一个依次调用就可以了。
面向对象是把构成问题事务分解成各个对象，建立对象的目的不是为了完成一个步骤，而是为了描叙某个事物在整个解决问题的步骤中的行为。；
20类别的作用
类别主要有3个作用：
(1)将类的实现分散到多个不同文件或多个不同框架中。
(2)创建对私有方法的前向引用。

(3)向对象添加非正式协议。
类别的局限性
有两方面局限性:
(1)无法向类中添加新的实例变量，类别没有位置容纳实例变量。
(2)名称冲突，即当类别中的方法与原始类方法名称冲突时，类别具有更高的优先级。类别方法将完全取代初始方法从而无法再使用初始方法。
无法添加实例变量的局限可以使用字典对象解决
21关键字volatile有什么含意?并给出三个不同的例子:
一个定义为volatile的变量是说这变量可能会被意想不到地改变，这样，编译器就不会去假设这个变量的值了。精确地说就是，优化器在用到
这个变量时必须每次都小心地重新读取这个变量的值，而不是使用保存在寄存器里的备份。下面是volatile变量的几个例子:
• 并行设备的硬件寄存器（如：状态寄存器）
• 一个中断服务子程序中会访问到的非自动变量(Non-automatic variables)
• 多线程应用中被几个任务共享的变量

• 一个参数既可以是const还可以是volatile吗？解释为什么。
• 一个指针可以是volatile 吗？解释为什么。

下面是答案:
• 是的。一个例子是只读的状态寄存器。它是volatile因为它可能被意想不到地改变。它是const因为程序不应该试图去修改它。
• 是的。尽管这并不很常见。一个例子是当一个中服务子程序修该一个指向一个buffer的指针时。
22@synthesize 是系统自动生成getter和setter属性声明
@dynamic 是开发者自己提供相应的属性声明
@dynamic 意思是由开发人员提供相应的代码：对于只读属性需要提供 setter，对于读写属性需要提供 setter 和getter。@synthesize 意思是，除非开发人员已经做了，否则由编译器生成相应的代码，以满足属性声明。
查阅了一些资料确定@dynamic的意思是告诉编译器,属性的获取与赋值方法由用户自己实现, 不自动生成。
23Difference between shallow copy and deep copy?
浅复制和深复制的区别?
答案：浅层复制：只复制指向对象的指针，而不复制引用对象本身。
深层复制：复制引用对象本身。
意思就是说我有个A对象，复制一份后得到A_copy对象后，对于浅复制来说，A和A_copy指向的是同一个内存资源，复制的只不过是是一个指针，对象本身资源
还是只有一份，那如果我们对A_copy执行了修改操作,那么发现A引用的对象同样被修改，这其实违背了我们复制拷贝的一个思想。深复制就好理解了,内存中存在了
两份独立对象本身。
用网上一哥们通俗的话将就是:
浅复制好比你和你的影子，你完蛋，你的影子也完蛋
深复制好比你和你的克隆人，你完蛋，你的克隆人还活着。
24What is advantage of categories? What is difference between implementing a category and inheritance?
类别的作用？继承和类别在实现中有何区别?
答案：category 可以在不获悉，不改变原来代码的情况下往里面添加新的方法，只能添加，不能删除修改。
并且如果类别和原来类中的方法产生名称冲突，则类别将覆盖原来的方法，因为类别具有更高的优先级。
类别主要有3个作用:
(1)将类的实现分散到多个不同文件或多个不同框架中。
(2)创建对私有方法的前向引用。
(3)向对象添加非正式协议。
继承可以增加，修改或者删除方法，并且可以增加属性。
25.Difference between categories and extensions?
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。
26.Difference between protocol in objective c and interfaces in java?
oc中的协议和java中的接口概念有何不同?
答案：OC中的代理有2层含义，官方定义为 formal和informal protocol。前者和Java接口一样。
informal protocol中的方法属于设计模式考虑范畴，不是必须实现的，但是如果有实现，就会改变类的属性。
其实关于正式协议，类别和非正式协议我很早前学习的时候大致看过，也写在了学习教程里

“非正式协议概念其实就是类别的另一种表达方式“这里有一些你可能希望实现的方法，你可以使用他们更好的完成工作”。
这个意思是，这些是可选的。比如我门要一个更好的方法，我们就会申明一个这样的类别去实现。然后你在后期可以直接使用这些更好的方法。
这么看，总觉得类别这玩意儿有点像协议的可选协议。"
现在来看，其实protocal已经开始对两者都统一和规范起来操作，因为资料中说“非正式协议使用interface修饰“，
现在我们看到协议中两个修饰词：“必须实现(@requied)”和“可选实现(@optional)”。
26What are KVO and KVC?
答案：kvc:键 - 值编码是一种间接访问对象的属性使用字符串来标识属性，而不是通过调用存取方法，直接或通过实例变量访问的机制。
很多情况下可以简化程序代码。apple文档其实给了一个很好的例子。
kvo:键值观察机制，他提供了观察某一属性变化的方法，极大的简化了代码。
具体用看到嗯哼用到过的一个地方是对于按钮点击变化状态的的监控。
比如我自定义的一个button
[cpp]
```
[self addObserver:self forKeyPath:@"highlighted" options:0 context:nil];
#pragma mark KVO
- (void)observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object change:(NSDictionary *)change context:(void *)context
{
    if ([keyPath isEqualToString:@"highlighted"] ) {
        [self setNeedsDisplay];
    }
}
```
对于系统是根据keypath去取的到相应的值发生改变，理论上来说是和kvc机制的道理是一样的。
对于kvc机制如何通过key寻找到value：
“当通过KVC调用对象时，比如：[self valueForKey:@”someKey”]时，程序会自动试图通过几种不同的方式解析这个调用。首先查找对象是否带有 someKey 这个方法，如果没找到，会继续查找对象是否带有someKey这个实例变量（iVar），如果还没有找到，程序会继续试图调用 -(id) valueForUndefinedKey:这个方法。如果这个方法还是没有被实现的话，程序会抛出一个 NSUndefinedKeyException 异常错误。
(cocoachina.com注：Key-Value Coding查找方法的时候，不仅仅会查找someKey这个方法，还会查找getsomeKey这个方法，前面加一个get，或者_someKey以及 _getsomeKey这几种形式。同时，查找实例变量的时候也会不仅仅查找someKey这个变量，也会查找_someKey这个变量是否存在。）
设计valueForUndefinedKey:方法的主要目的是当你使用-(id)valueForKey方法从对象中请求值时，对象能够在错误发生前，有最后的机会响应这个请求。这样做有很多好处，下面的两个例子说明了这样做的好处。“
来至cocoa，这个说法应该挺有道理。
因为我们知道button却是存在一个highlighted实例变量.因此为何上面我们只是add一个相关的keypath就行了，
27What is purpose of delegates?
代理的作用?
答案：代理的目的是改变或传递控制链。允许一个类在某些特定时刻通知到其他类，而不需要获取到那些类的指针。可以减少框架复杂度。
另外一点，代理可以理解为java中的回调监听机制的一种类似。
28What are mutable and immutable types in Objective C?
oc中可修改和不可以修改类型。
答案：可修改不可修改的集合类。这个我个人简单理解就是可动态添加修改和不可动态添加修改一样。
比如NSArray和NSMutableArray。前者在初始化后的内存控件就是固定不可变的，后者可以添加等，可以动态申请新的内存空间
29When we call objective c is runtime language what does it mean?
我们说的oc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？~~~

30what is difference between NSNotification and protocol?
通知和协议的不同之处?
答案：协议有控制链(has-a)的关系，通知没有。
首先我一开始也不太明白，什么叫控制链（专业术语了~）。但是简单分析下通知和代理的行为模式，我们大致可以有自己的理解
简单来说，通知的话，它可以一对多，一条消息可以发送给多个消息接受者。
代理按我们的理解，到不是直接说不能一对多，比如我们知道的明星经济代理人，很多时候一个经济人负责好几个明星的事务。
只是对于不同明星间，代理的事物对象都是不一样的，一一对应，不可能说明天要处理A明星要一个发布会，代理人发出处理发布会的消息后，别称B的
发布会了。但是通知就不一样，他只关心发出通知，而不关心多少接收到感兴趣要处理。
因此控制链（has-a从英语单词大致可以看出，单一拥有和可控制的对应关系。
31What is push notification?
什么是推送消息?
答案：太简单，不作答~~~~~~~~~~
这是cocoa上的答案。
其实到不是说太简单，只是太泛泛的一个概念的东西。就好比说，什么是人。
推送通知更是一种技术。
简单点就是客户端获取资源的一种手段。
普通情况下，都是客户端主动的pull。
推送则是服务器端主动push。
32.Polymorphism?
关于多态性
答案：多态，子类指针可以赋值给父类。
这个题目其实可以出到一切面向对象语言中，
因此关于多态，继承和封装基本最好都有个自我意识的理解，也并非一定要把书上资料上写的能背出来。
最重要的是转化成自我理解。
33
What is responder chain?
说说响应链
答案： 事件响应链。包括点击事件，画面刷新事件等。在视图栈内从上至下，或者从下之上传播。
可以说点事件的分发，传递以及处理。具体可以去看下touch事件这块。因为问的太抽象化了
严重怀疑题目出到越后面就越笼统。
34Difference between frame and bounds?
frame和bounds有什么不同?
答案:frame指的是：该view在父view坐标系统中的位置和大小。（参照点是父亲的坐标系统）
bounds指的是：该view在本身坐标系统中 的位置和大小。（参照点是本身坐标系统）
35
.Difference between method and selector?
方法和选择器有何不同?
答案：selector是一个方法的名字，method是一个组合体，包含了名字和实现.
36NSOperation queue?
答案：存放NSOperation的集合类。
操作和操作队列，基本可以看成java中的线程和线程池的概念。用于处理ios多线程开发的问题。
网上部分资料提到一点是，虽然是queue，但是却并不是带有队列的概念，放入的操作并非是按照严格的先进现出。
这边又有个疑点是，对于队列来说，先进先出的概念是Afunc添加进队列，Bfunc紧跟着也进入队列，Afunc先执行这个是必然的，
但是Bfunc是等Afunc完全操作完以后，B才开始启动并且执行，因此队列的概念离乱上有点违背了多线程处理这个概念。
但是转念一想其实可以参考银行的取票和叫号系统。
因此对于A比B先排队取票但是B率先执行完操作，我们亦然可以感性认为这还是一个队列。
但是后来看到一票关于这操作队列话题的文章，其中有一句提到
“因为两个操作提交的时间间隔很近，线程池中的线程，谁先启动是不定的。”
瞬间觉得这个queue名字有点忽悠人了，还不如pool~
综合一点，我们知道他可以比较大的用处在于可以帮组多线程编程就好了。
37What is lazy loading?
答案：懒汉模式，只在用到的时候才去初始化。
也可以理解成延时加载。
我觉得最好也最简单的一个列子就是tableView中图片的加载显示了。
一个延时载，避免内存过高，一个异步加载，避免线程堵塞。
38Can we use two tableview controllers on one viewcontroller?
是否在一个视图控制器中嵌入两个tableview控制器?

答案：一个视图控制只提供了一个View视图，理论上一个tableViewController也不能放吧，
只能说可以嵌入一个tableview视图。当然，题目本身也有歧义，如果不是我们定性思维认为的UIViewController，
而是宏观的表示视图控制者，那我们倒是可以把其看成一个视图控制者，它可以控制多个视图控制器，比如TabbarController
那样的感觉。
39Can we use one tableview with two different datasources? How you will achieve this?
一个tableView是否可以关联两个不同的数据源？你会怎么处理?
答案：首先我们从代码来看，数据源如何关联上的，其实是在数据源关联的代理方法里实现的。
因此我们并不关心如何去关联他，他怎么关联上，方法只是让我返回根据自己的需要去设置如相关的数据源。
因此，我觉得可以设置多个数据源啊，但是有个问题是，你这是想干嘛呢？想让列表如何显示，不同的数据源分区块显示?
40id、nil代表什么?
id 和void *并非完全一样。在上面的代码中，id是指向struct objc_object的一个指针，这个意思基本上是说，id是一个指向任何一个继承了Object（或者NSObject）类的对象。需要注意的是id 是一个指针，所以你在使用id的时候不需要加星号。比如id foo=nil定义了一个nil指针，这个指针指向NSObject的一个任意子类。而id *foo=nil则定义了一个指针，这个指针指向另一个指针，被指向的这个指针指向NSObject的一个子类。
nil和C语言的NULL相同，在objc/objc.h中定义。nil表示一个Objctive-C对象，这个对象的指针指向空（没有东西就是空）。
首字母大写的Nil和nil有一点不一样，Nil定义一个指向空的类（是Class，而不是对象）。
SEL是“selector”的一个类型，表示一个方法的名字
Method（我们常说的方法）表示一种类型，这种类型与selector和实现(implementation)相关
IMP定义为 id (*IMP) (id, SEL, ...)。这样说来， IMP是一个指向函数的指针，这个被指向的函数包括id(“self”指针)，调用的SEL（方法名），再加上一些其他参数.说白了IMP就是实现方法。
41层和UIView的区别是什么?
答： 两者最大的区别是,图层不会直接渲染到屏幕上，UIView是iOS系统中界面元素的基础，所有的界面元素都是继承自它。它本身完全是由 CoreAnimation来实现的。它真正的绘图部分，是由一个CALayer类来管理。UIView本身更像是一个CALayer的管理器。一个 UIView上可以有n个CALayer，每个layer显示一种东西，增强UIView的展现能力。
42GCD 为Grand Central Dispatch的缩写。 Grand Central Dispatch (GCD)是Apple开发的一个多核编程的较新的解决方法。在Mac OS X 10.6雪豹中首次推出，并在最近引入到了iOS4.0。 GCD是一个替代诸如NSThread等技术的很高效和强大的技术。GCD完全可以处理诸如 数据锁定和资源泄漏等复杂的异步编程问题。
GCD可以完成很多事情，但是这里仅关注在iOS应用中实现多线程所需的一些基础知识。 在开始之前，需要理解是要提供给GCD队列的是代码块，用于在系统或者用户创建的的队列上调度运行。
声明一个队列
如下会返回一个用户创建的队列:
dispatch_queue_t myQueue = dispatch_queue_create("com.iphonedevblog.post", NULL);其中，第一个参数是标识队列的，第二个参数是用来定义队列的参数（目前不支持，因此传入NULL）。
执行一个队列
如下会异步执行传入的代码:
dispatch_async(myQueue, ^{ [self doSomething]; });其中，首先传入之前创建的队列，然后提供由队列运行的代码块。
声明并执行一个队列
如 果不需要保留要运行的队列的引用，可以通过如下代码实现之前的功能:
dispatch_async(dispatch_queue_create ("com.iphonedevblog.post", NULL), ^{ [self doSomething]; });
如果需要暂停一个队列，可以调用如下代码。暂停一个队列会阻止和该队列相关的所有代码运行。
dispatch_suspend(myQueue);**暂停一个队列**
如 果暂停一个队列不要忘记恢复。暂停和恢复的操作和内存管理中的retain和release类似。调用dispatch_suspend会增加暂停计数， 而dispatch_resume则会减少。队列只有在暂停计数变成零的情况下才开始运行。dispatch_resume(myQueue);**恢复一个队列** **从队列中在主线程运行代码**有 些操作无法在异步队列运行，因此必须在主线程（每个应用都有一个）上运行。UI绘图以及任何对NSNotificationCenter的调用必须在主线 程长进行。在另一个队列中访问主线程并运行代码的示例如下： dispatch_sync(dispatch_get_main_queue(), ^{ [self dismissLoginWindow]; });注意，dispatch_suspend （以及dispatch_resume）在主线程上不起作用。
使用GCD，可以让你的程序不会失去响应. 多线程不容易使用，用了GCD，会让它变得简单。你无需专门进行线程管理, 很棒!

```
dispatch_queue_t t1=dispatch_queue_create("1", NULL);
    dispatch_queue_t t2=dispatch_queue_create("2", NULL);
    dispatch_async(t1, ^{
```

```
        [self print1];
    });
    dispatch_async(t2, ^{
        [self print2];
    });
```

43Provider是指某个iPhone软件的Push服务器，这篇文章我将使用.net作为Provider。
APNS 是Apple Push Notification Service（Apple Push服务器）的缩写，是苹果的服务器。
上图可以分为三个阶段。
第一阶段：.net应用程序把要发送的消息、目的iPhone的标识打包，发给APNS。
第二阶段：APNS在自身的已注册Push服务的iPhone列表中，查找有相应标识的iPhone，并把消息发到iPhone。
第三阶段：iPhone把发来的消息传递给相应的应用程序， 并且按照设定弹出Push通知。
http://blog.csdn.net/zhuqilin0/article/details/6527113 //消息推送机制
看内存泄露时候：在搜索中搜索run 找到Run Static Snalyzer .
44.可扩展标记语言extensible markup language;XML
2.用于标记电子文件使其具有结构性的标记语言，可以用来标记数据、定义数据类型，是一种允许用户对自己的标记语言进行定义的源语言。
3，数据库提供了更强有力的数据存储和分析能力，例如：数据索引、排序、查找、相关一致性等，XML仅仅是存储数据。
4.XML与HTML的设计区别是：XML的核心是数据，其重点是数据的内容。而HTML 被设计用来显示数据，其重点是数据的显示。
5.XML和HTML语法区别：HTML的标记不是所有的都需要成对出现，XML则要求所有的标记必须成对出现；HTML标记不区分大小写，XML则 大小敏感,即区分大小写。

结合

XML的简单使其易于在任何应用程序中读写数据，这使XML很快成为数据交换的唯一公共语言，虽然不同的应用软件也支持其它的数据交换格式，但不久之后他们都将支持XML，那就意味着程序可以更容易的与Windows,Mac OS,Linux以及其他平台下产生的信息结合，然后可以很容易加载XML数据到程序中并分析他，并以XML格式输出结果。

XML去掉了之前令许多开发人员头疼的SGML（标准通用标记语言）的随意语法。在XML中，采用了如下的语法：

1 任何的起始标签都必须有一个结束标签。

2 可以采用另一种简化语法，可以在一个标签中同时表示起始和结束标签。这种语法是在大于符号之前紧跟一个斜线（/），例如<tag/ >。XML解析器会将其翻译成<tag></tag>。

3 标签必须按合适的顺序进行嵌套，所以结束标签必须按镜像顺序匹配起始标签，例如**this is a*sample*string**。这好比是将起始和结束标签看作是数学中的左右括号：在没有关闭所有的内部括号之前，是不能关闭外面的括号的。

4 所有的特性都必须有值。

5 所有的特性都必须在值的周围加上双引号。

45union u

```
{
   double a;
   int b;
};

union u2
{
   char a[13];
   int b;
};

union u3
{
   char a[13];
   char b;
};

cout<<sizeof(u)<<endl; // 8
cout<<sizeof(u2)<<endl; // 16
cout<<sizeof(u3)<<endl; // 13
```

都知道union的大小取决于它所有的成员中，占用空间最大的一个成员的大小。所以对于u来说，大小就是最大的double类型成员a了，所以 sizeof(u)=sizeof(double)=8。但是对于u2和u3，最大的空间都是char[13]类型的数组，为什么u3的大小是13，而 u2是16呢？关键在于u2中的成员int b。由于int类型成员的存在，使u2的对齐方式变成4，也就是说，u2的大小必须在4的对界上，所以占用的空间变成了16（最接近13的对界）。 struct s1

```
{
   char a;
   double b;
   int c;
   char d;
};

struct s2
{
   char a;
   char b;
   int c;
   double d;
};

cout<<sizeof(s1)<<endl; // 24
cout<<sizeof(s2)<<endl; // 16
```

同样是两个char类型，一个int类型，一个double类型，但是因为对界问题，导致他们的大小不同。计算结构体大小可以采用元素摆放法，我举例子 说明一下：首先，CPU判断结构体的对界，根据上一节的结论，s1和s2的对界都取最大的元素类型，也就是double类型的对界8。然后开始摆放每个元素。

对于s1，首先把a放到8的对界，假定是0，此时下一个空闲的地址是1，但是下一个元素d是double类型，要放到8的对界 上，离1最接近的地址是8了，所以d被放在了8，此时下一个空闲地址变成了16，下一个元素c的对界是4，16可以满足，所以c放在了16，此时下一个空 闲地址变成了20，下一个元素d需要对界1，也正好落在对界上，所以d放在了20，结构体在地址21处结束。由于s1的大小需要是8的倍数，所以21- 23的空间被保留，s1的大小变成了24。

对于s2，首先把a放到8的对界，假定是0，此时下一个空闲地址是1，下一个元素的对界也 是1，所以b摆放在1，下一个空闲地址变成了2；下一个元素c的对界是4，所以取离2最近的地址4摆放c，下一个空闲地址变成了8，下一个元素d的对界是 8，所以d摆放在8，所有元素摆放完毕，结构体在15处结束，占用总空间为16，正好是8的倍数。

46ASIDownloadCache 设置下载缓存

它对Get请求的响应数据进行缓存（被缓存的数据必需是成功的200请求）：

```
[ASIHTTPRequest setDefaultCache:[ASIDownloadCache
sharedCache]];
```

当设置缓存策略后，所有的请求都被自动的缓存起来。

另外，如果仅仅希望某次请求使用缓存操作，也可以这样使用：

```
ASIHTTPRequest *request = [ASIHTTPRequest
requestWithURL:url];
[request setDownloadCache:[ASIDownloadCache sharedCache]];
```

缓存存储方式

你可以设置缓存的数据需要保存多长时间，ASIHTTPRequest提供了两种策略：

a，ASICacheForSessionDurationCacheStoragePolicy，默认策略，基于session的缓存数据存储。当下次运行或[ASIHTTPRequest clearSession]时，缓存将失效。

b，ASICachePermanentlyCacheStoragePolicy，把缓存数据永久保存在本地，

如：

```
ASIHTTPRequest *request = [ ASIHTTPRequest requestWithURL:url ];
[ request
setCacheStoragePolicy:ASICachePermanentlyCacheStoragePolicy ];
```

47HTTP协议详解

HTTP是一个属于应用层的面向对象的协议，由于其简捷、快速的方式，适用于分布式超媒体信息系统。目前在WWW中使用的是HTTP/1.0的第六版，HTTP/1.1的规范化工作正在进行之中。

http（超文本传输协议）是一个基于请求与响应模式的、无状态的、应用层的协议，常基于TCP的连接方式，HTTP1.1版本中给出一种持续连接的机制，绝大多数的Web开发，都是构建在HTTP协议之上的Web应用。
HTTP协议的主要特点可概括如下：
1.支持客户/服务器模式。
2.简单快速：客户向服务器请求服务时，只需传送请求方法和路径。请求方法常用的有GET、HEAD、POST。每种方法规定了客户与服务器联系的类型不同。由于HTTP协议简单，使得HTTP服务器的程序规模小，因而通信速度很快。
3.灵活：HTTP允许传输任意类型的数据对象。正在传输的类型由Content-Type加以标记。
4.无连接：无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间。
5.无状态：HTTP协议是无状态协议。无状态是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快。
48**URL**
HTTP URL (URL是一种特殊类型的URI是他的子类，包含了用于查找某个资源的足够的信息)的格式如下：
[url=http://host[%22%22port][abs_path/]http://host[":"port][abs_path[/url]]
http表示要通过HTTP协议来定位网络资源；host表示合法的Internet主机域名或者IP地址；port指定一个端口号，为空则使用缺省端口 80；abs_path指定请求资源的URI；如果URL中没有给出abs_path，那么当它作为请求URI时，必须以“/”的形式给出，通常这个工作 浏览器自动帮我们完成。
49TCP/UDP区别联系
TCP---传输控制协议,提供的是面向连接、可靠的字节流服务。当客户和服务器彼此交换数据前，必须先在双方之间建立一个TCP连接，之后才能传输数据。TCP提供超时重发，丢弃重复数据，检验数据，流量控制等功能，保证数据能从一端传到另一端。
UDP--- 用户数据报协议，是一个简单的面向数据报的运输层协议。UDP不提供可靠性，它只是把应用程序传给IP层的数据报发送出去，但是并不能保证它们能到达目的 地。由于UDP在传输数据报前不用在客户和服务器之间建立一个连接，且没有超时重发等机制，故而传输速度很快
TCP（Transmission Control Protocol，传输控制协议）是基于连接的协议，也就是说，在正式收发数据前，必须和对方建立可靠的连接。一个TCP连接必须要经过三次“对话”才能 建立起来，我们来看看这三次对话的简单过程：1.主机A向主机B发出连接请求数据包；2.主机B向主机A发送同意连接和要求同步（同步就是两台主机一个在 发送，一个在接收，协调工作）的数据包；3.主机A再发出一个数据包确认主机B的要求同步：“我现在就发，你接着吧！”，这是第三次对话。三次“对话”的 目的是使数据包的发送和接收同步，经过三次“对话”之后，主机A才向主机B正式发送数据。
UDP（User Data Protocol，用户数据报协议）是与TCP相对应的协议。它是面向非连接的协议，它不与对方建立连接，而是直接就把数据包发送过去！ UDP适用于一次只传送少量数据、对可靠性要求不高的应用环境。
tcp协议和udp协议的差别
是否连接 面向连接 面向非连接
传输可靠性 可靠 不可靠
应用场合 传输大量数据 少量数据
速度 慢 快
50**socket**连接和**http**连接的区别
简单说，你浏览的网页（网址以http://开头)都是http协议传输到你的浏览器的, 而http是基于socket之上的。socket是一套完成tcp，udp协议的接口。
HTTP协议：简单对象访问协议，对应于应用层 ，HTTP协议是基于TCP连接的
tcp协议： 对应于传输层
ip协议： 对应于网络层
TCP/IP是传输层协议，主要解决数据如何在网络中传输；而HTTP是应用层协议，主要解决如何包装数据。
Socket是对TCP/IP协议的封装，Socket本身并不是协议，而是一个调用接口（API），通过Socket，我们才能使用TCP/IP协议。
http连接：http连接就是所谓的短连接，即客户端向服务器端发送一次请求，服务器端响应后连接即会断掉；
socket 连接：socket连接就是所谓的长连接，理论上客户端和服务器端一旦建立起连接将不会主动断掉；但是由于各种环境因素可能会是连接断开，比如说：服务器 端或客户端主机down了，网络故障，或者两者之间长时间没有数据传输，网络防火墙可能会断开该连接以释放网络资源。所以当一个socket连接中没有数 据的传输，那么为了维持连接需要发送心跳消息~~具体心跳消息格式是开发者自己定义的
我 们已经知道网络中的进程是通过socket来通信的，那什么是socket呢？socket起源于Unix，而Unix/Linux基本哲学之一就是“一 切皆文件”，都可以用“打开open –> 读写write/read –> 关闭close”模式来操作。我的理解就是Socket就是该模式的一个实现，socket即是一种特殊的文件，一些socket函数就是对

其进行的操作 （读/写IO、打开、关闭），这些函数我们在后面进行介绍。我们在传输数据时，可以只使用（传输层）TCP/IP协议，但是那样的话，如果没有应用层，便 无法识别数据内容，如果想要使传输的数据有意义，则必须使用到应用层协议，应用层协议有很多，比如HTTP、FTP、TELNET等，也可以自己定义应用 层协议。WEB使用HTTP协议作应用层协议，以封装HTTP文本信息，然后使用TCP/IP做传输层协议将它发到网络上。
1)Socket是一个针对TCP和UDP编程的接口，你可以借助它建立TCP连接等等。而TCP和UDP协议属于传输层 。
而http是个应用层的协议，它实际上也建立在TCP协议之上。
(HTTP是轿车，提供了封装或者显示数据的具体形式；Socket是发动机，提供了网络通信的能力。)
2）Socket是对TCP/IP协议的封装，Socket本身并不是协议，而是一个调用接口（API），通过Socket，我们才能使用TCP/IP协 议。Socket的出现只是使得程序员更方便地使用TCP/IP协议栈而已，是对TCP/IP协议的抽象，从而形成了我们知道的一些最基本的函数接口。
51什么是**TCP**连接的三次握手
第一次握手：客户端发送syn包(syn=j)到服务器，并进入SYN_SEND状态，等待服务器确认；
第二次握手：服务器收到syn包，必须确认客户的SYN（ack=j+1），同时自己也发送一个SYN包（syn=k），即SYN+ACK包，此时服务器进入SYN_RECV状态；
第三次握手：客户端收到服务器的SYN＋ACK包，向服务器发送确认包ACK(ack=k+1)，此包发送完毕，客户端和服务器进入ESTABLISHED状态，完成三次握手。
握 手过程中传送的包里不包含数据，三次握手完毕后，客户端与服务器才正式开始传送数据。理想状态下，TCP连接一旦建立，在通信双方中的任何一方主动关闭连 接之前，TCP 连接都将被一直保持下去。断开连接时服务器和客户端均可以主动发起断开TCP连接的请求，断开过程需要经过“四次握手”（过程就不细写了，就是服务器和客 户端交互，最终确定断开）
52利用**Socket**建立网络连接的步骤
建立Socket连接至少需要一对套接字，其中一个运行于客户端，称为ClientSocket ，另一个运行于服务器端，称为ServerSocket 。
套接字之间的连接过程分为三个步骤：服务器监听，客户端请求，连接确认。
1。服务器监听：服务器端套接字并不定位具体的客户端套接字，而是处于等待连接的状态，实时监控网络状态，等待客户端的连接请求。
2。客户端请求：指客户端的套接字提出连接请求，要连接的目标是服务器端的套接字。为此，客户端的套接字必须首先描述它要连接的服务器的套接字，指出服务器端套接字的地址和端口号，然后就向服务器端套接字提出连接请求。
3。 连接确认：当服务器端套接字监听到或者说接收到客户端套接字的连接请求时，就响应客户端套接字的请求，建立一个新的线程，把服务器端套接字的描述发给客户 端，一旦客户端确认了此描述，双方就正式建立连接。而服务器端套接字继续处于监听状态，继续接收其他客户端套接字的连接请求。
53进程与线程
进程（process）是一块包含了某些资源的内存区域。操作系统利用进程把它的工作划分为一些功能单元。
进程中所包含的一个或多个执行单元称为线程（thread）。进程还拥有一个私有的虚拟地址空间，该空间仅能被它所包含的线程访问。
通常在一个进程中可以包含若干个线程，它们可以利用进程所拥有的资源。
在引入线程的操作系统中，通常都是把进程作为分配资源的基本单位，而把线程作为独立运行和独立调度的基本单位。
由于线程比进程更小，基本上不拥有系统资源，故对它的调度所付出的开销就会小得多，能更高效的提高系统内多个程序间并发执行的程度。
简而言之,一个程序至少有一个进程,一个进程至少有一个线程.一个程序就是一个进程，而一个程序中的多个任务则被称为线程。

线程只能归属于一个进程并且它只能访问该进程所拥有的资源。当操作系统创建一个进程后，该进程会自动申请一个名为主线程或首要线程的线程。应用程序（application）是由一个或多个相互协作的进程组成的。
另外，进程在执行过程中拥有独立的内存单元，而多个线程共享内存，从而极大地提高了程序的运行效率。
线程在执行过程中与进程还是有区别的。每个独立的线程有一个程序运行的入口、顺序执行序列和程序的出口。但是线程不能够独立执行，必须依存在应用程序中，由应用程序提供多个线程执行控制。
从逻辑角度来看，多线程的意义在于一个应用程序中，有多个执行部分可以同时执行。但操作系统并没有将多个线程看做多个独立的应用，来实现进程的调度和管理以及资源分配。这就是进程和线程的重要区别。
进程是具有一定独立功能的程序关于某个数据集合上的一次运行活动,进程是系统进行资源分配和调度的一个独立单位.
线程是进程的一个实体,是CPU调度和分派的基本单位,它是比进程更小的能独立运行的基本单位.线程自己基本上不拥有系统资源,只拥有一点在运行中必不可少的资源(如程序计数器,一组寄存器和栈),但是它可与同属一个进程的其他的线程共享进程所拥有的全部资源.
一个线程可以创建和撤销另一个线程;同一个进程中的多个线程之间可以并发执行.

54多线程
多线程编程是防止主线程堵塞，增加运行效率等等的最佳方法。而原始的多线程方法存在很多的毛病，包括线程锁死等。在Cocoa中，Apple提供了NSOperation这个类，提供了一个优秀的多线程编程方法。
本次介绍NSOperation的子集，简易方法的NSInvocationOperation：

一个NSOperationQueue 操作队列，就相当于一个线程管理器，而非一个线程。因为你可以设置这个线程管理器内可以并行运行的的线程数量等等
55oc语法里的@perpoerty不用写@synzhesize了，自动填充了。并且的_name;
写方法时候不用提前声明。llvm 全局方法便利。
枚举类型。enum hello:Integer{ } 冒号后面直接可以跟类型，以前是：
enum hello{} 后面在指定为Integer .
桥接。ARC 自动release retain 的时候 CFString CFArray . Core Fountion. 加上桥接_brige 才能区分CFString和NSString 而现在自动区分了，叫固定桥接。

下拉刷新封装好了。
UICollectionViewController. 可以把表格分成多列。

Social Framework(社交集成**)**
UIActivityViewController来询问用户的社交行为

缓存：就是存放在临时文件里，比如新浪微博请求的数据，和图片，下次请求看这里有没有值。
56Singleton（单例模式），也叫单子模式，是一种常用的软件设计模式。在应用这个模式时，单例对象的类必须保证只有一个实例存在。
代码如下：

```
static ClassA *classA = nil；//静态的该类的实例
+ (ClassA *)sharedManager
 {
@synchronized(self) {
if (!classA) {
classA = [[super allocWithZone:NULL]init];
}
return classA;
}
}
+ (id)allocWithZone:(NSZone *)zone {
return [[self sharedManager] retain];
}
- (id)copyWithZone:(NSZone *)zone {
return self;
}
- (id)retain {
return self;
}
- (NSUIntger)retainCount {
return NSUIntgerMax;
}
- (oneway void)release {
}
- (id)autorelease {
return self;
}
-(void)dealloc{
}
```

57请写一个C函数，若处理器是Big_endian的，则返回0；若是Little_endian的，则返回1 int checkCPU( ) {

```
    {
      union w
          {
                int a;
                char b;
           } c;
```

```
        c.a = 1;
    return  (c.b ==1);
 }
}
```

剖 析： 嵌入式系统开发者应该对Little-endian和Big-endian模式非常了解。采用Little-endian模式的CPU对操作数的存放方式 是从低字节到高字节， Big-endian 模式的CPU对操作数的存放方式是从高字节到低字节。 在弄清楚这个之前要弄清楚这个问题： 字节从右到坐为从高到低! 假设从地址0x4000开始存放: 0x12345678,是也个32位四个字节的数据，最高字节是0x12,最低字节是0x78： 在Little-endian模式CPU内存中的存放方式为： (高字节在高地址, 低字节在低地址)
内存地址0x4000 0x4001 0x4002 0x4003
存放内容 0x78 0x56 0x34 0x12
大端机则相反。

有 的处理器系统采用了小端方式进行数据存放，如Intel的奔腾。有的处理器系统采用了大端方式进行数据存放，如IBM半导体和Freescale的 PowerPC处理器。不仅对于处理器，一些外设的设计中也存在着使用大端或者小端进行数据存放的选择。 因此在一个处理器系统中，有可能存在大端和小端模式同时存在的现象。这一现象为系统的软硬件设计带来了不小的麻烦，这要求系统设计工程师，必须深入理解大 端和小端模式的差别。大端与小端模式的差别体现在一个处理器的寄存器，指令集，系统总线等各个层次中。 联合体union的存放顺序是所有成员都从低地址开始存放的。 以上是网上的原文。让我们看看在ARM处理器上union是如何存储的呢？ 地址A ---------- lA lA+1 lA+2 lA+3 lint a; l l l l ---------- lA lchar b; l l ---------

如果是小端如何存储c.a的呢？

地址A ----------

---------- lA lA+1 lA+2 lA+3 l int a;
l0x01 l0x00 l0x00 l0x00 l ---------- lA lchar b; l l ---------

如果是大端如何存储c.a的呢？

地址A ----------

--------- lA lA+1 lA+2 lA+3 lint a; l0x00 l0x00 l0x00 l0x01 l
---------- lA lchar b; l l ---------

现在知道为什么c.b==0的话是大端，c.b==1的话就是小端了吧。

58
堆和栈上的指针
指针所指向的这块内存是在哪里分配的,在堆上称为堆上的指针,在栈上为栈上的指针.
在堆上的指针,可以保存在全局数据结构中,供不同函数使用访问同一块内存.
在栈上的指针,在函数退出后,该内存即不可访问.
59什么是指针的释放?
具体来说包括两个概念.
1 释放该指针指向的内存,只有堆上的内存才需要我们手工释放,栈上不需要.
2 将该指针重定向为NULL.
60数据结构中的指针?
其实就是指向一块内存的地址,通过指针传递,可实现复杂的内存访问.
7 函数指针?
指向一块函数的入口地址.

8 指针作为函数的参数?
比如指向一个复杂数据结构的指针作为函数变量
这种方法避免整个复杂数据类型内存的压栈出栈操作,提高效率.
注意:指针本身不可变,但指针指向的数据结构可以改变.

9 指向指针的指针?
指针指向的变量是一个指针,即具体内容为一个指针的值,是一个地址.
此时指针指向的变量长度也是4位.
61指针与地址的区别?
区别:
1指针意味着已经有一个指针变量存在,他的值是一个地址,指针变量本身也存放在一个长度为四个字节的地址当中,而地址概念本身并不代表有任何变量存在.
2 指针的值,如果没有限制,通常是可以变化的,也可以指向另外一个地址.
地址表示内存空间的一个位置点,他是用来赋给指针的,地址本身是没有大小概念,指针指向变量的大小,取决于地址后面存放的变量类型.

62指针与数组名的关系?
其值都是一个地址,但前者是可以移动的,后者是不可变的.

12 怎样防止指针的越界使用问题?
必须让指针指向一个有效的内存地址,
1 防止数组越界
2 防止向一块内存中拷贝过多的内容
3 防止使用空指针
4 防止改变const修改的指针
5 防止改变指向静态存储区的内容
6 防止两次释放一个指针
7 防止使用野指针.

13 指针的类型转换?
指针转换通常是指针类型和void * 类型之前进行强制转换,从而与期望或返回void指针的函数进行正确的交接.
63static有什么用途? （请至少说明两种）
1.限制变量的作用域
2.设置变量的存储域
7. 引用与指针有什么区别?
1) 引用必须被初始化，指针不必。
2) 引用初始化以后不能被改变，指针可以改变所指的对象。
2) 不存在指向空值的引用，但是存在指向空值的指针。
8. 描述实时系统的基本特性
在特定时间内完成特定的任务，实时性与可靠性
64全局变量和局部变量在内存中是否有区别？如果有，是什么区别?
全局变量储存在静态数据库，局部变量在堆栈
10. 什么是平衡二叉树?
左右子树都是平衡二叉树 且左右子树的深度差值的绝对值不大于1
65堆栈溢出一般是由什么原因导致的?
没有回收垃圾资源
12. 什么函数不能声明为虚函数?
constructor
13. 冒泡排序算法的时间复杂度是什么?
O(n^2)
14. 写出float x 与“零值”比较的if语句。
if(x>0.000001&&x<-0.000001)
16. Internet采用哪种网络协议？该协议的主要层次结构?
tcp/ip 应用层/传输层/网络层/数据链路层/物理层
17. Internet物理地址和IP地址转换采用什么协议?
ARP (Address Resolution Protocol)（地址解析協議）
18.IP地址的编码分为哪俩部分?
IP地址由两部分组成，网络号和主机号。不过是要和“子网掩码”按位与上之后才能区分哪些是网络位哪些是主机位。
2.用户输入M,N值，从1至N开始顺序循环数数，每数到M输出该数值，直至全部输出。写出C程序。
循环链表，用取余操作做
3.不能做switch()的参数类型是:
switch的参数不能为实型。
華為
1、局部变量能否和全局变量重名?
答：能，局部会屏蔽全局。要用全局变量，需要使用"::"
局部变量可以与全局变量同名，在函数内引用这个变量时，会用到同名的局部变量，而不会用到全局变量。对于有些编译器而言，在同一个函数内可以定义多个同名的局部变量，比如在两个循环体内都定义一个同名的局部变量，而那个局部变量的作用域就在那个循环体内
2、如何引用一个已经定义过的全局变量?
答：extern
可以用引用头文件的方式，也可以用extern关键字，如果用引用头文件方式来引用某个

在头文件中声明的全局变理，假定你将那个变写错了，那么在编译期间会报错，如果你用extern方式引用时，假定你犯了同样的错误，那么在编译期间不会报错，而在连接期间报错

3、全局变量可不可以定义在可被多个.C文件包含的头文件中？为什么？

答：可以，在不同的C文件中以static形式来声明同名全局变量。

可以在不同的C文件中声明同名的全局变量，前提是其中只能有一个C文件中对此变量赋初值，此时连接不会出错

4、语句for(； 1 ；)有什么问题？它是什么意思？

答：和while(1)相同。

5、do……while和while……do有什么区别？

答：前一个循环一遍再判断，后一个判断以后再循环

661.IP Phone的原理是什么？

IPV6

2.TCP/IP通信建立的过程怎样，端口有什么作用？

三次握手，确定是哪个应用程序使用该协议

3.1号信令和7号信令有什么区别，我国某前广泛使用的是那一种？

4.列举5种以上的电话新业务？

微软亚洲技术中心的面试题！！！

1．进程和线程的差别。

线程是指进程内的一个执行单元,也是进程内的可调度实体.

与进程的区别:

(1)调度：线程作为调度和分配的基本单位，进程作为拥有资源的基本单位

(2)并发性：不仅进程之间可以并发执行，同一个进程的多个线程之间也可并发执行

(3)拥有资源：进程是拥有资源的一个独立单位，线程不拥有系统资源，但可以访问隶属于进程的资源.

(4)系统开销：在创建或撤消进程时，由于系统都要为之分配和回收资源，导致系统的开销明显大于创建或撤消线程时的开销。

2.测试方法

人工测试：个人复查、抽查和会审

机器测试：黑盒测试和白盒测试

2．Heap与stack的差别。

Heap是堆，stack是栈。

Stack的空间由操作系统自动分配/释放，Heap上的空间手动分配/释放。

Stack空间有限，Heap是很大的自由存储区

C中的malloc函数分配的内存空间即在堆上,C++中对应的是new操作符。

程序在编译期对变量和函数分配内存都在栈上进行,且程序运行过程中函数调用时参数的传递也在栈上进行

3．Windows下的内存是如何管理的？

4．介绍.Net和.Net的安全性。

5．客户端如何访问.Net组件实现Web Service？

6．C/C++编译器中虚表是如何完成的？

7．谈谈COM的线程模型。然后讨论进程内/外组件的差别。

8．谈谈IA32下的分页机制

小页(4K)两级分页模式，大页(4M)一级

9．给两个变量，如何找出一个带环单链表中是什么地方出现环的？

一个递增一，一个递增二，他们指向同一个接点时就是环出现的地方

10．在IA32中一共有多少种办法从用户态跳到内核态？

通过调用门，从ring3到ring0，中断从ring3到ring0，进入vm86等等

11．如果只想让程序有一个实例运行，不能运行两个。像winamp一样，只能开一个窗口，怎样实现？

用内存映射或全局原子（互斥变量）、查找窗口句柄..

FindWindow，互斥，写标志到文件或注册表,共享内存。

67如何截取键盘的响应，让所有的‘a’变成‘b’？

键盘钩子SetWindowsHookEx

13．Apartment在COM中有什么用？为什么要引入？

14．存储过程是什么？有什么用？有什么优点？

我的理解就是一堆sql的集合，可以建立非常复杂的查询，编译运行，所以运行一次后，以后再运行速度比单独执行SQL快很多

15．Template有什么特点？什么时候用？

16．谈谈Windows DNA结构的特点和优点。

网络编程中设计并发服务器，使用多进程 与 多线程 ，请问有什么区别?
1，进程：子进程是父进程的复制品。子进程获得父进程数据空间、堆和栈的复制品。
2，线程：相对与进程而言，线程是一个更加接近与执行体的概念，它可以与同进程的其他线程共享数据，但拥有自己的栈空间，拥有独立的执行序列。
两者都可以提高程序的并发度，提高程序运行效率和响应时间。
线程和进程在使用上各有优缺点：线程执行开销小，但不利于资源管理和保护；而进程正相反。同时，线程适合于在SMP机器上运行，而进程则可以跨机器迁移。
思科

68**2.**找错题

试题1：

```
void test1()
{
  char string[10];
  char* str1 = "0123456789";
  strcpy( string, str1 );
}
```

试题2：

```
void test2()
{
  char string[10], str1[10];
  int i;
  for(i=0; i<10; i++)
  {
    str1 = 'a';
  }
  strcpy( string, str1 );
}
```

试题3：

```
void test3(char* str1)
{
  char string[10];
  if( strlen( str1 ) <= 10 )
  {
    strcpy( string, str1 );
  }
}
```

解答：

试题1字符串str1需要11个字节才能存放下（包括末尾的'\0'），而string只有10个字节的空间，strcpy会导致数组越界；

对试题2，如果面试者指出字符数组str1不能在数组内结束可以给3分；如果面试者指出strcpy(string, str1)调用使得从str1[url=]内存[/url]起复制到string内存起所复制的字节数具有不确定性可以给7分，在此基础上指出库函数 strcpy工作方式的给10分；

对试题3，if(strlen(str1) <= 10)应改为if(strlen(str1) < 10)，因为strlen的结果未统计'\0'所占用的1个字节。

剖析：

考查对基本功的掌握：

(1)字符串以'\0'结尾；

(2)对数组越界把握的敏感度；

(3)库函数strcpy的工作方式，如果编写一个标准strcpy函数的总分值为10，下面给出几个不同得分

的答案：

2分

```
void strcpy( char *strDest, char *strSrc )
{
    while( (*strDest++ = * strSrc++) != '\0' );
}
```

4分

```
void strcpy( char *strDest, const char *strSrc )
//将源字符串加const，表明其为输入参数，加2分
{
    while( (*strDest++ = * strSrc++) != '\0' );
}
```

7分

```
void strcpy(char *strDest, const char *strSrc)
{
   //对源地址和目的地址加非0断言，加3分
   assert( (strDest != NULL) && (strSrc != NULL) );
   while( (*strDest++ = * strSrc++) != '\0' );
}
```

10分

```
//为了实现链式操作，将目的地址返回，加3分！

char * strcpy( char *strDest, const char *strSrc )
{
   assert( (strDest != NULL) && (strSrc != NULL) );
   char *address = strDest;
   while( (*strDest++ = * strSrc++) != '\0' );
     return address;
}
```

从2分到10分的几个答案我们可以清楚的看到，小小的strcpy竟然暗藏着这么多玄机，真不是盖的！需要多么扎实的基本功才能写一个完美的strcpy啊！

(4)对strlen的掌握，它没有包括字符串末尾的'\0'。

读者看了不同分值的strcpy版本，应该也可以写出一个10分的strlen函数了，完美的版本为： int strlen(const char *str) //输入参数const

```
{
   assert( strt != NULL ); //断言字符串地址非0
   int len;
   while( (*str++) != '\0' )
   {
      len++;
   }
   return len;
}
```

试题4：

```
void GetMemory( char *p )
{
   p = (char *) malloc( 100 );
}

void Test( void )
{
```

```
    char *str = NULL;
    GetMemory( str );
    strcpy( str, "hello world" );
    printf( str );
}
```

试题5：

```
char *GetMemory( void )
{
    char p[] = "hello world";
    return p;
}

void Test( void )
{
    char *str = NULL;
    str = GetMemory();
    printf( str );
}
```

试题6：

```
void GetMemory( char **p, int num )
{
    *p = (char *) malloc( num );
}

void Test( void )
{
    char *str = NULL;
    GetMemory( &str, 100 );
    strcpy( str, "hello" );
    printf( str );
}
```

试题7：

```
void Test( void )
{
    char *str = (char *) malloc( 100 );
    strcpy( str, "hello" );
    free( str );
    ... //省略的其它语句
}
```

解答：

试题4传入中GetMemory(char *p)函数的形参为字符串指针，在函数内部修改形参并不能真正的改变传入形参的值，执行完

```
char *str = NULL;
GetMemory( str );
```

后的str仍然为NULL；

试题5中

```
char p[] = "hello world";
return p;
```

的p[]数组为函数内的局部自动变量，在函数返回后，内存已经被释放。这是许多程序员常犯的错误，其根源在于不理解变量的生存期。

试题6的GetMemory避免了试题4的问题，传入GetMemory的参数为字符串指针的指针，但是在GetMemory中执行申请内存及赋值语句

```
*p = (char *) malloc( num );
```
后未判断内存是否申请成功，应加上：

```
if ( *p == NULL )
{
   ...//进行申请内存失败处理
}
```
试题7存在与试题6同样的问题，在执行

```
char *str = (char *) malloc(100);
```
后未进行内存是否申请成功的判断；另外，在free(str)后未置str为空，导致可能变成一个“野”指针，应加上：

```
str = NULL;
```
试题6的Test函数中也未对malloc的内存进行释放。

剖析：

试题4～7考查面试者对内存操作的理解程度，基本功扎实的面试者一般都能正确的回答其中50~60的错误。但是要完全解答正确，却也绝非易事。

对内存操作的考查主要集中在：

（1）指针的理解；

（2）变量的生存期及作用范围；

（3）良好的动态内存申请和释放习惯。

再看看下面的一段程序有什么错误：

```
swap( int* p1,int* p2 )
{
   int *p;
   *p = *p1;
   *p1 = *p2;
   *p2 = *p;
}
```
在swap函数中，p是一个“野”指针，有可能指向系统区，导致程序运行的崩溃。在VC++中DEBUG运行时提示错误“Access Violation”。该程序应该改为：

```
swap( int* p1,int* p2 )
{
   int p;
   p = *p1;
   *p1 = *p2;
   *p2 = p;
}
```
[img=12,12]file:///D:/鱼鱼软件/鱼鱼多媒体日记本/temp/{56068A28-3D3B-4D8B-9F82-AC1C3E9B128C}_arc_d[1].gif[/img] **3.**内功题　　试题1：分别给出BOOL，int，float，指针变量 与“零值”比较的 if 语句（假设变量名为var）

解答：

BOOL型变量：if(!var)

int型变量： if(var==0)

float型变量：

```
const float EPSINON = 0.00001;
```

if ((x >= - EPSINON) && (x <= EPSINON)

指针变量： if(var==NULL)

剖析：

考查对0值判断的“内功”，BOOL型变量的0判断完全可以写成if(var==0)，而int型变量也可以写成if(!var)，指针变量的判断也可以写成if(!var)，上述写法虽然程序都能正确运行，但是未能清晰地表达程序的意思。

一般的，如果想让if判断一个变量的“真”、“假”，应直接使用if(var)、if(!var)，表明其为“逻辑”判断；如果用if判断一个数值型变 量(short、int、long等)，应该用if(var==0)，表明是与0进行“数值”上的比较；而判断指针则适宜用if(var==NULL)， 这是一种很好的编程习惯。

浮点型变量并不精确，所以不可将float变量用“==”或“！=”与数字比较，应该设法转化成“>=”或“<=”形式。如果写成if (x == 0.0)，则判为错，得0分。

试题2：以下为Windows NT下的32位C++程序，请计算sizeof的值

```
void Func ( char str[100] )
{
  sizeof( str ) = ?
}

void *p = malloc( 100 );
sizeof ( p ) = ?
```

解答：

```
sizeof( str ) = 4
sizeof ( p ) = 4
```

剖析：

Func (char str[100])函数中数组名作为函数形参时，在函数体内，数组名失去了本身的内涵，仅仅只是一个指针；在失去其内涵的同时，它还失去了其常量特性，可以作自增、自减等操作，可以被修改。

数组名的本质如下：

（1）数组名指代一种数据结构，这种数据结构就是数组；

例如：

```
char str[10];
cout < < sizeof(str) < < endl;
```

输出结果为10，str指代数据结构char[10]。

（2）数组名可以转换为指向其指代实体的指针，而且是一个指针常量，不能作自增、自减等操作，不能被修改；

```
char str[10];
str++; //编译出错，提示str不是左值
```

（3）数组名作为函数形参时，沦为普通指针。

Windows NT 32位平台下，指针的长度（占用内存的大小）为4字节，故sizeof(str)、sizeof (p)都为4。

试题3：写一个“标准”宏MIN，这个宏输入两个参数并返回较小的一个。另外，当你写下面的代码时会发生什么事?

```
least = MIN(*p++, b);
```

解答：

```
#define MIN(A,B) ((A) <= (B) ? (A) : (B))
```

MIN(*p++, b)会产生宏的副作用

剖析：

这个面试题主要考查面试者对宏定义的使用，宏定义可以实现类似于函数的功能，但是它终归不是函数，而宏定义中括弧中的“参数”也不是真的参数，在宏展开的时候对“参数”进行的是一对一的替换。

程序员对宏定义的使用要非常小心，特别要注意两个问题：

（1）谨慎地将宏定义中的“参数”和整个宏用用括弧括起来。所以，严格地讲，下述解答：

```
#define MIN(A,B) (A) <= (B) ? (A) : (B)
#define MIN(A,B) (A <= B ? A : B )
```

都应判0分；

（2）防止宏的副作用。

宏定义#define MIN(A,B) ((A) <= (B) ? (A) : (B))对MIN(*p++, b)的作用结果是：

```
((*p++) <= (b) ? (*p++) : (*p++))
```

这个表达式会产生副作用，指针p会作三次++自增操作。

除此之外，另一个应该判0分的解答是：

```
#define MIN(A,B) ((A) <= (B) ? (A) : (B));
```

这个解答在宏定义的后面加“;”，显示编写者对宏的概念模糊不清，只能被无情地判0分并被面试官淘汰。

试题4：为什么标准头文件都有类似以下的结构?

```
#ifndef __INCvxWorksh
#define __INCvxWorksh
#ifdef __cplusplus

extern "C" {
#endif
/*...*/
#ifdef __cplusplus
}

#endif
#endif /* __INCvxWorksh */
```

解答：

头文件中的编译宏

```
#ifndef   __INCvxWorksh
#define   __INCvxWorksh
#endif
```

的作用是防止被重复引用。

作为一种面向对象的语言，C++支持函数重载，而过程式语言C则不支持。函数被C++编译后在symbol库中的名字与C语言的不同。例如，假设某个函数的原型为：

```
void foo(int x, int y);
```

该函数被C编译器编译后在symbol库中的名字为_foo，而C++编译器则会产生像_foo_int_int之类的名字。_foo_int_int这样的名字包含了函数名和函数参数数量及类型信息，C++就是考这种机制来实现函数重载的。

为了实现C和C++的混合编程，C++提供了C连接交换指定符号extern "C"来解决名字匹配问题，函

数声明前加上extern "C"后，则编译器就会按照C语言的方式将该函数编译为_foo，这样C语言中就可以调用C++的函数了。[img=12,12]file:///D:/鱼鱼软件/鱼鱼多媒体日记本/temp/{C74A38C4-432E-4799-B54D-73E2CD3C5206}_arc_d[1].gif[/img]
试题5：编写一个函数，作用是把一个char组成的字符串循环右移n个。比如原来是“abcdefghi”如果n=2，移位后应该是“hiabcdefgh”

函数头是这样的：

```
//pStr是指向以'\0'结尾的字符串的指针
//steps是要求移动的n

void LoopMove ( char * pStr, int steps )
{
   //请填充...
}
```

解答：

正确解答1：

```
void LoopMove ( char *pStr, int steps )
{
   int n = strlen( pStr ) - steps;
   char tmp[MAX_LEN];
   strcpy ( tmp, pStr + n );
   strcpy ( tmp + steps, pStr);
   *( tmp + strlen ( pStr ) ) = '\0';
   strcpy( pStr, tmp );
}
```

正确解答2：

```
void LoopMove ( char *pStr, int steps )
{
   int n = strlen( pStr ) - steps;
   char tmp[MAX_LEN];
   memcpy( tmp, pStr + n, steps );
   memcpy(pStr + steps, pStr, n );
   memcpy(pStr, tmp, steps );
}
```

剖析：

这个试题主要考查面试者对标准库函数的熟练程度，在需要的时候引用库函数可以很大程度上简化程序编写的工作量。

最频繁被使用的库函数包括：

（1） strcpy

（2） memcpy

（3） memset

此份面试题包含40个题目，是现在网上能搜索到的一个比较热的一份，但是答案并不是很详细和完整，基本答案来着cocoaChina，和一些自己的补充。
1.Difference between shallow copy and deep copy?
浅复制和深复制的区别?
答案：浅层复制：只复制指向对象的指针，而不复制引用对象本身。
深层复制：复制引用对象本身。
意思就是说我有个A对象，复制一份后得到A_copy对象后，对于浅复制来说，A和A_copy指向的是同一个内存资源，复制的只不过是是一个指针，对象本身资源
还是只有一份，那如果我们对A_copy执行了修改操作,那么发现A引用的对象同样被修改，这其实违背了我们复制拷贝的一个思想。深复制就好理解了,内存中存在了

两份独立对象本身。
用网上一哥们通俗的话将就是：
浅复制好比你和你的影子，你完蛋，你的影子也完蛋
深复制好比你和你的克隆人，你完蛋，你的克隆人还活着。

2.What is advantage of categories? What is difference between implementing a category and inheritance?
类别的作用？继承和类别在实现中有何区别?
答案：category 可以在不获悉，不改变原来代码的情况下往里面添加新的方法，只能添加，不能删除修改。
并且如果类别和原来类中的方法产生名称冲突，则类别将覆盖原来的方法，因为类别具有更高的优先级。
类别主要有3个作用：
(1)将类的实现分散到多个不同文件或多个不同框架中。
(2)创建对私有方法的前向引用。
(3)向对象添加非正式协议。
继承可以增加，修改或者删除方法，并且可以增加属性。

3.Difference between categories and extensions?
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。

4.Difference between protocol in objective c and interfaces in java?
oc中的协议和java中的接口概念有何不同?
答案：OC中的代理有2层含义，官方定义为 formal和informal protocol。前者和Java接口一样。
informal protocol中的方法属于设计模式考虑范畴，不是必须实现的，但是如果有实现，就会改变类的属性。
其实关于正式协议，类别和非正式协议我很早前学习的时候大致看过，也写在了学习教程里
“非正式协议概念其实就是类别的另一种表达方式“这里有一些你可能希望实现的方法，你可以使用他们更好的完成工作”。
这个意思是，这些是可选的。比如我门要一个更好的方法，我们就会申明一个这样的类别去实现。然后你在后期可以直接使用这些更好的方法。
这么看，总觉得类别这玩意儿有点像协议的可选协议。"
现在来看，其实protocal已经开始对两者都统一和规范起来操作，因为资料中说“非正式协议使用interface修饰“，
现在我们看到协议中两个修饰词：“必须实现(@requied)”和“可选实现(@optional)”。

5.What are KVO and KVC?
答案：kvc:键 - 值编码是一种间接访问对象的属性使用字符串来标识属性，而不是通过调用存取方法，直接或通过实例变量访问的机制。
很多情况下可以简化程序代码。apple文档其实给了一个很好的例子。
kvo:键值观察机制，他提供了观察某一属性变化的方法，极大的简化了代码。
具体用看到嗯哼用到过的一个地方是对于按钮点击变化状态的的监控。
比如我自定义的一个button
[cpp]
[self addObserver:self forKeyPath:@"highlighted" options:0 context:nil];

#pragma mark KVO

- (void)observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object change:(NSDictionary *)change context:(void *)context
{
    if ([keyPath isEqualToString:@"highlighted"]) {
        [self setNeedsDisplay];
    }
}

对于系统是根据keypath去取的到相应的值发生改变，理论上来说是和kvc机制的道理是一样的。
对于kvc机制如何通过key寻找到value：
“当通过KVC调用对象时，比如：[self valueForKey:@”someKey”]时，程序会自动试图通过几种不同的方式解析这个调用。首先查找对象是否带有 someKey 这个方法，如果没找到，会继续查找对象是否带有someKey这个实例变量（iVar），如果还没有找到，程序会继续试图调用 -(id) valueForUndefinedKey:这

个方法。如果这个方法还是没有被实现的话，程序会抛出一个NSUndefinedKeyException 异常错误。

(cocoachina.com注：Key-Value Coding查找方法的时候，不仅仅会查找someKey这个方法，还会查找getsomeKey这个方法，前面加一个get，或者_someKey以及 _getsomeKey这几种形式。同时，查找实例变量的时候也会不仅仅查找someKey这个变量，也会查找_someKey这个变量是否存在。）

设计valueForUndefinedKey:方法的主要目的是当你使用-(id)valueForKey方法从对象中请求值时，对象能够在错误发生前，有最后的机会响应这个请求。这样做有很多好处，下面的两个例子说明了这样做的好处。“
来至cocoa，这个说法应该挺有道理。
因为我们知道button却是存在一个highlighted实例变量.因此为何上面我们只是add一个相关的keypath就行了，
可以按照kvc查找的逻辑理解，就说的过去了。

6.What is purpose of delegates?
代理的作用?
答案：代理的目的是改变或传递控制链。允许一个类在某些特定时刻通知到其他类，而不需要获取到那些类的指针。可以减少框架复杂度。
另外一点，代理可以理解为java中的回调监听机制的一种类似。

7.What are mutable and immutable types in Objective C?
oc中可修改和不可以修改类型。
答案：可修改不可修改的集合类。这个我个人简单理解就是可动态添加修改和不可动态添加修改一样。比如NSArray和NSMutableArray。前者在初始化后的内存控件就是固定不可变的，后者可以添加等，可以动态申请新的内存空间。

8.When we call objective c is runtime language what does it mean?
我们说的oc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? ~~~

9.what is difference between NSNotification and protocol?
通知和协议的不同之处?
答案：协议有控制链(has-a)的关系，通知没有。
首先我一开始也不太明白，什么叫控制链（专业术语了~）。但是简单分析下通知和代理的行为模式，我们大致可以有自己的理解
简单来说，通知的话，它可以一对多，一条消息可以发送给多个消息接受者。
代理按我们的理解，到不是直接说不能一对多，比如我们知道的明星经济代理人，很多时候一个经济人负责好几个明星的事务。
只是对于不同明星间，代理的事物对象都是不一样的，一一对应，不可能说明天要处理A明星要一个发布会，代理人发出处理发布会的消息后，别称B的
发布会了。但是通知就不一样，他只关心发出通知，而不关心多少接收到感兴趣要处理。
因此控制链（has-a从英语单词大致可以看出，单一拥有和可控制的对应关系。

10.What is push notification?
什么是推送消息?
答案：太简单，不作答~~~~~~~~~~
这是cocoa上的答案。
其实到不是说太简单，只是太泛泛的一个概念的东西。就好比说，什么是人。
推送通知更是一种技术。
简单点就是客户端获取资源的一种手段。
普通情况下，都是客户端主动的pull。
推送则是服务器端主动push。

11.Polymorphism?
关于多态性
答案：多态，子类指针可以赋值给父类。

这个题目其实可以出到一切面向对象语言中，
因此关于多态，继承和封装基本最好都有个自我意识的理解，也并非一定要把书上资料上写的能背出来。
最重要的是转化成自我理解。

12.Singleton?
对于单例的理解
答案：11，12题目其实出的有点泛泛的感觉了，可能说是编程语言需要或是必备的基础。
基本能用熟悉的语言写出一个单例，以及可以运用到的场景或是你编程中碰到过运用的此种模式的框架类等。
进一步点，考虑下如何在多线程访问单例时的安全性。

13.What is responder chain?
说说响应链
答案： 事件响应链。包括点击事件，画面刷新事件等。在视图栈内从上至下，或者从下之上传播。
可以说点事件的分发，传递以及处理。具体可以去看下touch事件这块。因为问的太抽象化了
严重怀疑题目出到越后面就越笼统。

14.Difference between frame and bounds?
frame和bounds有什么不同?
答案:frame指的是：该view在父view坐标系统中的位置和大小。（参照点是父亲的坐标系统）
bounds指的是：该view在本身坐标系统中 的位置和大小。（参照点是本身坐标系统）

15.Difference between method and selector?
方法和选择器有何不同?
答案：selector是一个方法的名字，method是一个组合体，包含了名字和实现.
详情可以看apple文档。

16.Is there any garbage collection mechanism in Objective C.?
OC的垃圾回收机制?
答案： OC2.0有Garbage collection，但是iOS平台不提供。
一般我们了解的objective-c对于内存管理都是手动操作的，但是也有自动释放池。
但是差了大部分资料，貌似不要和arc机制搞混就好了。
求更多~~

17.NSOperation queue?
答案：存放NSOperation的集合类。
操作和操作队列，基本可以看成java中的线程和线程池的概念。用于处理ios多线程开发的问题。
网上部分资料提到一点是，虽然是queue，但是却并不是带有队列的概念，放入的操作并非是按照严格的先进现出。
这边又有个疑点是，对于队列来说，先进先出的概念是Afunc添加进队列，Bfunc紧跟着也进入队列，Afunc先执行这个是必然的，
但是Bfunc是等Afunc完全操作完以后，B才开始启动并且执行，因此队列的概念离乱上有点违背了多线程处理这个概念。
但是转念一想其实可以参考银行的取票和叫号系统。
因此对于A比B先排队取票但是B率先执行完操作，我们亦然可以感性认为这还是一个队列。
但是后来看到一票关于这操作队列话题的文章，其中有一句提到
“因为两个操作提交的时间间隔很近，线程池中的线程，谁先启动是不定的。”
瞬间觉得这个queue名字有点忽悠人了，还不如pool~
综合一点，我们知道他可以比较大的用处在于可以帮组多线程编程就好了。

18.What is lazy loading?
答案：懒汉模式，只在用到的时候才去初始化。
也可以理解成延时加载。
我觉得最好也最简单的一个列子就是tableView中图片的加载显示了。
一个延时载，避免内存过高，一个异步加载，避免线程堵塞。

19.Can we use two tableview controllers on one viewcontroller?
是否在一个视图控制器中嵌入两个tableview控制器?
答案：一个视图控制只提供了一个View视图，理论上一个tableViewController也不能放吧，
只能说可以嵌入一个tableview视图。当然，题目本身也有歧义，如果不是我们定性思维认为的UIViewController，
而是宏观的表示视图控制者，那我们倒是可以把其看成一个视图控制者，它可以控制多个视图控制器，比如TabbarController

那样的感觉。

20.Can we use one tableview with two different datasources? How you will achieve this?
一个tableView是否可以关联两个不同的数据源？你会怎么处理?
答案：首先我们从代码来看，数据源如何关联上的，其实是在数据源关联的代理方法里实现的。
因此我们并不关心如何去关联他，他怎么关联上，方法只是让我返回根据自己的需要去设置如相关的数据源。
因此，我觉得可以设置多个数据源啊，但是有个问题是，你这是想干嘛呢？想让列表如何显示，不同的数据源分区块显示?

这套题目来自cocoachina的yoyokko版 主大大招人时候的题目,论坛各路大神都觉得偏难。自己看了一下，发现很多是自己知道，但又说不上来的感觉。所以觉得有必要梳理完善一下，题很多，反正写到哪算哪吧！另外，因为我不是C/C++ 或写 mac “发家” 所以还是有解答不上来的，但是关于IOS方面，一定尽量解答，如果回答得有不尽入人意的地方，欢迎高手纠正。 下面先看看题目:

1.Object－c的类可以多重继承么？可以实现多个接口么？Category是什么？重写一个类的方式用继承好还是分类好？为什么?

2.#import 跟#include 又什么区别 ＃import<> 跟 #import""又什么区别?

3.类变量的@protected ,@private,@public,@package声明各有什么含义?

4.id 声明的对象有什么特性?

5.MVC是什么？有什么特性？为什么在iPhone上被广泛运用?

6.对于语句NSString* testObject = [[NSData alloc] init];testObject 在编译时和运行时分别时什么类型的对象?

7.什么是安全释放?

8.为什么有些4.0独有的objective－c 函数在3.1上运行时会报错.而4.0独有的类在3.1上分配内存时不会报错？分配的结果是什么?

9.为什么4.0独有的c函数在3.1的机器上运行不会报错（在没有调用的情况下？）而4.0独有的类名在3.1的机器上一运行就报错?

10.异常exception 怎么捕获？不同的CPU结构上开销怎样？C中又什么类似的方法?

11.property中属性retain,copy,assgin的含义分别是什么？有什么区别？将其转换成get／set方法怎么做？有什么注意事项?

12.委托是什么？委托的property声明用什么属性？为什么?

13.浅拷贝和深拷贝区别是什么？...

14.Cocoa中与虚基类的概念么？怎么简洁的实现?

15.自动释放池跟GC有什么区别？iPhone上有GC么？［pool release］和［pool drain］有什么区别?

16.

```
<span style="font-family: 幼圆;">for(int index = 0; index < 20; index +
+){
    NSString *tempStr = @"tempStr";
    NSLog(tempStr);
    NSNumber *tempNumber = [NSNumber numberWithInt:2];
    NSLog(tempNumber);
}
</span>
```

这段代码有什么问题.？会不会造成内存泄露（多线程）？在内存紧张的设备上做大循环时自动释放池是写在循环内好还是循环外好？为什么?

17.内存管理的几条原则时什么？按照默认法则.那些关键字生成的对象需要手动释放？在和property结合

的时候怎样有效的避免内存泄露?

18.在一个对象释放前.如果他被加到了notificationCenter 中.不在notificationcenter中remove这个对象可能会出现什么问题?

19.怎样实现一个 singleton的类.给出思路。

20.什么是序列化或者Acrchiving,可以用来做什么,怎样与copy结合,原理是什么?.

21.线程是什么？ 有哪些注意事项.?

22.在iphone上有两件事情要做,请问是在一个线程里按顺序做效率高还是两个线程里做效率高？为什么?

23.runloop是什么？在主线程中的某个函数里调用了异步函数，怎么样block当前线程,且还能响应当前线程的timer事件，touch事件等.

24.ios平台怎么做数据的持久化?coredata和sqlite有无必然联系？coredata是一个关系型数据库吗?

25.阐述一个nil对象从interface bulider产生，到载入程序运行空间，最后被释放时所经历的生命周期.

26.notification是同步还是异步? kvo是同步还是异步？notification是全进程空间的通知吗？kvo呢?

27.kvc是什么?kvo是什么?有什么特性?

28.响应者链是什么?

29.unix上进程怎么通信?

30.timer的间隔周期准吗？为什么？怎样实现一个精准的timer?

31.UIscrollVew用到了什么设计模式？还能再foundation库中找到类似的吗?

32如果要开发一个类似eclipse的软件，支持插件结构。且开放给第三方开发。你会怎样去设计它？（大概思路）

题目:1.Object－c的类可以多重继承么？可以实现多个接口么？Category是什么？重写一个类的方式用继承好还是分类好？为什么?

关于多继承**:**

首先 object-c不能够多继承,类似下面代码的这种方式是绝对通不过编译的.当然,你也可以把NSString前面的":"去掉再试试，呵呵!

那么有没有别的方式来替代呢？有，一种我们称之为伪继承，另一种我们可以通过ios中无处不在的@protocol委托方式来实现.

1.伪继承

尽管再objtive-C中不提供多继承，但它提供了另外一种解决方案,使对象可以响应在其它类中实现的消息(别的语言中，一般叫方法，两者无差别). 这种解决方案叫做消息转发，它可以使一个类响应另外一个类中实现的消息。

在一般情况下，发送一个无法识别的消息会产生一个运行时的错误，导致应用程序崩溃,但是注意，在崩溃之前，iphone运行时对象为每个对象提供了第二次机会来处理消息。捕捉到一条消息后可以把它重定向到可以响应该消息的对象。

这个功能完全通过消息转发来实现，发送消息给一个无法处理该选择器的对象时，这个选择器就会被转发给 forwardInvocation 方法.接收这条消息的对象，用一个NSInvocation的实例保存原始的选择器和

被请求的参数.所以，我们可以覆盖 forwardInvocation 方法，并把消息转发给另外一个对象.

1.1 实现消息转发功能

在给程序添加消息转发功能以前，必须覆盖两个方法，即methodSignatureForSelector: 和 forwardInvocation:。methodSignatureForSelector:的作用在于为另一个类实现的消息创建一个有效的方法签 名。forwardInvocation:将选择器转发给一个真正实现了该消息的对象.

例子:

1.

```
1 - (NSMethodSignature*)methodSignatureForSelector:(SEL)selector
2 {
3     NSMethodSignature* signature = [super
methodSignatureForSelector:selector];
4
5     if (!signature)
6         signature = [self.carInfo methodSignatureForSelector:selector];
7
8     return signature;
9 }
```

2.

```
1 - (void)forwardInvocation:(NSInvocation *)invocation
2 {
3     SEL selector = [invocation selector];
4
5     if ([self.carInfo respondsToSelector:selector])
6     {
7         [invocation invokeWithTarget:self.carInfo];
8     }
9 }
```

3.调用

```
1 Car *myCar = [Car car]; //Car为一个类
2 [(NSString *)myCar UTF8String]  //这里调用NSString中的UTF8String方,注意Car中
并未实现该方法
```

解释: 这里借iphone开发秘籍的例子来说明, self.carInfo是一个只读的NSString对象,存在于Car类中.例子中Car实例是无法正确的为另外一个对象(NSString)实现的 选择器创建一个有效的签名。运行时当检查到当前没有有效的签名,即进入该对象(这里是myCar)的 methodSignatureForSelector:方法中,此时，将在这个方法中对每个伪继承进行迭代并尝试构建一个有效的方法签名的机会.例如代 码中,当myCar调用UTF8String时,由于无法从当前对象中获得消息，转入第二次机会捕捉消息，首先进入methodSignatureForSelector:方法,采用迭代的方式为当前被调用的方法创建一个有效的签名，得到签名后，转入 forwardInvocation:方法对其调用的方法(UTF8String)进行实现. forwardInvocation:中，首先获得调用的方法(UTF8String),判断self.carInfo(一个nsstring对象)能 否响应该方法，如果可以，将调用UTF8String对象的目标转换为self.carInfo对象. 这样 ，我们就实现了多继承，呵呵！！

注:如果您仍有疑问，可访问苹果的官方文档查询消息转发相关内容:
地址 http://www.apple.com.cn/developer/mac/library/documentation/Cocoa/Conceptual/ObjCRuntimeGuide/Articles/chapter_6_section_1.html#//
apple_ref/doc/uid/TP40008048-CH105-SW1

2.委托

在IOS中委托通过一种@protocol的方式实现,所以又称为协议.协议是多个类共享的一个方法列表,在协议中所列出的方法没有响应的实现，由 其它人来实现.这叫好比我想买个手机，所以我有个buyIphone 方法,但是我不知道谁那买手机,所以把这个需求发布出去(比如公布在网站上),如果有卖手机的商人(也就是说他能实现buyIphone这个方法)看到， 他就会接受我的委托,(在商人自己的类中实现<XXXdelegate>),那么我的委托对象就指向了这个商人..当我要买手机的时候，直接 找他就行了.

例如:

```
@protocol MyDelegate
-(void)buyIphone:(NSString *)iphoneType money:(NSString *)money;

@end
@interface My : NSObject
{
    id<MyDelegate> deleage;
}
@property(assign,nonatomic)id<MyDelegate> delegate;
@end
```

代码中声明了一个协议 名叫Mydelegate,在其中有一个buyIphone方法，即一个委托项。当我要购买手机的时候只需要通过delegate 调用 BuyIphone方法即可.

如下:

```
-(void)willbuy
{
    [delegate buyIphone:@"iphone 4s" money:@"4888"];

}
```

我不必关心谁现实了这一委托，只要实现了这个委托的类，并且buyIphone是声明的委托中必须实现的方法，那么就一定能够得到结果.

例如:商人类实现了这一委托(用<Mydelegate>表示实现)

```
#import <Foundation/Foundation.h>
#import "My.h"
@interface Business : NSObject<MyDelegate>

@end
```

然后在 @implementation Business 中调用 buyIphone方法

```
#import "Business.h"

@implementation Business

-(void)buyIphone:(NSString *)iphoneType money:(NSString *)money
{
    NSLog(@"手机有货，这个价钱卖你了，发货中!!");
}
@end
```

就ok啦。这样是不是也模拟了多继承呢？通过委托，其实你也就摆脱了去考虑多继承方面的事情，从而关注当前类。

[疯狂的蛋蛋]

亲，开发一个ios应用没那么容易.!

让我们开门见山吧：做一个 iPhone 应用需要花多少钱?

就是这 个最常见的问题，我的很多朋友（大多是些西装革履的商务人士），还有我那些个对技术一知半解的客户们，他们都问过我这个的问题。通常，我会先给出一个大致 的报价，这个报价并没有细致到需要签合同确认每一个功能点的地步。即便是这样，每当的我报价一出口，对方都毫无例外的给惊着了（当然不是因为便宜）。

说实话，我没有狮子大开口。看看 StackOverflow 上这个著名的帖子吧，讨论的是开发Twitterific 这样一款应用需要多少钱，后来讨论范围扩展到开发一个 iOS 应用的合理费用范围。虽然这个帖子是在 2008 年发布的，而帖子的最佳答案是由一名来自 Twitteriffic 的开发人员于 2010 年回答的，但是时至今日，帖子里面讨论的数字仍然是很靠谱的，而且我预计到 2012 年底依然有效。而我的报价和这个帖子里面的数字比起来，简直是小巫见大巫了。

现在的趋势是，什么公司什么业务都想搞个 iOS 客户端，并且这种趋势在 2012 年看似依然火爆。所以我想起来写这篇博文，我想说一下开发一个 iOS 应用会碰到的各种细节问题和横生的变数，借此解释为什么 iOS 应用开发成本这么贵。如果你在考虑搞一个 iOS 应用，而你本身是搞业务而不是做技术的，如果你目前正在招标或者仅仅是想了解一下，那我这篇博会对你有帮助。当然，我说的东西并不局限于 iOS 应用开发，对 Android、Windows Phone 或者是 Blackberry（如果 RIM 还能活的话）等移动应用平台基本上也是适用的。

开发之前需要仔细考虑的

别做拍脑瓜的决策，在开工之前你需要考虑的比你想象的要多。我通常会帮助或者指导客户把以下几个要素都过一遍：

一：和客户谈他们的移动应用，最让我吃惊的是他们从来没有想过支撑一个 iPhone 应用运行，背后需要涉及到的方方面面。他们想象中的 iPhone 是独立存在于这个宇宙的，是如此的简单，以至于他们要我很快就给出一个项目预算报价，而不用讨论诸多细节。我问他们：“你们是否考虑过后台服务器的事情？ 你们的应用需要和后端服务器做数据通讯？ ”什么，听不懂？好吧，我用地球人的语言再把这个问题讲一遍：“你们的应用不是需要用户注册嘛，你们考虑过把用户的数据存放在哪里了吗？我们需要一个地方去 保存这些以后会用到的数据。”第一次碰到这样的客户时，哥简直就怒了。后来我发现这不是客户的错：我是搞编程的，CS 架构对我来说就像吃饭睡觉一样是不假思索的东西，而我的客户尽是些高富帅，他们懂个毛 CS 架构！

所以，如果你不大懂技术，那请仔细听我说：如果你想做的移动应用需要用户注册和登录，或者你想随时控制移动应用的一些输出，甚至是你仅仅是需要一个用户反馈意见调查表这么简单的功能，那么，你得搞一台后端服务器。

二：好了，现在你知道你需要一台后端服务器。同时你还需要想办法让你的 iOS 应用和你的服务器能够对话，就是相互间接收数据什么的。不，这个问题不是简答靠什么标准的即插即用的东东就能解决的，不是你们想象的那样！所有的东西都需 要定制化开发，这就好比发明一门语言：你希望你的服务器和你的应用之间能够通过一种语言沟通，但是你不希望其他人听得懂这门语言。

用行话说这就是制定服务器端 API 接口，或简称 API。这些 API 应该在开发 iPhone 客户端之前就到位了。为什么？因为你必须先规定好一门语言的单词和语法，然后才能用这门语言说话吧！？好了，这就带出了第三点—如何开发这些 API。

三：API 的成功定制是项目成功的一半（反之亦然），所以千万不要掉以轻心。你要考虑你的业务数据模型、业务流程、调用业务需要提供的参数、特定事件发生时数据间该 如何互动等等。简单来说，我们要做的就是开发一个网站，上门跑着你的业务流程，只不过这个网站的所有运行结果都不是通过网页形式展现出来，而是呈现在一行 行的文本和数字中。举个例子：一个登录成功的反馈页面仅仅包含 YES 一个单词。

iPhone 应用需要访问这些预先定义好的接口，并且按预定义格式提供必要的输入（比如用户名和密码），然后要对服务器端的反馈（YES 或者 NO）做出解析处理。所以，没有什么移动应用能够自动的含有用户注册和登录功能。

服务器端开发需要考虑的问题太多了：选择服务器，选择用什么语言开发，主机放在哪里才能增加访问速度，等等，这里我就不展开了。如果这一切对你来说很陌生，那么你最好去问问团队里的技术负

责人，或者干脆让开发人员做决策。

四: 所以，关于服务器端 API，你或者让自己的技术团队把它开发好，再将完善的 API 文档交给 iPhone 应用开发人员；或者你支付 iPhone 应用开发人员额外的报酬来搞定这些。你找的 iPhone 应用开发人员可能会服务器端开发也可能不会。如果他会的话，我建议最好让他也同时负责服务器端开发，因为他最清楚 iPhone 应用中需要哪些服务器端 API。

如果你的服务器端 API 已经存在了，那么除了向 iPhone 应用开发人员提供相关文档之外，你还要考虑让他能够便捷的同服务器开发团队沟通，因为大多数情况下，iPhone 应用需要在已有 API 基础上增加一些新的接口。

现在我们来看看 iPhone 应用开发本身

扯了大半天，我们终于开始谈 iPhone 应用开发本身了。一般来说，iOS 平台上做所有事情都不能随心所欲。你最好在开发人员写代码之前把所有的需求都确认好好。这和开发网站不一样，按照实现签订的合同开发 iOS 应用，开发过程中对需求变更的容纳度可能很低:

用户界面：无论你打算采用 iOS 标准界面还是自定义元素，在开发开始前一定要确认清楚，因为应用的程序架构是根据界面和用户使用流程来设计的。一个很好的例子就是在界面底部使用了 iOS 标准的标签栏（Tab Bar），此后如果你想让标签栏里面的图标变成彩色的，这个代码改动量可没你想象的那么小!

代码之间的耦合： 如果是开发网站，你可以随意的添加一个页面或者一处链接。做 iOS 应用就没有那么简单了，很多东西一开始都要设计好，后期的一处改动会牵连很多东西，具体原因是你无法理解的。iOS 应用的代码写好之后，再改动行不行？行！但必须小心。 这就像设计电路板一样， 如果你不小心把那根线搭错了，整块电路板就会不工作。有人说架构优良的程序可以有很高的延展性，那纯属纸上谈兵。在 About 屏幕上添加一个电子邮件按钮可能只需要几行代码的工作量，而添加一个转发到新浪微薄的按钮（译者注：原文是添加一个 Facebook Like）就完全不是那么简单的事儿了!

让一个 iPhone 应用同时也支持 iPad：如果要评选最坑爹“需求变更”，那么这个绝对是当之无愧的。理由很简单：支持 iPad 根本不是 TMD 什么附加功能！iPad 应用基本上都比 iPhone 应用来得要复杂，界面设计和用户体验也大不一样。我问你，制造一辆电动自行车，然后把它改装成一部烧汽油的摩托车，这能是一回事儿吗！？ 电动自行车跟摩托 车看起来是很像，但是制造它们完全是两码事。

拿广受欢迎的 Facebook 官方应用来说，它的 iPhone 和 iPad 版本看似相似，实际用户操作流程完全不同。不仅仅是界面上的不同会带来额外的工作，对后台服务器 API 的需求也可能不一样。拿我熟悉的一个应用 Denso 来说（我熟悉它因为这是我开发的），它的 iPad 版本比 iPhone 多了几个功能，这些都需要额外的服务器端 API 来支持。记住，iPhone 和 iPad 应用的用户体验需求是完全不一样的。

准备好开始了吗?

希望此文能够帮助你和你的团队了解移动应用开发幕后的方方面面。除非你们要做一个像计算器那么简单的单机应用，否则你们很难用极低的成本搞定。综上所述，如果你觉得外包成本太高，那你只好招人自己开发。

当然，如果你决定了要外包移动应用开发，那么我还要提醒一点：公司政治。如果你是在一家大公司或者有着严格制度的机构里面干活，那么帮助合同开发者搞定 那些个规章制度上的繁文缛节，对你来说是非常重要的一项工作，必要的时候甚至可以做一些政策上的变通。 我同几个大型企业客户接触过，当我要求看他们的服务器端数据接口的时候，他们流露出很不安的表情。我想这或许是因为他们受制于公司规定而不能透露信息，这 无可厚非；或者他们还没有想好这种情况下该如何操作；或者他们的品牌制度蛋疼到需要在移动应用的每个屏幕上都摆着公司 logo！最终我没有和这样的企业客户合作，因为我无法想象如果有一天我需要增加一些服务器端 API 接口的话，和他们的规章和流程折腾，那将会是多么悲剧的事情。

PS：开发移动应用很耗费时间，你最好有耐心。

11月 15 日消息，谷歌公司的面试题在刁钻古怪方面相当出名，科技博客 BusinessInsider 贴出了 15 道谷歌面试题，并一一给出了答案。

第一题：多少只高尔夫球才能填满一辆校车？（职位：产品经理）

解析：通过这道题，谷歌希望测试出求职者是否有能力判断出解决问题的关键。

网友的答案：我想，一辆标准大小的校车约有 8 英尺宽、6英尺高、20英尺长——我能知道这些数字完全是因为我曾经无数次被堵在校车后面。

据此估算，一辆校车的容积约为 960 立方英尺，也就是 160 万立方英寸。一个高尔夫球的半径约为0.85英寸，我认为一个高尔夫球的体积约为2.6立方英寸。

用校车的容积除以高尔夫球的体积，得到的结果是 66 万。不过，由于校车里面还有座位等等各种东西，而且高尔夫球的形状使得不同的球之间会有不少空隙。我的最终估算结果是 50 万。这听起来有

些荒唐。如果我直接猜的话，我给出的答案肯定是 10 万以下，不过我相信我的数学水平。

当然，如果这里的校车是小布什当年坐过的那种，结果还要除以2，差不多是 25 万个。

第二题：让你清洗西雅图所有的玻璃窗，你的报价是多少？（职位：产品经理）

答案：这一题我们可以玩点花招，我们的答案是“每扇窗 10 美元”。

第三题：有一个人们只想生男孩子的国家，他们在有儿子之前都会继续生育。如果第一胎是女儿，他们就会继续生育直到有一个儿子。这个国家的男女儿童比例是多少？（职位：产品经理）

答案：这一题引发了不少争议，不过我们发现，这一题的解答步骤如下：

1、假设一共用 10 对夫妻，每对夫妻有一个孩子，男女比例相等。（共有 10 个孩子，5男 5 女）；

2、生女孩的 5 对夫妻又生了 5 个孩子，男女比例相等。（共有 15 个孩子，男女儿童都是7.5个）；

3、生女孩的2.5对夫妻又生了2.5个孩子，男女比例相等。（共有 17.5 个孩子，男女儿童都是8.75个）；

4、因此，男女比例是1:1。

第四题：全世界共有多少名钢琴调音师？（职位：产品经理）

答案：我们的回答是“要看市场情况。如果钢琴需要每周调音一次，每次调音需要 1 个小时，且每个调音师每周工作 40 个小时。我们认为每 40 台钢琴就需要一名调音师。”

这个问题又被称为“费米问题”（Fermi problem）。费米提出的问题是“在芝加哥有多少钢琴调音师”。一个典型的答案是包括一系列估算数据的乘法。如果估计正确，就能得到正确答案。比如我们采用如下假设：

芝加哥约有 500 万人居住；

平均每个家庭有 2 人；

大约有1/20的家庭有定期调音的钢琴；

平均每台钢琴每年调音一次；

每个调音师调整一台钢琴需要 2 小时；

每个调音师每天工作 8 小时、每周 5 天、每年 50 周。

通过这些假设我们可以计算出每年在芝加哥需要调音的钢琴数量是：

(芝加哥的 500 万人口)/(2人/家)×(1架钢琴/20家)×(1架钢琴调整/1年)=125000

平均每个调音师每年能调整的钢琴数量是：

(50周/年)×(5天/周)×(8小时/天)/(1架钢琴/2小时)=1000

芝加哥的调音师数量是：

(芝加哥需要调音的钢琴数量 125,000)/(每个调音师每年能调整的钢琴数量 1000)=125

第五题：马路上的井盖为什么是圆的？（职位：软件工程师）

答案：圆形的井盖在任何角度都不会掉下去。

第六题：为旧金山市设计一个紧急撤离方案（职位：产品经理）

答案：这又是一个考察求职者是否能够发现问题核心的题目。我们在回答之前首先要问的是，“撤离方案应对的是什么样的灾难”。

第七题：一天之中，时钟的时针和分钟会重合几次？（职位：产品经理）

答案：**22**次。

重合的时间点分别是：上午，12:00、1:05、2:11、3:16、4:22、5:27、6:33、7:38、8:44、9:49、10:55；下午 12:00、1:05、2:11、3:16、4:22、5:27、6:33、7:38、8:44、9:49、10:55。

第八题：请阐述**“Dead beef”**的意义。（职位：软件工程师）

答案：网友给出的正确答案是，在大型机和汇编语言的时代，“DEADBEEF”是调试计算机时所使用的一个十六进制值，以便于在大量的十六进制中断信息中标记和查找特定的内存数据。大多数计算机科学专业毕业生都应该会在汇编语言的课程上见过这个概念。

第九题：有人把车停在旅馆外，丢失了他的财物，他接下来会干什么？（职位：软件工程师）

答案：下车踏到人行道上。

第十题：你需要确认朋友鲍勃是否有你正确的电话号码，但不能直接问他。你须在一张卡片上写下这个问题，然后交给爱娃，由爱娃把 卡片交给鲍勃，再转告你答案。除了在卡片上写下这个问题外，你还必须怎样写，才能确保鲍勃在给出答案的同时，不让爱娃知道你的电话号码？（职位：软件工程师）

答案：既然只需要核对鲍勃手中的号码是否正确，你只需要让他在某个特定的时刻给你打电话，如果他没打过来的话，就能确认他没有你的号码。

第十一题：假设你是海盗船的船长，船员们即将对黄金的分配方案投票。如果赞成票不到半数的话，你会被杀死。你怎样才能在保证自己存活的情况下拿到最多的黄金？（职位：软件工程师）

答案：将黄金平均分给最有权势的 51% 的船员。

第十二题：有八个大小相等的球，其中有一个重量比其他球略重。如何在只用天平称两次的情况下找出那个不一样的球？（职位：产品经理）

答案：从八个球中取出六个，在天平两边各放三个。如果平衡，把剩下的两个球分别放在天平两边，就能找出较重的球。如果不平衡，较重的球就在天平下沉的一边，从这三个当中取出两个称量，若不平衡，下沉的一边较重，若平衡，剩下的就是较重的球。

第十三题：你拿着两个鸡蛋站在 **100** 层的大楼上。鸡蛋或许结实到从楼顶掉下也不会摔破，或许很易碎，在一楼摔下就破碎。最少试验多少次可以找出鸡蛋不会被摔碎的最高楼层？（职位：产品经理）

答案：14次。从 14 楼丢下第一颗鸡蛋，如果破碎了就逐层往下试验，共需 14 次。如果没有破碎，往上走 13 层；在 27 楼第二次丢下第一颗鸡蛋，如果碎了，换第二颗鸡蛋往上走 12 曾测试，若仍没碎，往上走 12 层试验第一颗鸡蛋；以此类推，直到走到第 99 层。如果鸡蛋要到 100 层高度落下才会破

碎，总共需要 14 次尝试。

第十四题：如果用三句话向你 **8** 岁大的侄子解释什么叫数据库？（职位：产品经理）

答案：这一题考察的是求职者用简单的语言阐述复杂概念的能力。我们的答案是“数据库是一个能够记住关于很多东西的很多信息的机器。人们用它来帮助记住这些信息。出去玩吧。”

第十五题：你被缩小到只有硬币厚度那么点高，然后被扔到一个空的玻璃搅拌机中，刀片一分钟后就开始转动。你会怎么做？（职位：产品经理）

答案：这一题考察的是求职者的创造性。我们会尝试把电动机弄坏。

1 面试的目的
求职者通过表现证明自己对岗位的胜任
公司通过面试找到符合职位需求的员工
面试者面试的表现影响着公司用人选择，对于软件工程师，我的感觉技术面试往往是“天王山”之战，过去了BOSS面的时候，刷人机率不高，过不去，就得要找新的工作了。

2 面试的准备
现在大多数人对面试都挺重视的，我觉得也不应该简单拒绝面试宝典类的东西（这片文章也是这一类的），感觉软件开发过程 涉及很多方面，很难在短时间内对一个人完成全面的评估，举个例子，比如高考，本来的目的是通过考试依靠分数选拔优秀学生去好大学，能力是基础，但衡量是成 绩，那学生的最好方法，是根据考试来学习，这是个相对简单的方法，然后就有了应试教育。工作面试也一样，个人的能力、工作背景、项目经验是基本，面试的技 巧是应试技巧，面试技巧是表面文章，就像皮之不存，毛将焉附一样？又不得不说的是谁都喜欢毛色靓丽的皮草。

着装：
小伙子干净利索 姑娘 随意打扮，别浓妆艳抹就行

知识点：
这里我只列一些iOS的主要知识点：
objective -C 部分
cocoaTouch 框架部分
XCODE 使用部分
项目经验部分 前三部分的内容，基本是按面试官考察面试者的知识点的掌握情况，唯独项目经验，是面试者向面试官展示，可以提前练习下项目介绍，做到有层次，有重点（根据 不同的职位有不同的重点），例如，作为软件开发人员参与了iPad的点餐系统开发，完成了图片菜单显示的代码，遇到了scroll view显示大图片效率问题，用懒加载的方式解决了该问题。

3 面试的过程
实事求是的答题
刚毕业求职时，特害怕一道题目打不出来就直接被pass掉，这也是新手求职的过 程，确实会遇到，如果是特别基础题，建议回去加强基础知识。但对于一般面试，一两道题回答不出来，是非常正常的，在这样技术信息不断更新的时代，在牛的人 技术都不能面面俱到。以我自己面试别人的经验，senior些的面试官，都会了解面试人员肯定有一些问题不了解，所以会从各个方面的问题都会涉及到，然后 对面试者做一个综合评价。对于面

试过程中，遇到自己不熟悉的领域，一定要实事求是，不了解就是不了解，了解一点就说一点，一定不要知道一点就装资深，上来 就是：这个知道，简单的很，哪个做过，不复杂。然后面试官继续深入的问些细的技术点，就开始找理由："这个做太久了，那个模块是别人实现的"，这倒不要上 升到道德诚信，因为面试过程总会有一些表面上的东西，从面试官的角度来说，首先他能面试你，一般是比你资深，其次面试的问题，面试官一定挑选过，所以在这 上面抱着蒙混过关的心理，是有点天真的。在自己擅长的技术点与面试官进行深层次的沟通，能得到加分，技术点的知识都是可以再学的，对于问题的抽象深度，往 往决定一个程序员解决问题的能力。

答题的态度要谦虚
有些人不能说技术不强，对于知道的便唯他独尊，不知道就觉不重要，视野局限，例如一个iPhone程序员之前项目做的都是Native App，碰到面试官问他：HTML Hybrid框架的一些东西？ 就显出一副不屑一顾的轻视，说Html 5做出来的界面显示速度慢，都是垃圾，先不说Html 5在跨平台上的优势和已有网站业务的移动化升级等，面试官的问题很有可能是他的项目涉及这方面的技术，他是有主观感受的，主观上会如何评价这位面试者？如 俗语所说：“满招损，谦得益”，谦虚的人，在项目中的团队合作也会遇到较少障碍。

4 面试的心态
说下面试的心态，有些面试者，已经面试场特紧张，有的甚至声音会发颤，这是很影响发挥的。对于有这样的问题的，往往太想要这份工作，造成紧张过度，但生活中不止有一次机会，而即使这次面试很成功，也有可能最终拿不到offer，一份工 作不全由一次面试决定，一个人的人生也不全由一份工作决定。现在社会发展越来越快，一次失败往往是下一次工作机会的开始，所以去尝试，总会有新的、更好的 机会。

5 面试的后续
什么样面试是较成功的？我的理解，首先是面试官对你有兴趣，表现就是回答面试官问题时，他听的很仔细，听完你的回答后，面试官还会讲一些他对这个问题的看法，这样的互动就很成功。

面试结束便是学习的一个新开端，不论是否拿到最终的offer，都已经花了时间去面试，对于面试中的问题的总结，就非常必要了。一些没有答出来的题目，可以在网上查查资料，把不清楚的问题搞清楚，提高个人能力。
随着iOS平台开发的职位的增加，笔试、面试也越来越有“套路”，这里我总结了一些面试题，多数是Objective-C的基础知识，适合于面试新人，答案是我自己答的，不准确的地方，欢迎指出。

1．Object－c的类可以多重继承么？可以实现多个接口么？Category是什么？重写一个类的方式用继承好还是分类好？为什么？

Object-c的类不可以多重继承；可以实现多个接口，通过实现多个接口可以完成C++的多重继承；Category是类别，一般情况用分类好，用Category去重写类的方法，仅对本Category有效，不会影响到其他类与原有类的关系。

2.#import 跟#include 又什么区别，@class呢，＃import<> 跟 #import""又什么区别？

#import是Objective-C导入头文件的关键字，#include是C/C++导入头文件的关键字,使 用#import头文件会自动只导入一次，不会重复导入，相当于#include和#pragma once；@class告诉编译器某个类的声明，当执行时，才去查看类的实现文件，可以解决头文件的相互包含；#import<>用来包含系 统的头文件，

#import""用来包含用户头文件。

3. 属性readwrite，readonly，assign，retain，copy，nonatomic 各是什么作用，在那种情况下用?

readwrite 是可读可写特性；需要生成getter方法和setter方法时

readonly 是只读特性 只会生成getter方法 不会生成setter方法 ;不希望属性在类外改变

assign 是赋值特性，setter方法将传入参数赋值给实例变量；仅设置变量时；

retain 表示持有特性，setter方法将传入参数先保留，再赋值，传入参数的retaincount会+1;

copy 表示赋值特性，setter方法将传入对象复制一份；需要完全一份新的变量时。

nonatomic 非原子操作，决定编译器生成的setter getter是否是原子操作，atomic表示多线程安全，一般使用nonatomic

4.写一个setter方法用于完成@property （nonatomic,retain）NSString *name,写一个setter方法用于完成@property（nonatomic，copy）NSString *name

[cpp] view plaincopyprint?

```
1.  - (void) setName:(NSString*) str
2.  {
3.      [str retain];
4.      [name release];
5.      name = str;
6.  }
7.  - (void)setName:(NSString *)str
8.  {
9.      id t = [str copy];
10.     [name release];
11.     name = t;
12. }
```

5.对于语句NSString*obj = [[NSData alloc] init]; obj在编译时和运行时分别时什么类型的对象?

编译时是NSString的类型；运行时是NSData类型的对象

6.常见的object-c的数据类型有那些， 和C的基本数据类型有什么区别？如：NSInteger和int

object-c的数据类型有NSString，NSNumber，NSArray，NSMutableArray，NSData等等，这些都是class，创建后便是对象，而C语言的基本数据类型int，只是一定字节的内存空间，用于存放数值；NSInteger是基本数据类型，并不是NSNumber的子类，当然也不是NSObject的子类。NSInteger是基本数据类型Int或者Long的别名（NSInteger的定义typedef long NSInteger），它的区别在于，NSInteger会根据系统是32位还是64位来决定是本身是int还是Long。

7.id 声明的对象有什么特性?

Id 声明的对象具有运行时的特性，即可以指向任意类型的objcetive-c的对象；

8.Objective-C如何对内存管理的,说说你的看法和解决方法?

Objective-C的内存管理主要有三种方式ARC（自动内存计数）、手动内存计数、内存池。

9.内存管理的几条原则时什么？按照默认法则.那些关键字生成的对象

需要手动释放？在和property结合的时候怎样有效的避免内存泄露?

谁申请，谁释放

遵循Cocoa Touch的使用原则；

内存管理主要要避免“过早释放”和“内存泄漏”，对于“过早释放”需要注意@property设置特性时，一定要用对特性关键字，对于“内存泄漏”，一定要申请了要负责释放，要细心。

关键字alloc 或new 生成的对象需要手动释放；

设置正确的property属性，对于retain需要在合适的地方释放，

10.如何对iOS设备进行性能测试?

Profile-> Instruments ->Time Profiler

11.看下面的程序,第一个NSLog会输出什么？这时str的retainCount是多少？第二个和第三个呢？ 为什么?

[cpp] view plaincopyprint?

```
1.  ============================================================
2.  NSMutableArray* ary = [[NSMutableArray array] retain];
3.  NSString *str = [NSString stringWithFormat:@"test"];
4.  [strretain];
5.  [aryaddObject:str];
6.  NSLog(@"%@%d",str,[str retainCount]);
7.  [strretain];
8.  [strrelease];
9.  [strrelease];
10. NSLog(@"%@%d",str,[str retainCount]);
11. [aryremoveAllObjects];
12. NSLog(@"%@%d",str,[str retainCount]);
13. ============================================================
```

str的retainCount创建+1，retain+1，加入数组自动+1

3

retain+1，release-1，release-1

2

数组删除所有对象，所有数组内的对象自动-1

1

12. Object C中创建线程的方法是什么？如果在主线程中执行代码，方法是什么？如果想延时执行代码、方法又是什么?

线程创建有三种方法：使用NSThread创建、使用 GCD的dispatch、使用子类化的NSOperation,然后将其加入NSOperationQueue;在主线程执行代码，方法是performSelectorOnMainThread，如果想延时执行代码可以用performSelector:onThread:withObject:waitUntilDone:

13.描述一下iOS SDK中如何实现MVC的开发模式

MVC是模型、试图、控制开发模式，对于iOS SDK，所有的View都是视图层的，它应该独立于模型层，由视图控制层来控制。所有的用户数据都是模型层，它应该独立于视图。所有的ViewController都是控制层，由它负责控制视图，访问模型数据。

1.When to use NSMutableArray and when to use NSArray?

什么时候使用NSMutableArray，什么时候使用NSArray?

答案：当数组在程序运行时，需要不断变化的，使用NSMutableArray，当数组在初 始化后，便不再改变的，使用NSArray。需要指出的是，使用NSArray只表明的是该数组在运行时不发生改变，即不能往NSAarry的数组里新增 和删除元素，但不表明其数组内的元素的内容不能发生改变。NSArray是线程安全的，NSMutableArray不是线程安全的，多线程使用到 NSMutableArray需要注意。

2.Give us example of what are delegate methods and what are data source methods of uitableview.

给出委托方法的实例，并且说出UITableVIew的Data Source方法

答案：CocoaTouch框架中用到了大量委托，其中 UITableViewDelegate就是委托机制的典型应用，是一个典型的使用委托来实现适配器模式，其中UITableViewDelegate协 议是目标，tableview是适配器，实现UITableViewDelegate协议，并将自身设置为talbeview的delegate的对象， 是被适配器，一般情况下该对象是UITableViewController。

UITableVIew的Data Source方法有- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section;

- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath;

3.How many autorelease you can create in your application? Is there any limit?

在应用中可以创建多少autorelease对象，是否有限制?

答案：无

4.If we don’t create any autorelease pool in our application then is there any autorelease pool already provided to us?

如果我们不创建内存池，是否有内存池提供给我们?

答案:界面线程维护着自己的内存池，用户自己创建的数据线程，则需要创建该线程的内存池

5.When you will create an autorelease pool in your application?

什么时候需要在程序中创建内存池?

答案：用户自己创建的数据线程，则需要创建该线程的内存池

6.When retain count increase?

什么时候内存计数会增加?

答案：见iOS面试题（一）

7.What are commonly used NSObject class methods?

类NSObject的那些方法经常被使用?

答案：NSObject是Objetive-C的基类，其由NSObject类及一系列协议构成。

其中类方法alloc、class、 description 对象方法init、dealloc、– performSelector:withObject:afterDelay:等经常被使用

8.What is convenience constructor?

什么是简便构造方法?

答案：简便构造方法一般由CocoaTouch框架提供，如NSNumber
的 + numberWithBool: + numberWithChar: + numberWithDouble: + numberWithFloat: + numberWithInt:

Foundation下大部分类均有简便构造方法，我们可以通过简便构造方法，获得系统给我们创建好的对象，并且不需要手动释放。

9.How to design universal application in Xcode?

如何使用Xcode设计通用应用?

答案：使用MVC模式设计应用，其中Model层完成脱离界面，即在Model层，其是可运 行在任何设备上，在controller层，根据iPhone与iPad（独有UISplitViewController）的不同特点选择不同的viewController对象。在View层，可根据现实要求，来设计，其中以xib文件设计时，其设置其为universal。

10.What is keyword atomic in Objective C?

在Objetive-C什么时原子关键字

答案：atomic，nonatomic见iOS面试题（一）

11.What are UIView animations?

UIView的动画效果有那些?

答案：有很多，如 UIViewAnimationOptionCurveEaseInOut UIViewAnimationOptionCurveEaseIn UIViewAnimationOptionCurveEaseOut UIViewAnimationOptionTransitionFlipFromLeft UIViewAnimationOptionTransitionFlipFromRight UIViewAnimationOptionTransitionCurlUpUIViewAnimationOptionTransitionCurlDown

如何使用可见该博文

12.How can you store data in iPhone applications?

在iPhone应用中如何保存数据?

答案：有以下几种保存机制：

1.通过web服务，保存在服务器上

2.通过NSCoder固化机制，将对象保存在文件中

3.通过SQlite或CoreData保存在文件数据库中

13.What is coredata?

什么是coredata?

答案：coredata是苹果提供一套数据保存框架，其基于SQlite

14.What is NSManagedObject model?

什么是NSManagedObject模型?

答案：NSManagedObject是NSObject的子类 ，也是coredata的重要组成部分，它是一个通用的类,实现了core data 模型层所需的基本功能，用户可通过子类化NSManagedObject，建立自己的数据模型。

15.What is NSManagedobjectContext?

什么是NSManagedobjectContext?

答案：NSManagedobjectContext对象负责应用和数据库之间的交互。

16.What is predicate?

什么是谓词?

答案：谓词是通过NSPredicate，是通过给定的逻辑条件作为约束条件，完成对数据的筛选。

```
predicate = [NSPredicate predicateWithFormat:@"customerID == %d",n];
```

a = [customers filteredArrayUsingPredicate:predicate];

17.What kind of persistence store we can use with coredata?

使用coredata有哪几种持久化存储机制?

答案：无

1 谈谈对Block 的理解？并写出一个使用Block执行UIVew动画?
答案：Block是可以获取其他函数局部变量的匿名函数，其不但方便开发，并且可以大幅提高应用的执行效率（多核心CPU可直接处理Block指令）

[cpp] view plaincopyprint?

```
1.  [UIView transitionWithView:self.view
2.                    duration:0.2
3.                     options:UIViewAnimationOptionTransitionFlipFromLeft
4.
                  animations:^{ [[blueViewController view] removeFromSuperview]; [[self view] insertSubvie
    w:yellowViewController.view atIndex:0]; }
5.                  completion:NULL];
```

2 写出上面代码的Block的定义。

答案：

typedef void(^animations) (void);
typedef void(^completion) (BOOL finished);

3 试着使用+ beginAnimations:context:以及上述Block的定义，写出一个可以完成

+ (void)transitionWithView:(UIView *)view duration:(NSTimeInterval)duration options:(UIViewAnimationOptions)options animations:(void (^)(void))animations completion:(void (^)(BOOL finished))completion NS_AVAILABLE_IOS(4_0);操作的函数执行部分
答案：无
网络部分

3 做过的项目是否涉及网络访问功能，使用什么对象完成网络功能?

答案：ASIHTTPRequest与NSURLConnection

4 简单介绍下NSURLConnection类及
`+ sendSynchronousRequest:returningResponse:error:`与`– initWithRequest:delegate:`两个方法的区别?

答案：NSURLConnection主要用于网络访问，其中
`+ sendSynchronousRequest:returningResponse:error:`是同步访问数据，即当前线程会阻塞，并等待request的返回的response，而`– initWithRequest:delegate:`使用的是异步加载，当其完成网络访问后，会通过delegate回到主线程，并其委托的对象。

多线程部分

1 什么是block

对于闭包（block), 有很多定义，其中闭包就是能够读取其它函数内部变量的函数，这个定义即接近本质又较好理解。对于刚接触Block的同学，会觉得有些绕，因为我们习惯写这 样的程序main(){ funA();} funA(){funB();} funB(){.....}; 就是函数main调用函数A，函数A调用函数B... 函数们依次顺序执行，但现实中不全是这样的，例如项目经理M，手下有3个程序员A、B、C，当他给程序员A安排实现功能F1时，他并不等着A完成之后，再 去安排B去实现F2，而是安排给A功能F1，B功能F2，C功能F3，然后可能去写技术文档，而当A遇到问题时，他会来找项目经理M，当B做完时，会通知 M，这就是一个异步执行的例子。在这种情形下，Block便可大显身手，因为在项目经理M，给A安排工作时，同时会告诉A若果遇到困难，如何能找到他报告 问题（例如打他手机号），这就是项目经理M给A的一个回调接口，要回掉的操作，比如接到电话，百度查询后，返回网页内容给A，这就是一个Block，在M 交待工作时，已经定义好，并且取得了F1的任务号（局部变量），却是在当A遇到问题时，才调用执行，跨函数在项目经理M查询百度，获得结果后回调该 block。

2 block 实现原理
Objective-C是对C语言的扩展，block的实现是基于指针和函数指针。
从计算语言的发展，最早的goto，高级语言的指针，到面向对象语言的block，从机器的思维，一步步接近人的思维，以方便开发人员更为高效、直接的描述出现实的逻辑（需求）。
下面是两篇很好的介绍block实现的博文
iOS中block实现的探究
谈Objective-C Block的实现

3 block的使用
使用实例
cocoaTouch框架下动画效果的Block的调用

使用typed声明block
typedef void(^didFinishBlock) (NSObject *ob);
这就声明了一个didFinishBlock类型的block，
然后便可用
@property (nonatomic,copy) didFinishBlock finishBlock;
声明一个blokc对象，注意对象属性设置为copy，接到block 参数时，便会自动复制一份。

__block是一种特殊类型，
使用该关键字声明的局部变量，可以被block所改变，并且其在原函数中的值会被改变。

4 常见系列面试题
面试时，面试官会先问一些，是否了解block，是否使用过block，这些问题相当于开场白，往往是下面一系列问题的开始，所以一定要如实根据自己的情况回答。
1 使用block和使用delegate完成委托模式有什么优点?

首先要了解什么是委托模式，委托模式在iOS中大量应用，其在设计模式中是适配器模式中的对象适配器，Objective-C中使用id类型指向一切对象，使委托模式更为简洁。了解委托模式的细节：
iOS设计模式----委托模式
使用block实现委托模式，其优点是回调的block代码块定义在委托对象函数内部，使代码更为紧凑；
适配对象不再需要实现具体某个protocol，代码更为简洁。

2 多线程与block
GCD与Block
使用 dispatch_async 系列方法，可以以指定的方式执行block
GCD编程实例

dispatch_async的完整定义
```
  void dispatch_async(
  dispatch_queue_t queue,
  dispatch_block_t block);
```
功能：在指定的队列里提交一个异步执行的block，不阻塞当前线程

通过queue来控制block执行的线程。主线程执行前文定义的 finishBlock对象
```
dispatch_async(dispatch_get_main_queue(),^(void){finishBlock();});
```
IOS项目中会用到对通讯录的联系人或是会员按姓名为关键字排序，因为NSArray并不直接支持对汉字的排序，这就要通过将汉字转换成拼音完成按A~Z的排序，这看起来是个头疼的问题，因为牵扯到汉字转为拼音，kmyhy给出一个较易实现的方法，获取汉字的首字的首字母，如将“王”变成“W”,完整文章（传送门）。

其中他通过pinyinFirstLetter函数获取中文拼音，函数原理是：“pinyinFirstLetter基于这么一个简单的原理：我们知道， 在Objective C语言中，字符串是以unicode进行编码的。在unicode字符集中，汉字的编码范围为4E00 到 9FA5 之间（即从第19968开始的20902个字符是中文简体字符）。我们把这些字符的拼音首字母按照顺序都存放在一个char数组中。当我们查找一个汉字的 拼音首字母时，只需把这个汉字的unicode码（即char强制转换为int）减去19968，然后用这个数字作为索引去找char数组中存放的字母即 可。”函数代码我也贴了过来。

[cpp] view plaincopyprint?

```
1.  char pinyinFirstLetter(unsignedshort hanzi)
2.  {
3.    int index = hanzi - HANZI_START;
4.    if (index >= 0&& index <= HANZI_COUNT)
5.   {
6.    return firstLetterArray[index];
7.   }
8.   else
9.   {
10.   return hanzi;
11.  }
12. }
```

这个方法真是用很原理的东西，解决一个巧妙问题，但也存在一个问题：不支持汉字（首字）的第二个字母的排序，据说映射的char数组会很大。这里我给出 我的方法，先

在CocoaTouch框架上看看有什么能用到的，在NSString有一个函数localizedCompare:,它的功能是通过自身与 给定字符串的比较，返回一个本地化的比较结果，也就是说这个函数是支持汉字比较的。

进一步localizedCompare:只是字符串与字符串的比较，我们要让NSArray数组通过按关键字为汉字字符串排序，我们继续在 NSArray的SDK上下功夫，我们发现除了sortdArrayUsingDescriptors:通过NSSortDescriptor排序；还有sotedArrayUsingFunction:context:,这个函数支持我们自定义一个函数制定比较规则，返回比较结果，这样问题一下便解决 了。我先把代码贴出来。

[cpp] view plaincopyprint?

```
1.  NSInteger nickNameSort(id user1, id user2, void *context)
2.  {
3.      User *u1,*u2;
4.  //类型转换
5.      u1 = (User*)user1;
6.      u2 = (User*)user2;
7.      return  [u1.nickName localizedCompare:u2.nickName
8.  ];
9.  }
```

在需要比较地方，调用比较函数

[cpp] view plaincopyprint?

```
1.  sortArr = [arr sortedArrayUsingFunction:nickNameSort context:NULL];
```

需要的代码很少吧，这样做会有几方面的好处：1 支持多个汉字按字母序排序（若第一个字的第一个字母相同，则按第一个字的第二个字母比较，若第一个字的字母完全相同，按第二个字的首字母继续排序）。 2 原本可能需要保存汉字拼音的地方，现在不需要了。 3 可以通过对nickNameSortde进一步定制，完成更复杂的比较，比如先比较会员状态，在按姓名字母序完成比较。4 整体结构简单 使用的都是CocaTouch框架下的的方法。

再感叹一句，CocaTouch你真厉害!!

iOS APP 开发流程

分类： IOS开发 2013-03-09 14:57 785人阅读 评论(1) 收藏 举报

iOSApp

挺麻烦的，我我说一下大致流程：

你必须有一个visa信用卡，支持国际支付的。

然后到苹果开发者网站，注册一个账号，这一步是免费的。

然后登陆，点enroll，会进入一个流程，填写你要加入的开发计划，个人信息，信用卡信息等。由于中国不支持直接支付，这个流程会给你一个pdf 表单，打印填完签字，扫描后发到苹果亚洲的邮箱，说明原因。然后等大概三天(工作日)信用卡被扣款($99)苹果会回复邮件说明注册成功。

之后就是配置你的账号，通过iTunes connect网页配置你收款账号(只要能收外汇的就行，国内银行卡大部分都支持)，税务信息等等，之后上传程序就可以等收款了。

具体流程会更复杂，你可以到cocoachina论坛上搜iDP,有很多详细图文教程。

根据我自己的经验，一个iPad或iPhone app的开发周期大概是这样的：

1. App的idea形成

2. App的主要功能设计
3. App的大概界面构思和设计（使用流程设计）
4. 大功能模块代码编写
5. 大概的界面模块编写
6. 把大概的界面和功能连接后，app的大致demo就出来了
7. demo自己试用和体验几遍后，根据情况修改
8. app的0.8左右版本完成后可以加入production的图标和部分UI图片
9. 没有大错误后，0.9版本可以尝试寻找beta用户
10. 根据测试用户的反馈，重复 7 - 9的步骤

使用**TableView**实现多级树型**menu**

分类： IOS开发 2013-02-03 18:52 3886人阅读 评论(0) 收藏 举报
iosiOSIOSmenuMenuMENUObjective-Cobjective-cObjective-cobjective-CtableViewTableViewtableview

官方UIKit下的TableView，支持section和row的显示，但不支持在talbeview里显示多级树型结构的menu，因为项目需要便写了一个支持多级目录显示menu的Demo（下载传送门）。支持菜单展开动画效果，支持级联打开下下级子目录。
效果图如下：

要现实多级目录，首先要做的是在内存构建树型结构，通过这个树型结构，当用户点击了某个有子项的菜单，其变会根据树型结构将menu展开或收起。

下面是“树”中节点的构造，在这里并没有使用CFTree，出于两点考虑：一是menu的结构一般只需知道其子菜单即可，是一个简单应用；第二是项目内部有对C++不了解的童鞋。

[cpp] view plaincopyprint?

```
1.  #import <Foundation/Foundation.h>
2.  @interface MyItem : NSObject
3.  @property (nonatomic,retain)  NSString * title;
4.  @property (nonatomic) NSInteger level;
5.  @property (nonatomic, retain) NSMutableArray *subItems;
6.  @property (nonatomic) BOOL isSubItemOpen;
7.  @property (nonatomic) BOOL isSubCascadeOpen;
8.  @end
```

其中，subItems这个数组，存放该节点的子菜单，使用isSubItemOpen来标记自菜单是否被打开，使用isCascadeOpen标记该子菜单是否要在其父菜单展开时自动展开。

菜单级联收起代码

[cpp] view plaincopyprint?

```
1.  #pragma mark -- insert
2.  - (NSArray *)insertMenuIndexPaths:(MyItem *)item
3.  {
4.      NSArray * arr;
5.      [treeItemsToInsert removeAllObjects];
```

```
    [self insertMenuObject:item];
    arr = [self insertIndexsOfMenuObject:treeItemsToInsert];
    return arr;
}
- (void) insertMenuObject:(MyItem *)item
{
    NSLog(@"%d",[_tableViewData indexOfObject:item]);
    if (item == nil)
    {
        return ;
    }

    NSIndexPath *path = [NSIndexPath indexPathForRow:
[_tableViewData indexOfObject:item] inSection:0];

    MyItem *childItem;
    for (int i = 0; i<[item.subItems count] ; i++) {
        childItem = [item.subItems objectAtIndex:i];
        [_tableViewData insertObject:childItem atIndex:path.row + i +1];
        [treeItemsToInsert addObject:childItem];
        item.isSubItemOpen = YES;
    }

    for (int i = 0; i <[item.subItems count]; i ++) {
        childItem = [item.subItems objectAtIndex:i];

        if (childItem .isSubCascadeOpen) {
            [self insertMenuObject:childItem];
        }

    }
    return ;

}
- (NSArray *) insertIndexsOfMenuObject :(NSMutableArray *) array
{

    NSMutableArray * mutableArr;
    mutableArr = [NSMutableArray array];
    for (MyItem * item in array) {
        NSIndexPath *path = [NSIndexPath indexPathForRow:
[_tableViewData indexOfObject:item] inSection:0];
        [mutableArr addObject:path];
    }
    return mutableArr;
}
```

其中使用insertMenuObject函数的递归调用，来完成对item子树的遍历

在tableViewController里逻辑要尽量简单，其中在tableView:didSelectRowAtIndexPath里代码如下

[cpp] view plaincopyprint?

```
<pre name="code" class="cpp">    MenuItemCell * cell;
    cell = (MenuItemCell *)[tableView cellForRowAtIndexPath:indexPath];
```

```
3.      if (cell.item.isSubItemOpen)
4.      {
5.          //remove
6.          NSArray * arr;
7.          arr = [_menuData deleteMenuIndexPaths:cell.item];
8.          if ([arr count] >0) {
9.
                [tableView deleteRowsAtIndexPaths: arr withRowAnimation:UITableViewRowAnimationBotto
    m];
10.         }
11.     }
12.     else
13.     {
14.         //insert
15.         NSArray * arr;
16.         arr = [_menuData insertMenuIndexPaths:cell.item];
17.         if ([arr count] >0) {
18.
                [tableView insertRowsAtIndexPaths:arr withRowAnimation:UITableViewRowAnimationBottom
    ];
19.         }
20.     }</pre><br>
21. <br>
22. <pre></pre>
23. <span style="font-size:14px">其中，cell的插入和移除的动画，使用<span style="font-
    family:Menlo"></span>withRowAnimation完成。</span>
24. <p></p>
25. <p><span style="font-family:Menlo"><span style="font-size:20px"><br>
26. </span></span><br>
27. <br>
28. <br>
29. </p>
30. <p><span style="font-size:18px"><br>
31. </span></p>
```

[很全面的**ios**面试题以及解答，很多答案不一定全对，网上摘录以及自己的理解]

分类： IOS面试 2013-04-26 21:29 233人阅读 评论(0) 收藏 举报

1、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.

2、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

3、关键字const什么含义?

const意味着”只读”，下面的声明都是什么意思?

const int a;

int const a;

const int *a;

int * const a;

int const * a const;

前两个的作用是一样，a是一个常整型数。第三个意味着a是一个指向常整型数的指针（也就是，整型数是不可修改的，但指针可以）。第四个意思a是一个指向整型数的常指针（也就是说，指针指向的整型数是可以修改的，但指针是不可修改的）。 最后一个意味着a是一个指向常整型数的常指针（也就是说，指针指向的整型数是不可修改的，同时

指针也是不可修改的）。

结论：

•; 关键字const的作用是为给读你代码的人传达非常有用的信息，实际上，声明一个参数为常量是为了告诉了用户这个参数的应用目的。如果

你曾花很多时间清理其它人留下的垃圾，你就会很快学会感谢这点多余的信息。（当然，懂得用const的程序员很少会留下的垃圾让别人来清

理的。）

•; 通过给优化器一些附加的信息，使用关键字const也许能产生更紧凑的代码。

•; 合理地使用关键字const可以使编译器很自然地保护那些不希望被改变的参数，防止其被无意的代码修改。简而言之，这样可以减少bug的出现。

欲阻止一个变量被改变，可以使用 const 关键字。在定义该 const 变量时，通常需要对它进行初

始化，因为以后就没有机会再去改变它了；

（2）对指针来说，可以指定指针本身为 const，也可以指定指针所指的数据为 const，或二者同时指

定为 const；

（3）在一个函数声明中，const 可以修饰形参，表明它是一个输入参数，在函数内部不能改变其值；

（4）对于类的成员函数，若指定其为 const 类型，则表明其是一个常函数，不能修改类的成员变量；

（5）对于类的成员函数，有时候必须指定其返回值为 const 类型，以使得其返回值不为“左值”。

4、关键字volatile有什么含义？并给出三个不同例子?

一个定义为volatile的变量是说这变量可能会被意想不到地改变，这样，编译器就不会去假设这个变量的值了。精确地说就是，优化器在用到

这个变量时必须每次都小心地重新读取这个变量的值，而不是使用保存在寄存器里的备份。下面是volatile变量的几个例子：

• 并行设备的硬件寄存器（如：状态寄存器）

• 一个中断服务子程序中会访问到的非自动变量(Non-automatic variables)

• 多线程应用中被几个任务共享的变量

• 一个参数既可以是const还可以是volatile吗？解释为什么。

• 一个指针可以是volatile 吗？解释为什么。

下面是答案：

• 是的。一个例子是只读的状态寄存器。它是volatile因为它可能被意想不到地改变。它是const因为程序不应该试图去修改它。

• 是的。尽管这并不很常见。一个例子是当一个中服务子程序修该一个指向一个buffer的指针时。

static作用?

函数体内 static 变量的作用范围为该函数体，不同于 auto 变量，该变量的内存只被分配一次，

因此其值在下次调用时仍维持上次的值；

（2）在模块内的 static 全局变量可以被模块内所用函数访问，但不能被模块外其它函数访问；

（3）在模块内的 static 函数只可被这一模块内的其它函数调用，这个函数的使用范围被限制在声明

它的模块内；

（4）在类中的 static 成员变量属于整个类所拥有，对类的所有对象只有一份拷贝；

（5）在类中的 static 成员函数属于整个类所拥有，这个函数不接收 this 指针，因而只能访问类的static 成员变量。

6、#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

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.

10、为什么很多内置的类，如TableViewController的delegate的属性是assign不是retain?

循环引用

所有的引用计数系统，都存在循环应用的问题。例如下面的引用关系：

- 对象a创建并引用到了对象b.
- 对象b创建并引用到了对象c.
- 对象c创建并引用到了对象b.

这时候b和c的引用计数分别是2和1。当a不再使用b，调用release释放对b的所有权，因为c还引用了b，所以b的引用计数为1，b不会被释放。b不释放，c的引用计数就是1，c也不会被释放。从此，b和c永远留在内存中。

这种情况，必须打断循环引用，通过其他规则来维护引用关系。比如，我们常见的delegate往往是assign方式的属性而不是retain方式的属性，赋值不会增加引用计数，就是为了防止delegation两端产生不必要的 循环引用。如果一个UITableViewController 对象a通过retain获取了UITableView对象b的所有权，这个 UITableView对象b的delegate又是a，如果这个delegate是retain方式的，那基本上就没有机会释放这两个对象了。自己在设 计使用delegate模式时，也要注意这点。

11、定义属性时，什么情况使用copy、assign、retain?

assign用于简单数据类型，如NSInteger,double,bool,

retain和copy用于对象，

copy用于当a指向一个对象，b也想指向同样的对象的时候，如果用assign，a如果释放，再调用b会crash,如果用copy 的方式，a和b各自有自己的内存，就可以解决这个问

题。

retain 会使计数器加一，也可以解决assign的问题。

另外：atomic和nonatomic用来决定编译器生成的getter和setter是否为原子操作。在多线程环境下，原子操作是必要的，否则有可能引起错误的结果。

加了atomic，setter函数会变成下面这样：

if (property != newValue) {

[property release];

property = [newValue retain];

}

12、对象是什么时候被release的?

引用计数为0时。

autorelease实际上只是把对release的调用延迟了，对于每一个 Autorelease，系统只是把该Object放入了当前的Autorelease pool中，当该pool被释放时，该pool中的所有Object会被调用Release。对于每一个Runloop，系统会隐式创建一个Autorelease pool，这样所有的 release pool会构成一个象CallStack一样的一个栈式结构，在每一个Runloop结束时，当前栈顶的 Autorelease pool会被销毁，这样这个pool里的每个Object（就是autorelease的对象）会被release。那什么是一 个Runloop呢？一个UI事件，Timer call， delegate call， 都会是一个新的Runloop

13、iOS有没有垃圾回收?

Objective-C 2.0也是有垃圾回收机制的，但是只能在Mac OS X Leopard 10.5 以上的版本使用。

Objective-C 2.0有Garbage collection，但是iOS平台不提供。

一般我们了解的objective-c对于内存管理都是手动操作的，但是也有自动释放池。

不要和arc机制搞混就好了。arc并不是gc，只是编译器特性

iOS 5 最显著的变化就是增加了 Automatic Reference Counting(自动引用计数)。ARC 是新 LLVM 3.0 编译器的特性, 完全消除了手动内存管理的烦琐。在你的项目中使用 ARC 是非常简单的, 所有的编程都和以前一样, 除了你不再调用 retain, release, autorelease。启用 ARC之后, 编译器会自动在适当的地方插入适当的 retain, release,autorelease 语句。你不再需要担心内存管理, 因为编译器为你处理了一切。注意 ARC 是编译器特性, 而不是 iOS 运行时特性(除了 weak 指针系统), 它也不是其它语言中的垃圾收集器。因此 ARC 和手动内存管理性能是一样的, 有些时候还能更加快速, 因为编译器还可以执行某些优化。

指针保持对象的生命

ARC 的规则非常简单:只要还有一个变量指向对象, 对象就会保持在内存中。当指针指向新值, 或者指针不再存在时, 相关联的对象就会自动释放。这条规则对于实例变量、synthesize 属性、本地变量都是适用的。

Java的垃圾回收机制是Java虚拟机提供的能力，用于在空闲时间以不定时的方式动态回收无任何引用的对象占据的内存空间。

需要注意的是：垃圾回收回收的是无任何引用的对象占据的内存空间而不是对象本身，很多人来我公司面试时，我都会问这个问题的，70%以上的人回答的含义是回收对象，实际上这是不正确的。

System.gc()

Runtime.getRuntime().gc()

上面的方法调用时用于显式通知JVM可以进行一次垃圾回收，但真正垃圾回收机制具体在什么时间点开始发生动作这同样是不可预料的，这和抢占式的线程在发生作用时的原理一样。

31.自动释放池跟GC有什么区别？iPhone上有GC么？［pool release］ 和［pool drain］有什么区别

”Autorelease Pools”(自动释放池)在应用中的使用技巧。

1，Autorelease Pools概要

一 个”Autorelease Pool”实例中“包含”其它各种调用了”autorelease”方法的对象。当它释放时，其中所有被管理对象都会收 到”relrease”的消信。注意，同一个对象可以被多次调用”autorelease”方法，并可以放到同一个”Autorelease Pool” 中。引入这个自动释放池机制，对象的”autorelease”方法代替”relrease”方法可以延长它的生命周期，直接到当前”Autorelrease Pool”释放。如果想让此对象的生命周期超过”Autorelease Pool”，还可以再次”retain”，呵 呵，有意思吧?且让我慢慢道来。

Cocoa总是认为当前至少有一个”Autorelease Pool”对象是可用的。若此对象并不存在，你调用的”autorelease”的所有对象都不会被自动释放掉，可想而知，造成内存泄露。Cocoa把这个错误信息写入日志??仅仅是为了以后分析。

你 可以用”alloc”与”init”方法创建一个”NSAutoreleasePool”对象，并且可以调用”release”或”drain” （”release”与”drain”的区别是”drain”在有GC的环境中会引起GC回收操作，”release”反之。但在非GC环境中，两者相 同。官方的说法是为了程序的兼容性，应该考虑用”drain”代替”release”，）方法来回收它（调用它的”autorelease”或”retain”方法会引起异常）。在一个完整的上下文最后”Autorelease Pool”对象应该被”release”掉（在方法内或一段循环 体内创建的”Autorelease Pool”对象）。

“Autorelease Pools” 的所有实例在栈中管理（我们暂时叫他“自动释放池栈”），并且它们是可以被嵌套的（父生子，子生孙。。。子子孙孙 ^_^）。例如，当我们创建一个”Autorelease Pool”对象后，它就被自动放到“自动释放池栈”的栈顶。当本池对象回收时，它就随之从这个栈中POP掉。那么也就是说， 当任何一个对象调用”autorelease”方法后，它会被放入当前线程中当前栈顶的自动释放池中。

接 下来我们聊聊”Autorelease Pools”的嵌套问题。在你的应用中，你可以任意多的创建”Autorelease Pool”对象，而这些对 象被当前线程的“自动释放池栈”所管理。那么除了一个接一个的顺序创建并销毁它的情况外，还有一种使用方式，就是嵌套式的创建与使用。例如：在你的主函数 创建了一个”autorelease pool”，然后又调用了创建

了”autorelease pool”实例的其它方法；或是在外循环中创建 了”Autorelease Pool”的实例，而内循环中也做了相同的事情。有意思吧，呵呵，嵌套的机制使父Pool实例释放后，它的所有子Pool也 将释放。但这里还存在一些副作用，后续文章会详细讨论。

“Application kit”在一个事件循环里会自动创建一个”autorelease pool”。像鼠标键的按下与释放，所以你编写的代码通常不需要考虑太多这方面的事情。当然，有以下三种情况你会创建与销毁自己的Pool实例：

1，应用不是基于”Application Kit”，像”Command-line tool”，因为它并没有内置的”autorelease pools”的支持。
2，创建线程，你必需在线程开始时创建一个”Autorelease Pool”实例。反之，会造成内存池露（会在以后的文章详细说明线程与池的技巧）。
3，一个循环内创建了太多的临时对象，你应该为他们创建一个”Autorelease Pool”对象，并在下次循还前销毁它们。

2，自动释放池中的”Non-AppKit”应用

在”Non- AppKit”应用中使用自动释放池的机制其实是相当简单的事情。你仅仅需要在main()起始处创建”Autorelease Pool”对象，并在结 尾处释放掉它。就像在Xcode的Foundation Tool的创建模版里写的一样。这个确保你在应用生命周期内至少有一个”Autorelease Pool”是可用的。但是，这也使所有在此期间的所有”autorelease”的对象都必需在应用结束后才被释放。这也许 会引起在应用的使用中不断的增长，所以，你仍然考虑在不同的作用域创建新的”Autorelease Pool”。

大多应用中都存在各种级别的循环机制。在这些应用中，你可以在每个循环内的开头创建一个”Autorelease Pool”对象，并在结尾处释放掉它。

例如：

```
void main()
{
```

```
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];

    NSArray *args = [[NSProcessInfo processInfo] arguments];
    unsigned count, limit = [args count];

    for (count = 0; count < limit; count++)
    {
        NSAutoreleasePool *loopPool = [[NSAutoreleasePool alloc] init];
        NSString *fileContents;
        NSString *fileName;

        fileName = [args objectAtIndex:count];
        fileContents = [[[NSString alloc] initWithContentsOfFile:fileName] autorelease];
        // this is equivalent to using stringWithContentsOfFile:



        [loopPool release];
    }


    [pool drain];

    exit (EXIT_SUCCESS);
}
```

在 命令行中处理所有以参数传来的文件。一次循环处理一个文件。在循环的开头创建一个”NSAutoreleasePool”对象，并在循环结束时释放掉。因 此，任何在其中创建并调用“autorelease”的对象都将添加到这个Pool实例中，当本池被释放后，这些对象也将被回收。注意，任何在作用域内创 建的”autoreleased”对象（像”fileName”），虽然并没有显示的调用”autorelease”方法，但都将被当前池所管理并释放。

14、tableView的重用机制?

查看UITableView头文件，会找到NSMutableArray* visiableCells，和NSMutableDictnery* reusableTableCells两个结构。visiableCells内保存当前显示的cells，reusableTableCells保存可重用的cells。

TableView显示之初，reusableTableCells为空，那 么tableView dequeueReusableCellWithIdentifier:CellIdentifier返回nil。开始的cell 都是通过[[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault reuseIdentifier:CellIdentifier] 来创建，而且cellForRowAtIndexPath只是调用最大显示cell数的次数。

比如：有100条数据，iPhone一屏最多显示10个cell。程序最开始显示TableView的情况是：

1. 用[[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault reuseIdentifier:CellIdentifier] 创建10次cell，并给cell指定同样的重用标识(当然，可以为不同显示类型的cell指定不同的标识)。并且10个cell全部都加入到 visiableCells数组，reusableTableCells为空。

2. 向下拖动tableView，当cell1完全移出屏幕，并且 cell11(它也是alloc出来的，原因同上)完全显示出来的时候。cell11加入到visiableCells，cell1移出 visiableCells，cell1加入到reusableTableCells。

3. 接着向下拖动tableView，因为 reusableTableCells中已经有值，所以，当需要显示新的cell，cellForRowAtIndexPath再次被调用的时 候，tableView dequeueReusableCellWithIdentifier:CellIdentifier，返回cell1。 cell1加入到visiableCells，cell1移出reusableTableCells；cell2移出 visiableCells，cell2加入到reusableTableCells。之后再需要显示的Cell就可以正常重用了。

15、ViewController 的loadView、viewDidLoad、viewDidUnload分别是什么时候调用的，在自定义ViewCointroller时在这几个函数中应该做什么工作?

由init、loadView、viewDidLoad、viewDidUnload、dealloc的关系说起

init方法

在init方法中实例化必要的对象（遵从LazyLoad思想）

init方法中初始化ViewController本身

loadView方法

当view需要被展示而它却是nil时，viewController会调用该方法。不要直接调用该方法。

如果手工维护views，必须重载重写该方法

如果使用IB维护views，必须不能重载重写该方法

loadView和IB构建view

你在控制器中实现了loadView方法，那么你可能会在应用运行的某个时候被内 存管理控制调用。 如果设备内存不足的时候， view 控制器会收到didReceiveMemoryWarning的消息。 默认的实现是检查当前控 制器的view是否在使用。如果它的view不在当前正在使用的view hierarchy里面，且你的控制器实现了loadView方法，那么这个 view将被release, loadView方法将被再次调用来创建一个新的view。

viewDidLoad方法

viewDidLoad 此方法只有当view从nib文件初始化的时候才被调用。

重载重写该方法以进一步定制view

在iPhone OS 3.0及之后的版本中，还应该重载重写viewDidUnload来释放对view的任何索引

viewDidLoad后调用数据Model

viewDidUnload方法

当系统内存吃紧的时候会调用该方法（注：viewController没有被dealloc）

内存吃紧时，在iPhone OS 3.0之前didReceiveMemoryWarning是释放无用内存的唯一方式，但是OS 3.0及以后viewDidUnload方法是更好的方式

在该方法中将所有IBOutlet（无论是property还是实例变量）置为nil（系统release view时已经将其release掉了）

在该方法中释放其他与view有关的对象、其他在运行时创建（但非系统必须）的对 象、在viewDidLoad中被创建的对象、缓存数据等 release对象后，将对象置为nil（IBOutlet只需要将其置为nil，系统 release view时已经将其release掉了）

一般认为viewDidUnload是viewDidLoad的镜像，因为当view被重新请求时，viewDidLoad还会重新被执行

viewDidUnload中被release的对象必须是很容易被重新创建的对象（比如在viewDidLoad或其他方法中创建的对象），不要release用户数据或其他很难被重新创建的对象

dealloc方法

viewDidUnload和dealloc方法没有关联，dealloc还是继续做它该做的事情

16、ViewController的didReceiveMemoryWarning是在什么时候调用的？默认的操作是什么？

当程序接到内存警告时View Controller将会收到这个消息：didReceiveMemoryWarning

从iOS3.0开始，不需要重载这个函数，把释放内存的代码放到viewDidUnload中去。

这个函数的默认实现是:检查controller是否可以安全地释放它的view(这里加粗的view指的是controller的view属性)，比如view本身没有superview并且可以被很容易地重建（从nib或者loadView函数）。

如果view可以被释放，那么这个函数释放view并调用viewDidUnload。

你可以重载这个函数来释放controller中使用的其他内存。但要记得调用这个函数的

super实现来允许父类（一般是UIVIewController）释放view。

如果你的ViewController保存着view的子view的引用，那么，在早期的iOS版本中，你应该在这个函数中来释放这些引用。而在iOS3.0或更高版本中，你应该在viewDidUnload中释放这些引用。

17、列举Cocoa中常见的集中多线程的实现，并谈谈多线程安全的几种解决办法，一般什么地方会用到多线程?

NSThread,GCD等。尽量用上层分装好的方法去实现多线程而不是手动调用NSThread。

18、怎么理解MVC，在Cocoa中MVC是怎么实现的?

Model: 代表你的应用程序是什么（不是怎么展现）

Controller: 控制你的Model怎么展现给用户（UI逻辑）
View: Controller的奴隶。。。

1 Model，Controller，View相互通讯的规则:
Controller可以直接和Model通信

1 Controller也可以直接和View通信

1 Model和View永远不能直接通信

1 iOS中View和Controller的通信是透明和固定的，主要通过outlet和action实现

1 View使用Delegate接口和Controller同步信息

1 View不直接和数据通信，使用dataSource接口从Controller处获取数据

1 View的delegate和dataSource一般就是Controller

1 Controller负责为View翻译和格式化Model的数据

1 Model使用Notification & KVO的方式分发数据更新信息，Controller可以有选择的监听自己感兴趣的信息。

1 View也可以监听广播信息，但一般不是Model发出的信息

1 一个完整的App就是很多MVC的集合

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

Delegate:
一对一

Notification:
一对多

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。

20、self.跟self什么区别?

21、id、nil代表什么?

id和void *并非完全一样。在上面的代码中，id是指向struct objc_object的一个指针，这个意思基本上是说，id是一个指向任何一个继承了Object（或者NSObject）类的对象。需要注意的是id是一个指针，所以你在使用id的时候不需要加星号。比如
id foo=nil定义了一个nil指针，这个指针指向NSObject的一个任意子类。而
id *foo=nil则定义了一个指针，这个指针指向另一个指针，被指向的这个指针指向NSObject的一个子类。

nil和C语言的NULL相同，在objc/objc.h中定义。nil表示一个Objctive-C对象，这个对象的指针指向空（没有东西就是空）。

首字母大写的Nil和nil有一点不一样，Nil定义一个指向空的类（是Class，而不是对象）。

SEL是“selector”的一个类型，表示一个方法的名字

Method（我们常说的方法）表示一种类型，这种类型与selector和实现(implementation)相关

IMP定义为 id (*IMP) (id, SEL, …)。这样说来， IMP是一个指向函数的指针，这个被指向的函数包括id(“self”指针)，调用的SEL（方法名），再加上一些其他参数.说白了IMP就是实现方法。

22、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)

23、类别的作用?

有时我们需要在一个已经定义好的类中增加一些方法，而不想去重写该类。比如，当工程已经很大，代码量比较多，或者类中已经包住很多方法，已经有其他代码调用了该类创建对象并使用该类的方法时，可以使用类别对该类扩充新的方法。

注意：类别只能扩充方法，而不能扩充成员变量。

24、委托（举例）

委托代理（degegate），顾名思义，把某个对象要做的事情委托给别的对象去做。那么别的对象就是这个对象的代理，代替它来打理要做的事。反映到程序中，首先要明确一个对象的委托方是哪个对象，委托所做的内容是什么。

委托机制是一种设计模式，在很多语言中都用到的，这只是个通用的思想，网上会有很多关于这方面的介绍。

那么在苹果开发过程中，用到委托的程序实现思想如下，我主要拿如何在视图之间传输信息做个例子。

譬如：在两个页面（UIIview视图对象）实现传值，用委托（delegate）可以很好做到！

方法：

类A

```
@interface A：UIView

    id transparendValueDelegate;

   @property(nomatic, retain) id transparendValueDelegate;

@end


@implemtion A

@synthesize transparendValueDelegate

-(void)Function

{

    NSString* value = @"hello";

    //让代理对象执行transparendValue动作

    [transparendValueDelegate transparendValue: value];

}

@end
```

类B

```
@interface B：UIView

    NSString* value;
```

```
@end

@implemtion B

-(void)transparendValue:(NSString*)fromValue

{

    value = fromValue;

    NSLog(@"the value is %@ ",value);

}

@end


//下面的设置A代理委托对象为B

//在定义A和B类对象处：

A* a = [[A alloc] init];

B* b = [[B alloc] init];

a. transparendValueDelegate = b;//设置对象a代理为对象b
```

这样在视图A和B之间可以通过委托来传值！

25、retainCount？

26..属性readwrite，readonly，assign，retain，copy，nonatomic 各是什么作用，在那种情况下用

assign：指定setter方法用简单的赋值，这是默认操作。你可以对标量类型（如int）使用

这个属性。你可以想象一个float，它不是一个对象，所以它不能retain、copy。

retain：指定retain应该在后面的对象上调用，前一个值发送一条release消息。你可以想象一个NSString实例，它是一个对象，而且你可能想要retain它。

copy：指定应该使用对象的副本（深度复制），前一个值发送一条release消息。基本上像retain，但是没有增加引用计数，是分配一块新的内存来放置它。

readonly：将只生成getter方法而不生成setter方法（getter方法没有get前缀）。

readwrite：默认属性，将生成不带额外参数的getter和setter方法（setter方法只有一个参数）。

atomic：对于对象的默认属性，就是setter/getter生成的方法是一个原子操作。如果有多个线程同时调用setter的话，不会出现某一个线程执行setter全部语句之前，另一个线程开始执行setter的情况，相关于方法头尾加了锁一样。

nonatomic：不保证setter/getter的原子性，多线程情况下数据可能会有问题。

27.类变量的@protected ,@private,@public,@package声明各有什么含义

Objective-C 对存取权限的设定。也是变量的作用域。

protected 该类和所有的子类中的方法可以直接访问这样的变量，这是默认的。

private — 该类中的方法可以访问这样的变量，子类不可以。 public — 除了自己和子类中的方法外，也可以被其他类或者其他模块中的方法所访问。开放性最大。 package — 对于64位图像，这样的成员变量可以在实现这个类的图像中随意访问。

28.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

29.Cocoa中与虚基类的概念么？怎么简洁的实现

30.NSString 和 NSMutableString 有什么区别

NSString相当于一个const char* 不可以改变。

而 NSMutableString相当于 char* 可以改变内部的内容。

32.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

33.响应者链是什么

响应者链是Application Kit事件处理架构的中心机制。它由一系列链接在一起的响应者对象组成，事件或者动作消息可以沿着这些对象进行传递。如图6-20显示的那样，如果一个响应者对象不能处理某个事件或动作－也就是说，它不响应那个消息，或者不认识那个事件，则将该消息重新发送给链中的下一个响应者。消息沿着响应者链向上、向更高级别的对象传递，直到最终被处理（如果最终还是没有被处理，就会被抛弃）。

当Application Kit在应用程序中构造对象时，会为每个窗口建立响应者链。响应者链中的基本对象是NSWindow对象及其视图层次。在视图层次中级别较低的视图将比级别更高的视图优先获得处理事件或动作消息的机会。NSWindow中保有一个第一响应者的引用，它通常是当前窗口中处于选择状态的视图，窗口通常把响应消息的机会首先给它。对于事件消息，响应者链通常以发生事件的窗口对应的NSWindow对象作为结束，虽然其它对象也可以作为下一个响应者被加入到NSWindow对象的后面。

34..UIscrollVew用到了什么设计模式？还能再foundation库中找到类似的吗?
组合模式composition，所有的container view都用了这个模式
观察者模式observer，所有的UIResponder都用了这个模式。

模板(Template)模式，所有datasource和delegate接口都是模板模式的典型应用

33. .timer的间隔周期准吗？为什么？怎样实现一个精准的timer?
NSTimer可以精确到50-100毫秒.
NSTimer不是绝对准确的,而且中间耗时或阻塞错过下一个点,那么下一个点就pass过去了

此份面试题包含40个题目，是现在网上能搜索到的一个比较热的一份，但是答案并不是很详细和完整，基本答案来着cocoaChina，和一些自己的补充。
1.Difference between shallow copy and deep copy?
浅复制和深复制的区别?
答案：浅层复制：只复制指向对象的指针，而不复制引用对象本身。
深层复制：复制引用对象本身。

意思就是说我有个A对象，复制一份后得到A_copy对象后，对于浅复制来说，A和A_copy指向的是同一个内存资源，复制的只不过是是一个指针，对象本身资源还是只有一份，那如果我们对A_copy执行了修改操作,那么发现A引用的对象同样被修改，这其实违背了我们复制拷贝的一个思想。深复制就好理解了,内存中存在了两份独立对象本身。

用网上一哥们通俗的话将就是：

浅复制好比你和你的影子，你完蛋，你的影子也完蛋

深复制好比你和你的克隆人，你完蛋，你的克隆人还活着。

2.What is advantage of categories? What is difference between implementing a category and inheritance?

类别的作用？继承和类别在实现中有何区别?

答案：category 可以在不获悉，不改变原来代码的情况下往里面添加新的方法，只能添加，不能删除修改。

并且如果类别和原来类中的方法产生名称冲突，则类别将覆盖原来的方法，因为类别具有更高的优先级。

类别主要有3个作用：

(1)将类的实现分散到多个不同文件或多个不同框架中。

(2)创建对私有方法的前向引用。

(3)向对象添加非正式协议。

继承可以增加，修改或者删除方法，并且可以增加属性。

3.Difference between categories and extensions?

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。

4.Difference between protocol in objective c and interfaces in java?

oc中的协议和java中的接口概念有何不同?

答案：OC中的代理有2层含义，官方定义为 formal和informal protocol。前者和Java接口一样。

informal protocol中的方法属于设计模式考虑范畴，不是必须实现的，但是如果有实现，就会改变类的属性。

其实关于正式协议，类别和非正式协议我很早前学习的时候大致看过，也写在了学习教程里

“非正式协议概念其实就是类别的另一种表达方式“这里有一些你可能希望实现的方法，你可以使用他们更好的完成工作”。

这个意思是，这些是可选的。比如我门要一个更好的方法，我们就会申明一个这样的类别去实现。然后你在后期可以直接使用这些更好的方法。

这么看，总觉得类别这玩意儿有点像协议的可选协议。"

现在来看，其实protocal已经开始对两者都统一和规范起来操作，因为资料中说“非正式协议使用interface修饰“，

现在我们看到协议中两个修饰词：“必须实现(@requied)”和“可选实现(@optional)”。

5.What are KVO and KVC?

答案：kvc:键 - 值编码是一种间接访问对象的属性使用字符串来标识属性，而不是通过调用存取方法，直接或通过实例变量访问的机制。

很多情况下可以简化程序代码。apple文档其实给了一个很好的例子。

kvo:键值观察机制，他提供了观察某一属性变化的方法，极大的简化了代码。

具体用看到嗯哼用到过的一个地方是对于按钮点击变化状态的的监控。

比如我自定义的一个button

[cpp]

```
[self addObserver:self forKeyPath:@"highlighted" options:0 context:nil];


#pragma mark KVO

- (void)observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object change:
(NSDictionary *)change context:(void *)context
{
    if ([keyPath isEqualToString:@"highlighted"] ) {
        [self setNeedsDisplay];
    }
}
```

对于系统是根据keypath去取的到相应的值发生改变，理论上来说是和kvc机制的道理是

一样的。
对于kvc机制如何通过key寻找到value：
“当通过KVC调用对象时，比如：[self valueForKey:@”someKey”]时，程序会自动试图通过几种不同的方式解析这个调用。首先 查找对象是否带有 someKey 这个方法，如果没找到，会继续查找对象是否带有someKey这个实例变量（iVar），如果还没有找到，程序会继续 试图调用 -(id) valueForUndefinedKey:这个方法。如果这个方法还是没有被实现的话，程序会抛出一个 NSUndefinedKeyException异常错误。

(cocoachina.com注：Key-Value Coding查找方法的时候，不仅仅会查找someKey这个方法，还会查找 getsomeKey这个方法，前面加一个get，或者_someKey以及_getsomeKey这几种形式。同时，查找实例变量的时候也会不仅仅查找 someKey这个变量，也会查找_someKey这个变量是否存在。）

设计valueForUndefinedKey:方法的主要目的是当你使用-(id)valueForKey方法从对象中请求值时，对象能够在错误发生前，有最后的机会响应这个请求。这样做有很多好处，下面的两个例子说明了这样做的好处。“
来至cocoa，这个说法应该挺有道理。
因为我们知道button却是存在一个highlighted实例变量.因此为何上面我们只是add一个相关的keypath就行了，
可以按照kvc查找的逻辑理解，就说的过去了。

6.What is purpose of delegates?
代理的作用?
答案：代理的目的是改变或传递控制链。允许一个类在某些特定时刻通知到其他类，而不需要获取到那些类的指针。可以减少框架复杂度。
另外一点，代理可以理解为java中的回调监听机制的一种类似。

7.What are mutable and immutable types in Objective C?
oc中可修改和不可以修改类型。
答案：可修改不可修改的集合类。这个我个人简单理解就是可动态添加修改和不可动态添加修改一样。
比如NSArray和NSMutableArray。前者在初始化后的内存控件就是固定不可变的，后者可以添加等，可以动态申请新的内存空间。

8.When we call objective c is runtime language what does it mean?

我们说的oc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? ~~~

9.what is difference between NSNotification and protocol?

通知和协议的不同之处?

答案：协议有控制链(has-a)的关系，通知没有。

首先我一开始也不太明白，什么叫控制链（专业术语了~）。但是简单分析下通知和代理的行为模式，我们大致可以有自己的理解

简单来说，通知的话，它可以一对多，一条消息可以发送给多个消息接受者。

代理按我们的理解，到不是直接说不能一对多，比如我们知道的明星经济代理人，很多时候一个经济人负责好几个明星的事务。

只是对于不同明星间，代理的事物对象都是不一样的，一一对应，不可能说明天要处理A明星要一个发布会，代理人发出处理发布会的消息后，别称B的

发布会了。但是通知就不一样，他只关心发出通知，而不关心多少接收到感兴趣要处理。

因此控制链（has-a从英语单词大致可以看出，单一拥有和可控制的对应关系。

10.What is push notification?

什么是推送消息?

答案：太简单，不作答~~~~~~~~~~

这是cocoa上的答案。

其实到不是说太简单，只是太泛泛的一个概念的东西。就好比说，什么是人。

推送通知更是一种技术。

简单点就是客户端获取资源的一种手段。
普通情况下，都是客户端主动的pull。
推送则是服务器端主动push。

11.Polymorphism?
关于多态性
答案：多态，子类指针可以赋值给父类。
这个题目其实可以出到一切面向对象语言中，
因此关于多态，继承和封装基本最好都有个自我意识的理解，也并非一定要把书上资料上写的能背出来。
最重要的是转化成自我理解。

12.Singleton?
对于单例的理解
答案：11，12题目其实出的有点泛泛的感觉了，可能说是编程语言需要或是必备的基础。
基本能用熟悉的语言写出一个单例，以及可以运用到的场景或是你编程中碰到过运用的此种模式的框架类等。
进一步点，考虑下如何在多线程访问单例时的安全性。

13.What is responder chain?
说说响应链
答案： 事件响应链。包括点击事件，画面刷新事件等。在视图栈内从上至下，或者从下之上传播。
可以说点事件的分发，传递以及处理。具体可以去看下touch事件这块。因为问的太抽象化了
严重怀疑题目出到越后面就越笼统。

14.Difference between frame and bounds?
frame和bounds有什么不同?
答案:frame指的是：该view在父view坐标系统中的位置和大小。（参照点是父亲的坐标系统）
bounds指的是：该view在本身坐标系统中 的位置和大小。（参照点是本身坐标系统）

15.Difference between method and selector?

方法和选择器有何不同?
答案：selector是一个方法的名字，method是一个组合体，包含了名字和实现.
详情可以看apple文档。

16.Is there any garbage collection mechanism in Objective C.?
OC的垃圾回收机制?
答案： OC2.0有Garbage collection，但是iOS平台不提供。
一般我们了解的objective-c对于内存管理都是手动操作的，但是也有自动释放池。
但是差了大部分资料，貌似不要和arc机制搞混就好了。
求更多~~

17.NSOperation queue?
答案：存放NSOperation的集合类。
操作和操作队列，基本可以看成java中的线程和线程池的概念。用于处理ios多线程开发的问题。
网上部分资料提到一点是，虽然是queue，但是却并不是带有队列的概念，放入的操作并非是按照严格的先进现出。
这边又有个疑点是，对于队列来说，先进先出的概念是Afunc添加进队列，Bfunc紧跟着也进入队列，Afunc先执行这个是必然的，
但是Bfunc是等Afunc完全操作完以后，B才开始启动并且执行，因此队列的概念离乱上有点违背了多线程处理这个概念。
但是转念一想其实可以参考银行的取票和叫号系统。
因此对于A比B先排队取票但是B率先执行完操作，我们亦然可以感性认为这还是一个队列。
但是后来看到一票关于这操作队列话题的文章，其中有一句提到
“因为两个操作提交的时间间隔很近，线程池中的线程，谁先启动是不定的。”
瞬间觉得这个queue名字有点忽悠人了，还不如pool~
综合一点，我们知道他可以比较大的用处在于可以帮组多线程编程就好了。

18.What is lazy loading?
答案：懒汉模式，只在用到的时候才去初始化。
也可以理解成延时加载。
我觉得最好也最简单的一个列子就是tableView中图片的加载显示了。

一个延时载，避免内存过高，一个异步加载，避免线程堵塞。

19.Can we use two tableview controllers on one viewcontroller?
是否在一个视图控制器中嵌入两个tableview控制器?
答案：一个视图控制只提供了一个View视图，理论上一个tableViewController也不能放吧，
只能说可以嵌入一个tableview视图。当然，题目本身也有歧义，如果不是我们定性思维认为的UIViewController，
而是宏观的表示视图控制者，那我们倒是可以把其看成一个视图控制者，它可以控制多个视图控制器，比如TabbarController
那样的感觉。

20.Can we use one tableview with two different datasources? How you will achieve this ?
一个tableView是否可以关联两个不同的数据源？你会怎么处理?
答案：首先我们从代码来看，数据源如何关联上的，其实是在数据源关联的代理方法里实现的。
因此我们并不关心如何去关联他，他怎么关联上，方法只是让我返回根据自己的需要去设置如相关的数据源。
因此，我觉得可以设置多个数据源啊，但是有个问题是，你这是想干嘛呢？想让列表如何显示，不同的数据源分区块显示?

@implementation

实现类

@protocol

声明协议

@optional

与@protocol配合使用，说明协议中的某个或者某几个方法可以不实现

@required

与@protocol配合使用，说明协议中的某个方法或者某几个方法必须实现

@end

与@interface ,@implementation,@protocol配合使用，代表声明或者实现结束

@encode

@encode为编译器宏，它可以将类型转换为相应的字符串。

id

id是指向Objective-C类对象的指针，它可以声明为任何类对象的指针，当在Objective-C中使用id时，编译器会假定你知道，id指向哪个类的对象。与void*是不同的是，void*编译器不知道也不假定指向任何类型的指针。

nil

定义为一个常量，如果一个指针的值为nil,代表这个指针没有指向任何对象。

self

在Objective-C中，关键字self与c++中this是同一概念，就是类对象自身的地址，通过self可以调用自己的实例变量和方法

Super

当子类需要调用父类的方法时，会用到Super关键字. Super指向的是父类的指针，子类重写父类的方法时，调用父类的方法是一个比较好的习惯。因为当我们不知道父类在该

方法中实现的功能时，如果不调用父类的方法，有可能我们重写的方法会失去该功能，这是我们不愿意看到的情况。

NSNull

NSNull是没有的意思，如果一个字典的值为NSNull,那说明与该值对应的Key是没有值的，例如Key为address，说明与address对应的是值是没有。

self super class public protected private id

[self class] [super class] selector

objective-c runtime reference

标准用法

self = [super init]

new

1 Objective-C有一个特性，就是可以把类当成对象来发送消息，这种用法通常用于新建对像时，例如 XXX *object = [XXX new];

类方法 +

如果想声明属于类而不属于类对象的方法，用+。＋用来修饰类的方法，使用+修饰的类方法，是整个类的方法，不属于哪一个类对象，这与C++中的static在类中使用的概念一样，

%@

在NSLog中，使用%@表示要调用对象的description方法。

概念

类

是一种结构，它表示对象的类型，就像int与 char 一样，也可以声明类的变量(对像)

实例化

为类的对象分配内存和初始化，达到可以使用该 类对象的目的。

对象(实例)

类的实例化后的产物

消息

在Object-C中，类的对象执行的操作，是通过给该类或者该类对象发送消息实现，如：［object func］；就是给object对象发送func消息，类似C++中的方法调用。给object对象发送func消息后，object对象查询所属类的func方法执行。

方法调度

当向一个对象发送消息时(调用方法)，这个方法是怎么被调用的呢？这就依赖于方法高度程序，方法调度程序查找的方法如下：

在本类的方法中，找被调用的方法，如果找到了，就调用，如果找不到被沿着继承路径去查找，从哪个类找到，就调用哪个类的方法，如果到最根上的类还是没有找到，那编译就会出错。

继承与复合

在Objective-C中支持继承，但只是支持单一继承(有且只有一个父类有)，如果想使用多继承的特性，可以使用分类和协议技术。

继承是is-a,复合是has-a。复合是通过包含指向对象的指针实现的，严格意义上讲，复合是针对于对象间来说，对于基本数据类型来说，它们被认为是对象的一部分。

装箱与拆箱

由于NSArray,NSDirectory等类不能直接存储基本数据类型，所以要想在NSArray＼NSDirectory中使用基本数据类型，就得使用装箱与拆箱。

在Objective-C中，可以使用NSNumber和NSValue来实现对数据类型的包装，NSNumber可以实现对基本数据类型的包装，NSValue可以实现对任意类型数据的包装。

将基本类型封装成对象叫装箱，从封装的对象中提取基本类型叫拆箱(取消装箱)，其它语言如Java原生支持装箱与拆箱，Ojbective-C不支持自动装箱与拆箱，如果需要得需要自己来实现装箱与拆箱。

存取方法

在使用类对象的实例变量(成员数据)时，不要直接使用对象中的实例，要使用存以方法来获取或者修改实例，既setter和getter,在Cocoa中，存取方法有命名习惯，我们得符合这种习惯，以便于与其它团队成员合作。setter方法是修改或者设置实例值，命名习惯为set+实例名，例有一个类有path实例变量，那setter命名为setPath,getter命名为Path,为什么不是getPath,因为get在Cocoa中有特殊的含义，这个含义就是带有get的方法就意味着这个方法通过形参指针(传入函数的参数指针)来返回值。我们要遵守这个命名习惯或者说规则。

在Objective-C 2.0中加入了@property和@synthesize来代替setter和getter，这两个关键字为编译器指令。还有点表达式，存取类成员的值时，可以使用点表达式。

Object.attribute,当点表达式在=号左边时，调用的是setter方法，在=号右边时，调用的

是getter方法。

@property 语法为:@property (参数) 类型 变量名.

在这里主要说明一下参数.

参数分为三种:

第一种:读写属性包括(readonly/readwrite/)

第二种:setter属性(assign,copy,retain),assign是简单的赋值，copy是释放旧成员变量，并新分配内存地址给成员变量，将传入参数内容复制一份，给成员变量。retain是将传入 参数引用计数加1，然后将原有的成员变量释放，在将成员变量指向该传入参数。

第三种:与多线程有关(atomic,nonatomic).当使用多线程时，使用atomic,在不使用多线程时使用nonatomic

对象创建与初始化

在Objective-C中创建对象有两种方法，一种是[类 new];另一种是[[类 alloc] init],这两种方法是等价的，但按惯例来讲，使用[[类 alloc] init];

alloc操作是为对象分配内存空间，并将对象的数据成员都初始，int 为0，BOOL 为NO, float 为0.0等。

初始化，默认的初始化函数为init,init返回值为id,为什么回返回id呢，因为要实现链式表达式，在Objective-C中叫嵌套调用。

为什么要嵌套调用？？因为初始化方法init返回值可能与alloc返回的对象不是同一个？为什么会发生这种情况？基于类簇的初始化，因为init可以接受参数，在init内部有可能根据不同的参数来返回不同种类型的对象，所以最会发生上面说的情况。

在初始化时，建议使用if (self = [super init])

便利初始化

当一个类需要根据不同的情况来初始化数据成员时，就需要便利初始化函数，与init初始化不同的是，便利初始化函数有参数，参数个数可以有1到N个，N是类数据成员个数。

指定初始化函数：什么是指定初始化函数？在类中，某个初始化函数会被指定为指定的初始化函数，确定指定初始化函数的规则是初始化函数中，参数最多的为指定初始化函数，

其它未被指定为指定初始化函数的初始化函数要调用指定初始化函数来实现。对于该类的子类也是一样，只要重写或者直接使用父类的指定初始化函数。上述文字有些绕，来个例子吧

```
@interface A{

int x;

int y;

}


-(id) init;

-(id) initWithX:(int) xValue;

-(id) initWithY:(int) yValue;

-(id) initWithXY:(int) xValue

    yVal:(int) yValue;

@end
```

这里initWithXY被确定为指定初始化函数。

```
-(id) initWithXY:(int) xValue

yVal:(int) yValue{
```

```
 if (self = [super init]){

x = xValue;

y = yValue;

}

return self;

}



-(id) init{

if (self = self initWithXY:10

yVal:20){

}

return self;

}

.......

@interface B: A{

int z;

}

-(jd) initWithXY......;

@end

@implementation B
```

```
-(id) initWithXY:(int) xValue

yVal:(int) yValue{

 if (self = [super initWithXY:10

yVal=20]){

z= 40;

}

return self;

}

@end
```

自动释放池

内存管理是软件代码中的重中之重，内存管理的好坏，直接影响着软件的稳定性。在Cocoa中，有自动释放池，这类似于C++中的智能指针。

NSObject有一个方法是autorelease，当一个对象调用这个方法时，就会将这个对象放入到自动释放池中。

drain,该方法是清空自动释放池，不是销毁它。drain方法只适用于Mac OS X 10.4以上的版本，在我们写的代码中要使用release，release适用于所有版本。

自动释放池是以栈的方式实现，当创建一个自动释放池A时，A被压入栈顶，这时将接入autorelease消息的对象放入A自动释放池，这时创建一个新的B自动释放池，B被压入栈顶，创建完成后删除B,这个接收autorelease消息的对象依然存在，因为A自动释放池依然存在。

引用计数

每个对象都有一个与之相应的整数，称它为引用计数，当该引用计数为0时，Objective-C自动向该对象发送dealloc,以销毁该对向，与该引用计数相关的方法(消息)有下面几个

1 增加引用计数：通过alloc,new,copy创建一个对象时，该对象的引用计数加1(其实就是1，因为之前为0)

2 增加引用计数: retain

3 减少引用计数: release

局部分配内存(临时对象):

1 如果使用alloc,new,copy创建对象，则需要主动调用对象的release方法

2 如果使用非alloc,new,copy创建对象，我们认为该对象引用计数为1，并已经加入了自动释放池，我们不需要主动的调用对象的release方法。

拥有对象(在类中以成员的方法存在):

1 如果使用alloc,new,copy创建对象，则需要在dealloc方法中，释放该对象

2 如果使用非alloc,new,copy创建对象，则在拥有该对象时，保留该对象(执行retain方法)，在dealloc方法中，释放该对象。

dealloc

当对象的引用计数为0时，Objective-C会自动发送对象的dealloc消息(自动调用对象的dealloc方法，类似于C++的析构函数)，所以我们可以自己重写dealloc方法，来实现类里的对其它使用资源的释放工作。

注意：不要直接在代码中显示调用dealloc方法。

垃圾回收

在Objective-C 2.0中引入了垃圾回收机制(自动管理内存)，在工程设置里设置Objective-C Garbage Collection为Required[-fobjc-gc-only]就可以使用垃圾回收机制。

启用垃圾回收机制后，通常的内存管理命令都变成了空操作指令，不执行任何操作。

Objective-C的垃圾回收机制是一种继承性的垃圾回收器，垃圾回收器定期检查变量和对象以及他们之间的指针，当发现没有任何变量指向对象时，就将该对象视为被丢弃的垃圾。所以在不在使用一个对象时，将指针他的指针设置为nil,这时垃圾回收器就会清理该对象。

注意：如果开发iPhone软件，则不能使用垃圾回收。在编写iPhone软件时，Apple公司建议不要在自己的代码中使用autorelease方法，并且不要使用创建自动释放对象的函数。

类别

什么是类别？类别是一种为现有类添加新方法的方式。

为什么使用类别或者说使用类别的目的是什么？有以下三点：

第一，可以将类的实现分散到多个不同的文件或多个不同的框架中。

如果一个类需要实现很多个方法，我们可以将方法分类，把分好的类形成类别，可以有效的管理和驾驭代码。

第二，创建对私有方法的前向引用。

第三，向对象添加非正式协议。

委托

委托的意思就是你自己想做某事，你自己不做，你委托给别人做。

在Ojbective-C中，实现委托是通过类别(或非正式协议)或者协议来实现。

举个例子：Apple要生产iPhone,Apple自己不生产(种种原因，其中之一就是在中国生产成本低，他们赚的银子多),Apple委托富士康来生产,本来富士康原来不生产iPhone,现在要生产了,所以他得自己加一个生产iPhone的生产线(类别，增加生产iPhone方法)，这就是通过类别来实现委托。下面用代码来说明这个例子。

.....

```
Apple *apple = [[Apple alloc ] init];

Foxconn *fox = [[Foxconn alloc] init];

[apple setDelegate:fox];

[apple produceIPhone];
```

.....

```
@implementation Apple

-(...) setDelegate:(id) x{

delegate  = x; //! 将委托的生产对象指定为x

}


-(...) produceIPhone{

[delegate produceIPhone]; //! 委托对象生产iPhone

}


@interface Foxconn : NSObject
```

...

@end

@interface NSObject(ProduceIPhone) //! Foxconn之前就可以生产其它产品，有过声明和定义

-(...) produceIPhone //! 增加生产iPhone能力

@end

@implementation NSObject(ProduceIPhone)

//! 生产iPhone

-(...) produceIPhone{

.....

}

@end

非正式协议

创建一个NSObject的类别， 称为创建一个非正式协议。为什么叫非正式协议呢?

也就是说可以实现，也可以不实现被委托的任务。

拿上面的例子来说，Apple要求Foxconn除了能生产iPhone外，还有一个要求是在一定时间内完成.由于双方没有签合同，所以时间要求和生产要求规格都是非正式协议

选择器

选择器就是一个方法的名称。选择器是在Objective-C运行时使用的编码方式，以实现快速查找。可以使用@selector预编译指令，获取选择器@selector(方法名)。NSObject提

供了一个方法respondsToSelector:的方法，来访问对象是否有该方法(响应该消息)。

拿上面的Apple请Foxconn生产iPhone为例，Apple怎么知道Foxconn有没有生产iPhone的能力呢?Apple就通过respondsToSelector方法询问Foxconn，是否可以生产iPhone(是否可以响应produceIPhone)，询问结果是可以，那Apple就委托Foxconn生产，Foxconn就生产出来了人们比较喜欢的iPhone产品。

正式协议

与非正式协议比较而言，在Ojbective-C中，正式协议规定的所有方法必须实现。在Ojbective-C2.0中，Apple又增加了两个关键字，协议中的方法也可以不完全实现，是哪个关键字见关键字部份的@optional,@required。

正式协议声明如下:

@protocol XXX

-(...) func1;

-(...) func2;

@end

使用协议:

@interface Object : NSObject //! Object从NSObject派生，并遵循XXX协议，要实现func1,func2函数。

...

@end

习惯用法

分配内存和初始化

self = [super init];

对象间交互

在Objective-C中，所有对象间的交互都是通过指针实现。

快速枚举

for (Type *p in array)

注意:

Objective-C不支持多继承

objective-c只不过是拥有一些附加特性的C语言。本质上就是C语言

1.C语言使用#include通知编译器应在头文件中查询定义。objective-c也可以使用#include来实现这个目的，但你永远不可能这么做，你会用#import，它是GCC编译器提供的，#import可以保证头文件只被包含一次。

xcode会使用预编译头文件（一种经过压缩的，摘要形式的头文件），在通过#import导入这种文件时，加载速度会非常快。

2.什么是框架

框架是一种聚集在一个单元的部件集合，包含头文件，库，图像，声音文件等。苹果公司将cocoa，Carbon，QuickTime和OpenGL等技术作为框架集提供。cocoa的组成部分有Foundation和 Application Kit框架。还有一个支持框架的套件，包含Core Animation和

Core Image，这为Cocoa增添了多种精彩 的功能。

每个框架都是一个重要的技术集合，通常包含数十个甚至上百个头文件。每个框架都有一个主头文件，它包含了所有框架的各个头文件。通过使用#import导入主头文件，可以使用所有框架的特性。

3.Foundation框架处理的是用户界面之下的层（layer）中得特性，例如数据结构和通信机制。

4.NS前缀

NS这个前缀告诉你函数来自cocoa而不是其他工具包。

两个不同事物使用相同标示符时会导致名称冲突，而前缀可以预防这个大问题。

5.BOOL类型

objective-c中得BOOL实际上是一种对带符号的字符类型（signed char）的定义。它使用8位存储空间，YES为1，NO为0.

6.间接

不在代码中直接使用某个值，而是使用指向该值的指针。另一个意思是，让别的类来完成本类的工作。

例子：1.循环次数的变量。变量与间接

2.使用从文件读取。文件与间接

在OOP（面向对象编程）中，间接十分重要。OOP使用间接来获取数据，OOP真正的革命性就是它在调用代码中使用间接。比如在调用函数时，不是直接调用，而是间接调

用。

7.过程式程序与OOP的区别

过程式程序建立在函数之上，数据为函数服务。面向对象编程从相反的角度来看待问题。它以程序的数据为中心，函数为数据服务。在OOP中，不在重点关注程序中得函数，而是专注与数据。

8.id

id是一种泛型，用于表示任何种类的对象。

9.OOP中得一些术语

类：类是一种结构，它表示对象的类型。对象引用类来获取和本身有关的各种信息，特别是运行什么代码来处理每种操作。

对象：对象是一种结构，它包含值和指向其类的隐藏指针。

实例：实例是“对象”的另一种称呼。

消息：消息是对象可以执行的操作，用于通知对象去做什么。

方法：方法是为响应消息而运行的代码。根据对象的类，消息可以调用不同的方法。

方法调度程序：是objective-c使用的一种机制，用于推测执行什么方法以响应某个特定的消息。

接口：接口是对象的类应该提供的特性的描述。接口不提供实现细节。

实现：实现是使接口正常工作的代码。

10.中缀符

objective-c有一种名为中缀符的语法技术。方法的名称及其参数都是合在 一起的。例如： [trxtThing setStringValue:@"Hello there" color:kBlueColor]; 中 setStringValue: 和 color:实际上是参数的名称（实际上是方法名称的一部分）。使代码可读性更强，更容易理解参数的用途。

11.先行短线

-(void)draw;

前面的短线表明这是objective-c方法的生命。这是一种区分函数原型与方法声明的方式，函数原型中没有先行短线。-代表是实例方法。+代表是类方法。

12.@interface

创建某个特定类的对象之前，objective-c编译器需要一些有关该类的信息。他必须知道对象的数据成员和它提供的特性可以使用@interface指令把这种信息传递给编译器。用于定义类的公共接口。

13.@implementation

是一个编译器指令，表明你将为某个类提供代码。类名出现在@implementation之后。该行的结尾处没有分号。因为在objective-c编译器指令后不必使用分号。

@interface和@implementation间的参数名不同是正确的。

在@interface中没有声明却在@implementation中实现的方法是私有方法。

14.实例化

创建对象的过程叫做实例化。实例化对象时，需要分配内存，然后这些内存被初始化并

保存一些有用的默认值，这些值不同于你在获得新分配内存时得到的随机值。内存分配和初始化完成后，就创建了一个新的对象实例。

15.继承

创建一个新类时，通常需要定义新类以区别于其他类以及现有类。使用继承可以定义一个具有父类所有功能的新类，它继承了父类的这些功能。

objective-c没有多继承。

创建一个新类时，其对象首先从自身的超类中继承实例变量，然后添加他们自己的实例变量。

超类

父类

子类

孩子类

重写

方法调度:objective-c的方法调度程序将子当前类中搜索响应的方法。如果调度程序无法在接受消息的对象类中找到响应的方法，它就会在该类的超类中进行查找。顺着继承链找到下一个超类进行查找，直到NSObject类中也没有该方法，则会出现运行时错误。

16.复合

对象引用其他对象时，可以利用其他对象提供的特性，这就是复合。

17.UML

UML是一种用图表表示类，类的内容以及他们之间关系的常见方法。

18.多态

使用更具体种类的对象（子类对象）代替一般类型（父类），这种能力称为多态性。

19.self

是一个指向接收消息的对象的指针。指向第一个实例变量isa。因为objective-c编译器已经看到了所有这些类的@interface声明，因此，它能直到对象中实力变量的布局，通过这些重要的信息，编译器可以产生代码并查找任何需要的实例变量。

基地址加偏移：编译器使用“基地址加偏移”机制实现奇妙的功能。给定的对象基地址，是指第一个实例变量的首个字节在内存中得位置，通过在该地址加上偏移地址，编译器就可以查到其他实例变量的位置。

20.间接寻址方式，直接寻址方式

21.super

objective-c提供某种方式来重写方法，并且仍然调用超类的实现方式。当需要超类实现自身的功能，同时在前面或者后面执行某些额外的工作时，这种机制非常有用。为了调用继承方法的实现，需要使用super作为方法调用的目标。

22.cocoa

cocoa实际上是由2个不同的框架组成的：Foundation Kit和 Application Kit。
Application Kit包含了所有的用户接口对象和高级类。

Foundation Kit

23.NSAutoreleasePool

mian()函数创建了(通过alloc)并初始化(通过init)了一个NSAutoreleasePool实例。在mian()函数结尾，这个池被排空。这就是Cocoa内存管理的预览。

24.NSRange

typedef struct _NSRange {

unsigned int location;

unsigned int length;

}NSRange;

这个结构体用来表示相关事务的范围，通常是字符串里的字符范围或者数组里的元素范围。

25.三种赋值方式

1.NSRange range;

range.location = 17;

range.length = 4;

2.C语言的聚合结构赋值机制

NSRange range = {17, 4};

3.Cocoa提供的一个快捷函数NSMakeRange()

NSRange range = NSMakeRange(17, 4);

使用NSMakeRange()的好处是你可以在任何能够使用函数的地方使用它，例如在方法调用中将其当成参数传递。

26.几何数据类型

1.NSPoint 代表笛卡儿平面中得一个点(x, y).

typedef struct _NSPoint {

float x;

float y;

}NSPoint;

2.NSSize 用来存储长度和宽度

typedef struct NSSize {

float width;

float height;

}NSSize;

3.NSRect 矩形数据类型，它是由点和大小复合而成

typedef struct _NSRect {

NSPoint origin;

NSSize size;

}NSRect;

27.字符串NSString

stringWithFormat:就是一个工厂方法，它根据你提供的参数创建新对象。

length:长度

isEqualToString:比较字符串内容是否相同

compart:将接受对象和传递来的字符串逐个字符的进行比较。返回一个enum数据

NSCaseInsensitiveSearch:不区分大小写字符。

NSLiteralSearch:进行完全比较，区分大小写

NSNumericSearch:比较字符串的字符个数，而不是字符值。

-(NSRange)rangeOfString:(NSString *)aString;

返回的range.start为开始位置，range.length为长度。

28.NSMutableString可变字符串。

stringWithCapacity:创建一个新的NSMutableString

字符串的大小并不仅限于所提供的容量，这个容量仅是个最优值。如果要创建一个40mb的字符串。

NSMutableString *str = [NSMutableString stringWithCapacity：42];

appendString接受参数aString，然后将其复制到接收对象的末尾。

appendFormat与stringWithFormat：类似，但它将格式化的字符串附加在接收字符串的末尾，而不是创建新的字符串对象。

29.集合家族

1.NSArray:是一个Cocoa类，用来存储对象的有序列表。

两个限制：1.只能存储objective-c的对象，不能存储C语言中得基本数据类型。

2.也不能存储nil。

30.枚举器，快速枚举

31.NSDictionary字典

关键字及其定义的集合。

32.NSNumber包装(以对象形式实现)基本数据类型

装箱：将一个基本类型的数据包装成对象。

取消装箱：从对象中提取基本类型的数据。

objective-c不支持自动装箱。

33.NSValue是NSNumber的父类。

+(NSValue *)valueWithBytes:(const void *)value objCType:(const char *)type;

传递的参数是你想要包装的数值的地址(如一个NSSize或你自己的 struct)。通常，得到的是你想要存储的变量的地址（在C语言中使用操作符&）。你也可以提供一个用来描述这个数据类型的字符串，通常用来说 明struct中实体的类型和大小。你不用自己写代码来生成这个字符串，@encode编译器指令可以接受数据类型的名称并为你生成合适的字

符串。

NSRect rect = NSMakeRect(1, 2, 30, 40);

NSValue *value;

value = [NSValue valueWithBytes:&rect objCType:@encode(NSRect)];

[array addObject:value];

34.NSNull只有一个方法[NSNull null];

[NSNull null]总是返回一样的数值，所以你可以使用运算符==将该值与其他值进行比较。

35.单实例架构：只需要一个实例。

查找文件：

例如：NSFileManager *manager;

manager = [NSFileManager defaultManager];

defaultManager可以为我们创建一个属于我们自己的NSFileManger对象。

NSString *home = [@"~" stringByExpandingTildeInPath];将~替换成当前用户的主目录。

NSDirectoryEnumerator *direnum = [manager enumeratorAtPath:home]; 返回一个NSDictionaryEnumerator，它是NSEnumerator的子类。每次在这个枚举器对象中调用nextObject时，都会 返回该目录中得一个文件的另一个路径。

36.内存管理

1)对象生命周期

对象的生命周期包括诞生(alloc或者new方法实现)，生存(接受消息和执行操作)，交友(借助方法的组合和参数)以及当他们的生命结束时最终死去(被释放)。当对象的生命周期结束时，他们的原材料(内存)将被回收以供新的对象使用。

2)引用计数

cocoa采用了一种称为引用计数的技术，有时候也叫保留计数。每个对象有一个与 之相关联的整数，称作它的引用计数器或保留计数器。当某段代码需要访问一个对象时，该代码将该对象的保留计数器值+1，表示“我要访问该对象”。当这段代 码结束对象访问时，将对象的保留计数器值-1，表示它不再访问该对象，当保留计数器值为0时，表示不再有代码访问该对象了，因此该对象被销毁，其占用的内 存被系统回收以便重用。

alloc，new，copy 1

retain +1

release -1

3)对象所有权

如果一个对象具有指向其他对象的实力变量，则称该对象拥有这些对象。

在类A中 B对象拥有其指向的C对象，则B对象拥有C对象。

如果一个函数创建了一个对象，则称该函数拥有它创建的这个对象。

main()函数创建了对象a 称main()函数拥有a对象

当多个实体拥有某个特定的对象时，对象的所有权关系就更加复杂了，这也是保留计数器值可能大于1的原因。

例子:

```
main() {

Engine *engine = [Engine new];
```

[car setEngine:engine];

}

现在哪个实体拥有engine对象？是main函数还是car类？哪个实体负责确 保当engine对象不再被使用时能够收到release消息？因为car类正在使用engine对象，所以不可能是main函数。因为main函数随后 还可能会使用engine对象，所以也不可能是car类。

解决方法是让car类保留engine对象，将engine对象的保留计数器值增 加到2。这是因为car类和main函数这2个实体都正在使用engine对象。car类应该在setEngine：方法中保留engine对象，而 main函数应该释放engine对象。然后，当car类完成其任务时再释放engine对象(在其dealloc方法中)，最后engine对象占用的 资源被回收。

如果您使用名字以“alloc”或“new”开头或名字中包含“copy”的方法（例如alloc，newObject或mutableCopy）创建了一个对象，则您会获得该对象的所有权；或者如果您向一个对象发送了一条retain消息，则您也会获得该对象的所有权。

4)访问方法中得保留和释放

5)自动释放池NSAutoreleasePool

是一个存放实体的池(集合)。你可以用NSMutableArray来编写自己的自动释放池，以容纳对象并在dealloc方法中向池中得所有对象发送release消息。

autorelease

当给一个对象发送autorelease消息时，实际上是将该对象添加到NSAutoreleasePool中。当自动释放池被销毁时，会向该池中得所有对象发送release消息。

6)自动释放池的销毁时间

在我们一直使用的Foudation库工具中，创建和销毁自动释放池的方法非常明确：

NSAutoreleasePool *pool;

pool = [[NSAutoreleasePool alloc] init];

...

[pool release];

创建一个自动释放池时，该池自动成为活动的池。释放该池时，其保留计数器值归0，然后该池被销毁。在销毁的过程中，该池释放其包含的所有对象。当使用Application Kit时，cocoa定期自动为你创建和销毁自动释放池。通 常是在程序处理完当前事件以后执行这些操作。你可以使用任意多得自动释放对象，当不再使用它们时，自动释放池将自动为你清理这些对象。

你可能已经在xcode自动生成代码中遇见过另一种销毁自动释放池中对象的方式：-drain方法。该方法只是清空自动释放池而不是销毁它。并且只适用于mac os x10.4以上的版本。

7)自动释放池的工作过程

我们在任何时候向一个对象发送autorelease消息，该对象都会呗添加到这 个自动释放池中。被加入到自动释放池的对象的引用计数器值不会变化。当自动释放池被销毁时(向自动释放池发送release消息，自动释放池的引用计数器 值变为0，调用自身的dealloc函数)，会调用自身的dealloc函数，会向池中得对象发送release消息。

37.cocoa内存管理规则

1)当你使用new，alloc或copy方法创建一个对象时，该对象的保留计数器值为1。当不再使用该对象时，你要负责向该对象发送一条release或autorelease消息。这样，该对象将在其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方法的使用次数相同。

38.清理自动释放池

由于自动释放池的销毁时间是完全确立的，所以它在循环执行过程中不会被销毁。在迭代中或者循环中，需要建立自己的自动释放池。

39.垃圾回收gc

自动内存管理机制。objective-c的垃圾回收器是一种继承性的垃圾回收 器。与那些已经存在了一段时间的对象相比，新创建的对象更可能被当成垃圾。垃圾回收器定期检查变量和对象以及他们之间的指针，当发现没有任何变量指向某个 对象时，就将该对象视为应该被丢弃的垃圾。如果你再一个实例变量中指向某个对象，一定要在某个时候使该实例变量赋值为nil，以取消对该对象的引用并使垃 圾回收器知道该对象可以被清理了。

与自动释放池一样，垃圾回收器也是在时间循环结束时才触发。

ARC是什么?

ARC是iOS 5推出的新功能，全称叫 ARC(Automatic Reference Counting)。简单地说，就是代码中自动加入了retain/release，原先需要手动添加的用来处理内存管理的引用计数的代码可以自动地由编译器完成了。

该机能在 iOS 5/ Mac OS X 10.7 开始导入，利用 Xcode4.2 可以使用该机能。简单地理解ARC，就是通过指定的语法，让编译器(LLVM 3.0)在编译代码时，自动生成实例的引用计数管理部分代码。有一点，ARC并不是GC，它只是一种代码静态分析（Static Analyzer）工具。

40.分配

是一个新对象诞生的过程。是从操作系统获得一块内存并将其指定为存放对象的实例变量的位置。

40.初始化

与分配对应的操作是初始化。初始化从操作系统取得一块内存。准备用于存储对象。

嵌套调用技术非常重要，因为初始化方法返回的对象可能与分配的对象不同。

1)初始化时做什么?

给实例变量赋值并创建你得对象完成任务所需要的其他对象。

2)便利初始化函数

许多类包含便利初始化函数。用来完成某些额外的工作的初始化方法，可以减少你自己完成这些工作的麻烦。

例如：NSString类中-(id)initWithFormat:(NSString *)format,...;

3)指定初始化函数

类中的某个初始化方法被指派为指定初始化函数。该类所有初始化方法使用指定初始化函数执行初始化操作。子类使用其超类的指定初始化函数实现超类的初始化。

例如：其他初始化函数通过指定初始化函数实现。

41.初始化函数规则

不需要为你自己的类创建初始化函数方法。如果不需要设置任何状态，或者只需要alloc方法将内存清零的默认行为，则不需要担心init。

如果构造了一个初始化函数，则一定要在你自己的指定初始化函数中调用超类的指定初

始化函数。一定要将超类的初始化函数的值赋给self对象，并返回你自己的初始化方法的值。因为超类可能决定返回一个完全不同的对象。

如果初始化函数不止一个，则要选择一个座位指定初始化函数。被选定的方法应该调用超类指定初始化函数。要按照指定初始化函数的形式实现所有其他初始化函数，就像我们在前面的实现一样。

42.特性@property

objective-c2.0的特性只适用于mac os x 10.5或更高版本。特性主要应用于cocoa的新组件(尤其是华丽夺目的core Animation效果)。

1)简化接口

@property预编译指令的作用是自动生命属性的setter和getter方法。

2)@synthesize也是一种新的编译器功能，表示“创建该属性的访问器”。

3)点表达式

如果点表达式出现在=左边，该属性名称的setter方法(set方法)将被调用。如果点表达式出现在对象变量右边，则该属性名称的getter方法(get方法)将被调用。

4)特性扩展

@property()括号里面的东西,是对应在set方法中要添加的语句。比如我在括号里写retain，就相当于在它的set方法里添加了一句 [xx retain]。

@property属性

属性分为3类:

1.读写属性（Writability）包含：readwrite / readonly

2.setter语义（Setter Semantics）包含：assign / retain / copy

3.原子性（Atomicity）包含：nonatomic

下面具体说明各个属性的含义

readwrite / readonly：

决定是否生成set访问器，readwrite是默认属性，生成getter和setter方法；
readonly只生成getter方法，不生成setter方法。
readonly关键字代表setter不会被生成， 所以它不可以和 copy/retain/assign组合使用。

assign / retain / copy:

这些属性用于指定set访问器的语义，也就是说，这些属性决定了以何种方式对数据成员赋予新值。

assign:

直接赋值，索引计数不改变，适用于简单数据类型，例如：NSIngeter、CGFloat、int、char等。

retain：

指针的拷贝，使用的是原来的内存空间。
对象的索引计数加1。
此属性只能用于Objective-C对象类型，而不能用于Core Foundation对象。(原因很明显，retain会增加对象的引用计数，而基本数据类型或者Core Foundation对象都没有引用计数)。

copy：

对象的拷贝，新申请一块内存空间，并把原始内容复制到那片空间。
新对象的索引计数为1。
此属性只对那些实行了NSCopying协议的对象类型有效。

很多Objective-C中的object最好使用用retain，一些特别的object（例如：string）使用copy。

nonatomic：

非原子性访问，不加同步，多线程并发访问会提高性能。如果不加此属性，则默认是两个访问方法都为原子型事务访问。默认值是atomic，为原子操作。
（atomic是Objc使用的一种线程保护技术，基本上来讲，是防止在写未完成的时候被另外一个线程读取，造成数据错误。而这种机制是耗费系统资源的，所以在iPhone这种小型设备上，如果没有使用多线程间的通讯编程，那么nonatomic是一个非常好的选择。）

5)保留周期retain cycle

引用计数器在该周期中归零。

6)什么是属性访问器

属性访问器（Property Accessor），包括 get 访问器和 set 访问器分别用于字段的读写操作

其设计目的主要是为了实现面向对象（OO）中的封装思想。根据该思想，字段最好设为private，一个精巧的类最好不要直接把字段设为公有提供给客户调用端直接访问

另外要注意属性本身并不一定和字段相联系

7)self.a与a的区别

self.a使编译器知道我们期望使用访问器访问a。如果只使用裸名a，编译器将假设我们直接修改了实例变量。

8)self.a = nil

这行代码表示使用nil参数调用setName:方法。生成的访问器方法将自动释放以前的name对象，并使用nil替代a。该方法完成了释放name对象所占用内存的操作。当然，也可以只释放name对象以清理其占用的内存。如果你再dealloc方法以外的地方清除特性，那么使用"将nil赋值给对象"的方法可以将特性设置为nil，同时可以使我们避免对已释放内存的悬空引用问题。

9)特性不是万能的

有些方法不适合特性所能涵盖的方法的相当狭小的范围。特性不支持那些需要接受额外

参数的方法。

43.类别@category

1)声明类别

@interface NSString (NumberConvenience)

-(NSNumber *)lengthAsNumber;

@end

该声明具有2个特点。首先，现有类位于@interface关键字之后，其后是位于圆括号中的一个新名称。该声明表示，类别的名称是NumberConvenience，而且该类别将向NSString类中添加方法。只要保证类别名称的唯一性，你可以向一个类中添加任意多得类别。

其次，你可以指定希望向其添加类别的类以及类别的名称，而且你还可以列出添加的方法，最后以@end结束。由于不能添加新实现变量，因此与类声明不同的是，类别的声明中没有实例变量部分。

2)实现类别

3)类别的局限性

第一，无法向类中添加新的实例变量。类别没有位置容纳实例变量。

第二，名称冲突，即类别中得方法与现有的方法重名。当发生名称冲突时，类别具有更高的优先级。你得类别方法将完全取代初始方法，从而无法再使用初始方法。有些编程人员在自己的类别方法中增加一个前缀，以确保不发生名称冲突。

有一些技术可以克服类别无法增加新实例变量的局限。例如，可以使用全局字典存储对象与你想要关联的额外变量之间的映射。但此时你可能需要认真考虑一下，类别是否是完成当前任务的最佳选择。

4)类别的作用

cocoa中得类别主要用于3个目的：第一，将类的实现分散到不同文件或者不同框架中。第二，创建对私有方法的前向引用。第三，向对象添加非正式协议。

44.run循环是一种cocoa构造，它一直处于阻塞状态(即不执行任何处理)，知道某些事件发生为止。

45.响应选择器

一个类别如何知道其委托对象是否能够处理那些发送给它的消息?

类别首先检查对象，询问其能否响应该选择器。如果该对象能够响应该选择器，

1)选择器@selector()

选择器只是一个方法名称，但它以objective-c运行时使用的特殊方式编码，以快速执行查询。你可以使用@selector()预编译指令选择器，其中方法名位于圆括号中。

46.委托 非正式协议

47.正式协议

与非正式协议一样，正式协议是一个命名的方法列表。但与非正式协议不同的是，正式协议要求显式的采用协议。采用协议的办法是在类的@interface声明中列出协议名称。采用协议意味着你承诺实现该协议的所有方法。

1)声明协议

@protocol NSCopying

-(id)copyWithZone:(NSZone *)zone;

@end

2)采用协议

@interface Car: NSObject <NSCopying>

{

}

@end

3)协议和数据类型

如果一个用尖括号括起来的协议名称跟随在id之后，则编译器将知道你期望任意类型的对象，只要其遵守该协议。

4)objective-c2.0的新特性@optional @required

@optional可选择实现的方法

@required必须实现的方法

因此cocoa中得非正式协议正被带有@optional的正式协议所取代。

48.Application Kit

1)IBOutlet与IBAction

他们实际上只是APPKit提供的#defines。IBOutlet的定义没有任何作用，因此将不会对它进行编译。IBAction定义为void，这意味着在AppController中声明的方法的返回类型将使void。IBOutlet和IBAction不执行任何操作，他们并不是用于编译的，实际上他们是为Interface Builder以及阅读代码的人提供的标记。通过查找IBOutlet和IBAction，Interface Builder知道AppController对象具有两个能够连接的实例变量。

2)IBOutlet是如何工作的

当加载nib文件时(MainMenu.nib会在应用程序启动时自动加载，可以创建你自己的nib文

件并自行加载)，存储在nib文件中得任何对象都会被重新创建。这意味着会再后台执行alloc和init方法。所以，当应用程序启动时，会分配并初始化一个AppController实例。在执行init方法期间，所有IBOutlet实例变量都为nil。只有创建了nib文件中得所有对象(这包括窗口和文本域和按钮),所有连接才算完成。

一旦建立了所有连接(也就是将NSTextField对象的地址添加到AppController的实例变量中)，会向创建的每个对象发送消息awakeFromNib。一个非常常见的错误是试图在init方法中使用IBOutlet执行一些操作。由于所有实例变量都为nil，发送给他们的所有消息不执行任何操作，所以在init中得任何尝试都会发生无提示失败。(这是Cocoa导致效率低和占用大量调试时间的一个方面)。如果你想知道为什么这些操作不起作用，可以使用NSLog输出实例变量的值，并查看他们是否都为nil。对于创建的对象和发送的awakeFromNib消息，都不存在预定义顺序。

文件加载与保存

49.属性列表

1)自动释放对象

NSDate

NSData NSData对象是不可改变的。他们被创建后就不能改变。可以使用他们，但不能改变其中的内容。

2)编码对象　编码和解码

cocoa具备一种机制来将对象自身转换为某种格式并保存到磁盘中。对象可以将他们的实例变量和其他数据块编码为数据块，然后保存到磁盘中。以后将这些数据块读回到内存中，并且还能基于保存的数据创建新对象。这个过程称为编码和解码。或称为序列化和反序列化。

50.键/值编码　KVC

是一种间接改变对象状态的方式，其实现方法是使用字符串描述要更改的对象状态部

分。

1)valueForKey与setValue:forKey:

这两种方法的工作方式相同。他们首先查找名称的setter(getter)方法，如果不存在setter(getter)方法，他们将在类中查找名为名称或_名称的实例变量。然后给它赋值(取值)。无需通过对象指针直接访问实例变量。

2)路径

键路径的深度是任意的，具体取决于对象图。

键路径不仅能引用对象值，还可以引用一些运算符来进行一些运算，例如获取一组值的平均值或返回这组值中得最小值和最大值。

例如：NSNumber *count;

```
count = [garage valueForKeyPath:@"cars.@count"];

NSLog(@"We have %@ cars", count);
```

我们将路径“cars.@count”拆开，cars用于获取cars属性，它是来自garage的NSArray类型的值。接下来的部分是@count，其中@符号意味着后面将进行一些运算。

和 cars@sun.mileage

最大值 cars@min.mileage

最小值 cars@max.mileage

3)整体操作

KVC非常棒的一点是，如果向NSArray请求一个键值，它实际上会查询数组中得每个对象来查找这个键值，然后将查询结果打包到另一个数组中并返回给你。这种方法也适用于通过键路径访问的对象内部的数组。

4)批处理

KVC包含两个调用，可以使用他们对对象进行批量更改。第一个调用是dictionaryWith-ValuesForKeys：。它接受一个字符串数组。该调用获取一些键，对每个键使用

valueForKey：，然后为键字符串和刚才获取的值构建一个字典。

1、IBM面试题：平面上画1999条直线，最多能将平面分成多少部分?

分析：

没有直线时有一个空间；（1）

1条直线时，这条这些可以将这个空间分成两个；（1+1）

2条直线时，第二条直线可以和第一条直线相交，这样第二条直线可以将两个空间分成四个；（1+1+2）

....

注意到画每条直线时能增加多少个空间，取决于此直线从多少个空间中通过。

而从多少个空间中通过，取决于和多少条直线相交。

例如，如果一条直线和其它5条直线相交，那么最大可以通过6个空间，此直线可以增加6个子空间。

画每条直线时，能相交的直线数为总的已经画过的直线。

所以总的空间数最多为

1+1+2+3+...+1999 = 1999001

2、IBM面试题：使用两根烧1小时的香，确定15分钟的时间

不均匀分布的香，每根香烧完的时间是一个小时，你能用什么方法来确定一段15分钟的时间?

分析：第一根点燃两头，第二根只点一头。

当第一根烧完时，时间过去了30分钟，所以第二根还能烧30分钟。这时点燃第二根的另外一头，第二根香还能烧的时间就是15分钟。

3、IBM面试题：27个人去买矿泉水

有27个人去买矿泉水，商店正好在搞三个空矿泉水瓶可以换一瓶矿泉水的活动，他们至少要买几瓶矿泉水才能每人喝到一瓶矿泉水?

分析：19

4、百度面试题：将多个集合合并成没有交集的集合

给定一个字符串的集合，格 式如：**{aaa bbb ccc}， {bbb ddd}，{eee fff}，{ggg}，{ddd hhh}**要求将其中交集不为空的集合合并，要求合并完成后的集合之间无交集，例如上例应输出**{aaa bbb ccc ddd hhh}，{eee fff}，{ggg}**。

（1）请描述你解决这个问题的思路；

（2）请给出主要的处理流程，算法，以及算法的复杂度

（3）请描述可能的改进。

[java] view plaincopy

```
1. import java.io.BufferedReader;
2. import java.io.IOException;
3. import java.io.InputStreamReader;
4. import java.util.Arrays;
5. import java.util.LinkedList;
6. import java.util.List;
7. import java.util.ListIterator;
8. import java.util.regex.Pattern;
9. /*实现思路:
10. 将接受的字符串进行切割存入一个二维的字符串数组，使用二重循环去找出有交集的数组并合并，将被合并的数组置为空;
11. 由于使用二维数组，而数组定义时必须定义大小，给算法的实现带来了不必要的麻烦；原本打算使用二维的list，但是使用起来有错误，原因尚不明确，但是使用
12. list应该会优于数组;
13. 由于在算法执行的数组合并的过程中是直接将新出现的元素直接插入到数组的尾部，所以最后在输出过滤的时候还要将数组内的元素进行排序。
14. 我使用了冒泡排序和插入排序两种方法。
15. 纯属练手，高手勿喷！o(∩_∩)o 哈哈*/
16. public class BaiDu_UnionStringSet {
17.     static String ReadString() throws IOException {
18.         /* 接收输入字符串 */
19.         String s = new String();
20.         BufferedReader stdin = new BufferedReader(new InputStreamReader(
21.                 System.in));
22.         System.out
23.                 .println("输入格式如{aaa bbb ccc},{bbb ddd},{eee fff},
    {ggg hhh},{ddd hhh}的字符串: ");
```

```
24.         s = stdin.readLine();
25.         stdin.close();
26.         return s;
27.     }
28.
29.     static String[][] String_to_List(String s) {
30.         /* 将输入的数据存入一个二维字符串数组，本来打算使用二维的list，结果老是报
    错。 */
31.         List list;
32.         String ss = new String();
33.         s = s.replace("{", "").replace("}", "");// 去掉 { 和 }
34.         list = Arrays.asList(Pattern.compile(",").split(s));// 将字符串
    以“,”进行切割生成一个数组再转换成一个list（可以直接用数组）
35.         String[][] sz = new String[list.size()][3 * list.size()];// 定义一个
    二维数组，其中有一维的大小不知道，这样是不合适的，这也是为什么我想用list的原因
36.         for (int i = 0; i < list.size(); i++) {
37.             ss = (String) list.get(i);
38.             sz[i] = Pattern.compile(" ").split(ss);
39.         }
40.         return sz;
41.     }
42.
43.     static boolean ComStr(String[] s1, String[] s2) {
44.         /* 判断两个字符串数组中是否有交集 */
45.         int i, j;
46.         for (i = 0; i < s1.length; i++)
47.             for (j = 0; j < s2.length;) {
48.                 if (s1[i] != null && s2[j] !
    = null && s1[i].equals(s2[j])) {
49.                     return true;
50.                 } else
51.                     j++;
52.             }
53.         return false;
54.     }
55.
56.     static boolean belong(String s, String[] s1) {
57.         /* 判断字符串s是否出现在s1数组中 */
58.         for (int i = 0; i < s1.length;) {
59.             if (s1[i] != null) {
60.                 if (s.equals(s1[i]))
61.                     return true;
```

```
62.                 else
63.                     i++;
64.             } else
65.                 i++;
66.         }
67.         return false;
68.     }
69.
70.     static String[] MergeStr(String[] s1, String[] s2) {
71.         /* 将两个字符数组去重后融合在一起 */
72.         String[] s = new String[s1.length + s2.length];
73.         int i, j;
74.         for (i = 0; i < s1.length; i++)
75.             s[i] = s1[i];
76.         for (i = 0, j = 0; j < s2.length; j++) {
77.             if (s1 != null && s2[j] != null) {
78.                 if (!belong(s2[j], s1)) {
79.                     s[s1.length + i] = s2[j];
80.                     i++;
81.
82.                 }
83.             }
84.         }
85.         return s;
86.     }
87.
88.     static void findUnion(String[][] s) {
89.         /* 找出二维数组s中符合条件的字符串合并，并打印 */
90.         int i, j;
91.         for (i = 0; i < s.length; i++) {
92.             for (j = 0; j < s.length; j++) {
93.                 if (s[i] != null && s[j] != null && i != j) {
94.                     if (ComStr(s[i], s[j])) {
95.                         s[i] = MergeStr(s[i], s[j]);
96.                         s[j] = null;
97.                     }
98.                 }
99.             }
100.          }
101.          for (int k = 0; k < s.length; k++) {
102.              if (s[k] != null) {
103.                  OutPutString(s[k]);
```

```
104.                 }
105.             }
106.         }
107.
108.         static void OutPutString(String[] str) {
109.             System.out.println("使用冒泡排
    序: "+"\n"+SortString_bubbling(RemoveNull(str)));
110.             System.out.println("使用插入排
    序: "+"\n"+SortString_insert(RemoveNull(str)));
111.         }
112.
113.         static List RemoveNull(String[] str) {/* 使用List去除null */
114.             List nonull_list = new LinkedList();
115.             for (int i = 0; i < str.length; i++) {
116.                 if (str[i] != null) {
117.                     nonull_list.add(str[i]);
118.                 }
119.             }
120.             return nonull_list;
121.         }
122.
123.         static List SortString_bubbling(List list) {
124.             /* 冒泡排序 */
125.             String s1 = new String();
126.             String s2 = new String();
127.             for (int i = 0; i < list.size(); i++) {
128.                 for (int j = i; j < list.size(); j++) {
129.                     s1 = list.get(i).toString();
130.                     s2 = list.get(j).toString();
131.                     if (s1.compareTo(s2) > 0) {
132.                         list.set(i, s2);
133.                         list.set(j, s1);
134.                     }
135.                 }
136.             }
137.             return list;
138.         }
139.
140.         static List SortString_insert(List list) {
141.             /* 插入排序 */
142.             List sortedlist = new LinkedList();
143.             ListIterator x = list.listIterator(0);
```

```
144.             String str = new String();
145.             String comstr = new String();
146.             String comstr_a = new String();
147.             int count, moveid, flag;
148.             while (x.hasNext()) {
149.                 str = x.next().toString();
150.                 flag = -1;// 用来判断元素是否已经插入新表中
151.
152.                 for (count = 0; flag == -1 && count < sortedlist.size(); count++) {
153.                     if (sortedlist.size() == 0) {
154.                         sortedlist.set(count, str);
155.                         flag = 0;
156.                     } else {
157.                         comstr = sortedlist.get(count).toString();
158.                         if (str.compareTo(comstr) < 0) {
159.
160.                             for (moveid = sortedlist.size(); moveid > count; moveid--) {
161.                                 comstr_a = sortedlist.get(moveid - 1).toString();
162.                                 if (moveid == sortedlist.size()) {
163.                                     sortedlist.add(moveid, comstr_a); // 增加并不替换原位置上的元素，原元素后移
164.                                 } else
165.                                     sortedlist.set(moveid, comstr_a);// 替换原位置元素；但是不能对表尾操作
166.
167.                             }// 后移
168.                             sortedlist.set(count, str);
169.                             flag = 0;// 插入元素
170.                         }
171.                     }
172.                 }
173.                 if (flag == -1) {
174.                     sortedlist.add(str);// 插在表尾
175.                     flag = 0;
176.                 }
177.             }
178.             return sortedlist;
179.         }
180.
```

```
179.          public static void main(String[] args) throws IOException {
180.              findUnion(String_to_List(ReadString()));
181.          }
182.      }
```

4、网易面试题：new/delete和malloc/free的区别

new/delete和malloc/free的区别，并说说你在什么情况下会自另行建立自己的内存分配。

解析：

new/delete：给定数据类型，new/delete会自动计算内存大小，并进行分配或释放。如果是对类进行操作，new/delete还会自动调用相应的构造函数和析构函数。

malloc/free：没有进行任何数据类型检查，只负责分配和释放给定大小的内存空间。

有些情况下，new/delete和malloc/free都不能满足性能的要求，我们需要自建内存分配来提高效率。比如，如果程序需要动态分配大量很小的对象，我们可以一次分配可以容纳很多小对象的内存，将这些小对象维护在链表中，当程序需要时直接从链表中返回一个。

5、百度面试题：有两个文件，各含50M和500个url，找出共同的url

一个大的含有50M个URL的记录，一个小的含有500个URL的记录，找出两个记录里相同的URL。

解析：

首先使用包含500个url的文件创建一个hash_set。

然后遍历50M的url记录，如果url在hash_set中，则输出此url并从hash_set中删除这个url。

所有输出的url就是两个记录里相同的url。（复杂度为：O(m×n)）

AC自动机算法也可以解决。参见：http://www.cppblog.com/mythit/archive/2009/04/21/80633.html

6、微软面试题：删除链表中的重复项

一个没有排序的链表，比如list={a,l,x,b,e,f,f,e,a,g,h,b,m}，请去掉重复项，

并保留原顺序，以上链表去掉重复项后为newlist={a,l,x,b,e,f,g,h,m}，请写出一个高效算法(时间比空间更重要)。

解析：

建立一个hash_map，key为链表中已经遍历的节点内容，开始时为空。

从头开始遍历链表中的节点：

- 如果节点内容已经在hash_map中存在，则删除此节点，继续向后遍历；
- 如果节点内容不在hash_map中，则保留此节点，将节点内容添加到hash_map中，继续向后遍历。

[java] view plaincopy

```
1.  import java.util.HashMap;
2.  import java.util.LinkedList;
3.  import java.util.Map;
4.  public class hashmap_noduplicates {
5.  static void removeduplicates(String[] str)
6.  {
7.      LinkedList list = new LinkedList();
8.      Map hm=new HashMap();
9.      String s=new String();
10.     int flag=0;
11.     for (int i = 0; i < str.length; i++) {
12.       list.add(str[i]);}
13.     for(int counter=0;counter<str.length;counter++){
14.         s=str[counter];
15.         if(!hm.containsKey(s)){hm.put(s, null);}
16.         else {
17.             list.remove(counter-flag);//list链表remove一个元素过后，它的size也
    减小了
18.             flag++;
19.         }
20.     }
21.     System.out.println(list);
22.     }
23.
24. public static void main(String[] args){
25.     String[] str={"a","l","x","b","e","f","f","e","a","g","h","b","m"};
26.     removeduplicates(str);
```

```
27. }
28. }
```

7、谷歌面试题：找到两个字符串的公共字符，并按照其中一个的排序

写一函数f(a,b)，它带有两个字符串参数并返回一串字符，该字符串只包含在两个串中都有的字符并按照在a中的顺序。写一个版本算法复杂度O(N^2)和一个O(N)。

解析：

O(N^2)：

对于a中的每个字符，遍历b中的每个字符，如果相同，则拷贝到新字符串中。

O(N)：

首先使用b中的字符建立一个hash_map，对于a中的每个字符，检测hash_map中是否存在，如果存在则拷贝到新字符串中。

[java] view plaincopy

```
1.  import java.util.HashMap;
2.
3.
4.  public class find_samechars {
5.  static String find_samechars_n2(String s1,String s2){
6.      char[] c1=s1.toCharArray();
7.      char[] c2=s2.toCharArray();
8.      char c;
9.      String s=new String();
10.     int i,j,flag;
11.     for(i=0;i<c1.length;i++){
12.         c=c1[i];
13.         flag=-1;
14.         for(j=0;flag==-1 && j<c2.length;j++){
15.             if(c==c2[j]){flag=0;}
16.         }
```

```
17.         if(flag==0){s=s+c;}
18.     }
19.     return s;
20. }
21. static String find_samechars_n(String s1,String s2){
22.     HashMap hm=new HashMap();
23.     String s=new String();
24.     char[] c1=s1.toCharArray();
25.     char[] c2=s2.toCharArray();
26.     for(int i=0;i<c2.length;i++){
27.         hm.put(c2[i],null);
28.     }
29.     for(int j=0;j<c1.length;j++){
30.         if(hm.containsKey(c1[j])){s=s+c1[j];}
31.     }
32.     return s;
33.
34. }
35. public static void main(String[] args){
36.     String s1="ijklmnabcdefgh";
37.     String s2="azbyixhwgvcn";
38.     System.out.println("s1="+s1);
39.     System.out.println("s2="+s2);
40.     System.out.println("时间复杂度为N^2: "+find_samechars_n2(s1,s2));
41.     System.out.println("时间复杂度为N: "+find_samechars_n(s1,s2));
42. }
43. }
```

8、谷歌面试题：如何尽快找到一个好人

有n个人，其中超过半数是好人，剩下的是坏人。好人只说真话，坏人可能说真话也可能说假话。这n个人互相都知道对方是好人还是坏人。

现在要你从这n个人当中找出一个好人来，只能通过以下方式：

每次挑出两个人，让这两个人互相说出对方的身份，你根据两个人的话进行判断。

问通过何种方法才能最快的找出一个好人来。(要考虑最坏的情况)

解析：

先找一个人A，然后其他所有人评价A

1如果半数说A是好人：A是好人

2如果半数以上说A是坏人：A是坏人

－ － － － －

如果A是坏人

去掉说A是好人的人（一定是坏人）

在剩下的人里找一个人重复上面的（这里好人肯定更多于一半）

－ － －

递归进行

最坏情况就是坏人都说实话且每次运气不好都选出的是坏人

这时复杂度是O(n^2)

实际计算时选出好人的几率越来越高的，因为坏人不断的被去掉

(好吧，我承认我的智商太低了)

一些面试题，转关注的一个博客

分类： IOS 2013-05-20 23:31 149人阅读 评论(0) 收藏 举报

腾讯面试题：tcp三次握手的过程，accept发生在三次握手哪个阶段?

答accept发生在三次握手之后。

第一次握手：客户端发送syn包(syn=j)到服务器。

第二次握手：服务器收到syn包，必须确认客户的SYN（ack=j+1），同时自己也发送一个ASK包（ask=k）。

第三次握手：客户端收到服务器的SYN＋ACK包，向服务器发送确认包ACK(ack=k+1)。

三次握手完成后，客户端和服务器就建立了tcp连接。这时可以调用accept函数获得此连接。

const的含义及实现机制，比如：const int i,是怎么做到i只可读的?

const用来说明所定义的变量是只读的。

这些在编译期间完成，编译器可能使用常数直接替换掉对此变量的引用。

用UDP协议通讯时怎样得知目标机是否获得了数据包

可以在每个数据包中插入一个唯一的ID，比如timestamp或者递增的int。

发送方在发送数据时将此ID和发送时间记录在本地。

接收方在收到数据后将ID再发给发送方作为回应。

发送方如果收到回应，则知道接收方已经收到相应的数据包；如果在指定时间内没有收到回应，则数据包可能丢失，需要重复上面的过程重新发送一次，直到确定对方收到。

求一个论坛的在线人数，假设有一个论坛，其注册ID有两亿个，每个ID从登陆到退出会向一个日志文件中记下登陆时间和退出时间，要求写一个算法统计一天中论坛的用户在线分布，取样粒度为秒。

一天总共有 3600*24 = 86400秒。

定义一个长度为86400的整数数组int delta[86400]，每个整数对应这一秒的人数变化值，可能为正也可能为负。开始时将数组元素都初始化为0。

然后依次读入每个用户的登录时间和退出时间，将与登录时间对应的整数值加1，将与退出时间对应的整数值减1。

这样处理一遍后数组中存储了每秒中的人数变化情况。

定义另外一个长度为86400的整数数组int online_num[86400]，每个整数对应这一秒的论坛在线人数。

假设一天开始时论坛在线人数为0，则第1秒的人数online_num[0] = delta[0]。第n+1秒的人数online_num[n] = online_num[n-1] + delta[n]。

这样我们就获得了一天中任意时间的在线人数。

在一个文件中有 10G 个整数，乱序排列，要求找出中位数。内存限制为 2G。

不妨假设10G个整数是64bit的。

2G内存可以存放256M个64bit整数。

我们可以将64bit的整数空间平均分成256M个取值范围，用2G的内存对每个取值范围内出现整数个数进行统计。这样遍历一边10G整数后，我们便知道中数在那个范围内出现，以及这个范围内总共出现了多少个整数。

如果中数所在范围出现的整数比较少，我们就可以对这个范围内的整数进行排序，找到中数。如果这个范围内出现的整数比较多，我们还可以采用同样的方法将此范围再次分成多个更小的范围（256M=2^28，所以最多需要3次就可以将此范围缩小到1，也就找到了中数）。

两个整数集合A和B，求其交集。

1. 读取整数集合A中的整数，将读到的整数插入到map中，并将对应的值设为1。

2. 读取整数集合B中的整数，如果该整数在map中并且值为1，则将此数加入到交集当中，并将在map中的对应值改为2。

通过更改map中的值，避免了将同样的值输出两次。

2. 也可以将A和B分别排序，然后利用归并的思想搞定。

有1到10w这10w个数，去除2个并打乱次序，如何找出那两个数?

申请10w个bit的空间，每个bit代表一个数字是否出现过。

开始时将这10w个bit都初始化为0，表示所有数字都没有出现过。

然后依次读入已经打乱循序的数字，并将对应的bit设为1。

当处理完所有数字后，根据为0的bit得出没有出现的数字。

首先计算1到10w的和，平方和。

然后计算给定数字的和，平方和。

两次的到的数字相减，可以得到这两个数字的和，平方和。

所以我们有

x + y = n

x^2 + y^2 = m

解方程可以得到x和y的值。

有1000瓶水，其中有一瓶有毒，小白鼠只要尝一点带毒的水24小时后就会死亡，至少要多少只小白鼠才能在24小时时鉴别出那瓶水有毒?

最容易想到的就是用1000只小白鼠，每只喝一瓶。但显然这不是最好答案。

既然每只小白鼠喝一瓶不是最好答案，那就应该每只小白鼠喝多瓶。那每只应该喝多少瓶呢?

首先让我们换种问法，如果有x只小白鼠，那么24小时内可以从多少瓶水中找出那瓶有

毒的?

由于每只小白鼠都只有死或者活这两种结果，所以x只小白鼠最大可以表示2^x种结果。如果让每种结果都对应到某瓶水有毒，那么也就可以从2^x瓶水中找到有毒的那瓶水。那如何来实现这种对应关系呢?

第一只小白鼠喝第1到2^(x-1)瓶，第二只小白鼠喝第1到第2^(x-2)和第2^(x-1)+1到第2^(x-1) + 2^(x-2)瓶....以此类推。

回到此题，总过1000瓶水，所以需要最少10只小白鼠。

根据上排给出十个数，在其下排填出对应的十个数, 要求下排每个数都是上排对应位置的数在下排出现的次数。上排的数：0，1，2，3，4，5，6，7，8，9。

0，1，2，3，4，5，6，7，8，9

6，2，1，0，0，0，1，0，0，0

通过一个循环做，三次循环搞定.

任意0-N，都是N-3的位置是1，前面是N-4,2,1，其他是0

给40亿个不重复的unsigned int的整数，没排过序的，然后再给几个数，如何快速判断这几个数是否在那40亿个数当中?

unsigned int 的取值范围是0到2^32-1。我们可以申请连续的2^32/8=512M的内存，用每一个bit对应一个unsigned int数字。首先将512M内存都初始化为0，然后每处理一个数字就将其对应的bit设置为1。当需要查询时，直接找到对应bit，看其值是0还是1即 可。

IBM面试题：c++中引用和指针有什么不同？指针加上什么限制等于引用?

引用不是一个变量，它只表示该引用名是目标变量名的一个别名，它本身不是一种数据类型，因此引用本身不占存储单元，系统也不给引用分配存储单元。引用一经确定就不能修改。

指针是一个变量，需要在内存中分配空间，此空间中存储所指对象的地址。由于指针是一个普通变量，所以其值还可以通过重新赋值来改变。

把指针定义为const后，其值就不能改变了，功能和引用类似，但有本质的区别。

谷歌面试题：1024! 末尾有多少个0?

末尾0的个数取决于乘法中因子2和5的个数。显然乘法中因子2的个数大于5的个数，所以我们只需统计因子5的个数。

是5的倍数的数有： 1024 / 5 = 204个

是25的倍数的数有：1024 / 25 = 40个

是125的倍数的数有：1024 / 125 = 8个

是625的倍数的数有：1024 / 625 = 1个

所以1024! 中总共有204+40+8+1=253个因子5。

也就是说1024! 末尾有253个0。

谷歌面试题：给定能随机生成整数1到5的函数，写出能随机生成整数1到7的函数

只要我们可以从 n 个数中随机选出 1 到 n 个数，反复进行这种运算，直到剩下最后一个数即可。

我们可以调用 n 次给定函数，生成 n 个 1 到 5 之间的随机数，选取最大数所在位置即可满足以上要求。

例如

初始的 7 个数 [1,2,3,4,5,6,7].

7 个 1 到 5 的随机数 [5, 3,1,4,2,5,5]

那么我们保留下[1,6,7],

3 个1 到 5 的随机数[2,4,1]

那么我们保留下[6]

6 就是我们这次生成的随机数。

产生K个数(k>1) 假定产生的数分别为n1,n2,n3,n4...

那么定义产生的数为n1-1+（n2-2）*5+（n3-1）*5^2+（n4-1）*5^3.....

于是产生的数位于区间（0，5^k-1)

然后把5^k分成k等分，产生的数位于哪个等分就是那个产生的随机数（0~6），然后+1即可

如果位于k等分的余数范围，则重新执行一次上述过程

不用担心余数问题，当k取3时落到余数范围的概率就已经降低为6/125

判断一个自然数是否是某个数的平方。当然不能使用开方运算。

假设待判断的数字是 N。

方法1：

遍历从1到N的数字，求取平方并和N进行比较。

如果平方小于N，则继续遍历；如果等于N，则成功退出；如果大于N，则失败退出。

复杂度为O(n^0.5)。

方法2：

使用二分查找法，对1到N之间的数字进行判断。

复杂度为O(log n)。

方法3：

由于

(n+1)^2

=n^2 + 2n + 1，

= ...

= 1 + (2*1 + 1) + (2*2 + 1) + ... + (2*n + 1)

注意到这些项构成了等差数列（每项之间相差2）。

所以我们可以比较 N-1， N - 1 - 3， N - 1 - 3 - 5 ... 和0的关系。

如果大于0，则继续减；如果等于0，则成功退出；如果小于 0，则失败退出。

复杂度为O(n^0.5)。不过方法3中利用加减法替换掉了方法1中的乘法，所以速度会更快些。

给定一个未知长度的整数流，如何随机选取一个数?

方法1.

将整个整数流保存到一个数组中，然后再随机选取。

如果整数流很长，无法保存下来，则此方法不能使用。

方法2.

如果整数流在第一个数后结束，则我们必定会选第一个数作为随机数。

如果整数流在第二个数后结束，我们选第二个数的概率为1/2。我们以1/2的概率用第2个数替换前面选的随机数，得到满足条件的新随机数。

....

如果整数流在第n个数后结束，我们选第n个数的概率为1/n。我们以1/n的概率用第n个数替换前面选的随机数，得到满足条件的新随机数。

....

利用这种方法，我们只需保存一个随机数，和迄今整数流的长度即可。所以可以处理任意长的整数流。

设计一个数据结构，其中包含两个函数，1.插入一个数字，2.获得中数。并估计时间复杂度。

1. 使用数组存储。

插入数字时，在O(1)时间内将该数字插入到数组最后。

获取中数时，在O(n)时间内找到中数。（选数组的第一个数和其它数比较，并根据比较结果的大小分成两组，那么我们可以确定中数在哪组中。然后对那一组按照同样的方法进一步细分，直到找到中数。）

2. 使用排序数组存储。

插入数字时，在O(logn)时间内找到要插入的位置，在O(n)时间里移动元素并将新数字插入到合适的位置。

获得中数时，在O(1)复杂度内找到中数。

3. 使用大根堆和小根堆存储。

使用大根堆存储较小的一半数字，使用小根堆存储较大的一半数字。

插入数字时，在O(logn)时间内将该数字插入到对应的堆当中，并适当移动根节点以保持两个堆数字相等（或相差1）。

获取中数时，在O(1)时间内找到中数。

谷歌面试题：在一个特殊数组中进行查找

给定一个固定长度的数组，将递增整数序列写入这个数组。当写到数组尾部时，返回数组开始重新写，并覆盖先前写过的数。请在这个特殊数组中找出给定的整数。

假设数组为a[0, 1, ..., N-1]。

我们可以采用类似二分查找的策略。

首先比较a[0]和a[N/2]，如果a[0] < a[N/2]，则说明a[0,1,...,N/2]为递增子序列，否则另一部分是递增子序列。

然后判断要找的整数是否在递增子序列范围内。如果在，则使用普通的二分查找方法继续查找；如果不在，则重复上面的查找过程，直到找到或者失败为止。

谷歌面试题：给定两个已排序序列，找出共同的元素

不妨假设序列是从小到大排序的。定义两个指针分别指向序列的开始。

如果指向的两个元素相等，则找到一个相同的元素；如果不等，则将指向较小元素的指针向前移动。

重复执行上面的步骤，直到有一个指针指向序列尾端。如果两个数组大小差不多，用你的方法就行了，如果数组大小差得很多，就遍历小的，然后在大的里二分查找~

编程实现两个正整数的除法，当然不能用除法操作符。

```
// return x/y.
int div(const int x, const int y) {
....
}
int div(const int x, const int y) {
int left_num = x;
int result = 0;
while (left_num >= y) {
    int multi = 1;
    while (y * multi <= (left_num >> 1)) {
```

```
        multi = multi << 1;
    }
    result += multi;
    left_num -= y * multi;
}
return result;
}
```

微软面试题：计算n bit的整数中有多少bit 为1

设此整数为x。

方法1：

让此整数除以2，如果余数为1，说明最后一位是1，统计值加1。

将除得的结果进行上面运算，直到结果为0。

方法2：

考虑除法复杂度有些高，可以使用移位操作代替除法。

将 x 和 1 进行按位与操作（x&1），如果结果为1，说明最后一位是1，统计值加1。

将x 向右一位（x >> 1），重复上面过程，直到移位后结果为0。

方法3：

如果需要统计很多数字，并且内存足够大，可以考虑将每个数对应的bit为1的数量记录下来，这样每次计算只是一次查找操作。

微软面试题：快速求取一个整数的7倍

乘法相对比较慢，所以快速的方法就是将这个乘法转换成加减法和移位操作。

可以将此整数先左移三位（×8）然后再减去原值：X << 3 - X。

微软面试题：判断一个数是不是2的n次幂

设要判断的数是无符号整数X。

首先判断X是否为0，如果为0则不是2的n次幂，返回。

X和X-1进行按位与操作，如果结果是0，则说明这个数是2的n次幂；如果结果非0，则说明这个数不是2 的n次幂。

证明：

如果是2的n次幂，则此数用二进制表示时只有一位是1，其它都是0。减1后，此位变成0，后面的位变成1，所以按位与后结果是0。

如果不是2的n次幂，则此数用二进制表示时有多位是1。减1后，只有最后一个1变成0，前面的 1还是1，所以按位与后结果不是0。

微软面试题：判断数组中是否包含重复数字

给定一个长度为N的数组，其中每个元素的取值范围都是1到N。判断数组中是否有重复的数字。（原数组不必保留）

方法1.

对数组进行排序（快速，堆），然后比较相邻的元素是否相同。

时间复杂度为O(nlogn)，空间复杂度为O(1)。

方法2.

使用bitmap方法。

定义长度为N/8的char数组，每个bit表示对应数字是否出现过。遍历数组，使用bitmap对数字是否出现进行统计。

时间复杂度为O(n)，空间复杂度为O(n)。

方法3.

遍历数组，假设第 i 个位置的数字为 j ，则通过交换将 j 换到下标为 j 的位置上。直到所有数字都出现在自己对应的下标处，或发生了冲突。

时间复杂度为O(n)，空间复杂度为O(1)。

微软面试题：删除链表中的重复项

一个没有排序的链表，比如list={a,l,x,b,e,f,f,e,a,g,h,b,m}，请去掉重复项，并保留原顺序，以上链表去掉重复项后为newlist={a,l,x,b,e,f,g,h,m}，请写出一个高效算法(时间比空间更重要)。

建立一个hash_map，key为链表中已经遍历的节点内容，开始时为空。

从头开始遍历链表中的节点：

- 如果节点内容已经在hash_map中存在，则删除此节点，继续向后遍历；

- 如果节点内容不在hash_map中，则保留此节点，将节点内容添加到hash_map中，继续向后遍历。

微软面试题：编一个程序求质数的和

编一个程序求质数的和，例如F(7) = 2+3+5+7+11+13+17=58。

方法1：

对于从2开始的递增整数n进行如下操作：

用 [2，n-1] 中的数依次去除n，如果余数为0，则说明n不是质数；如果所有余数都不是0，则说明n是质数，对其进行加和。

空间复杂度为O(1)，时间复杂度为O(n^2)，其中n为需要找到的最大质数值（例子对应的值为17）

方法2：

可以维护一个质数序列，这样当需要判断一个数是否是质数时，只需判断是否能被比自己小的质数整除即可。

对于从2开始的递增整数n进行如下操作：

用 [2，n-1] 中的质数（2，3，5，7，开始时此序列为空）依次去除n，如果余数为0，则说明n不是质数；如果所有余数都不是0，则说明n是质数，将此质数加入质数序列，并对其进行加和。

空间复杂度为O(m)，时间复杂度为O(mn)，其中m为质数的个数（例子对应的值为7），n为需要找到的最大质数值（例子对应的值为17）。

方法3：

也可以不用除法，而用加法。

申请一个足够大的空间，每个bit对应一个整数，开始将所有的bit都初始化为0。

对于已知的质数（开始时只有2），将此质数所有的倍数对应的bit都改为1，那么最小的值为0的bit对应的数就是一个质数。对新获得的质数的倍数也进行标注。

对这样获得的质数序列累加就可以获得质数和。

空间复杂度为O(n)，时间负责度为O(n)，其中n为需要找到的最大质数值（例子对应的值为17）

微软面试题：给出一种洗牌算法

给出洗牌的一个算法，并将洗好的牌存储在一个整形数组里。

假设数组Card[0 - 53]中的54个数对应54张牌，从第一张牌（i = 0）开始直到倒数第二张牌（i = 52），每次生成一个[i, 53]之间的数r，将Card[i]和Card[r]中的数互换。

微软面试题：找到两个单向链表的第一个公共节点

如果两个单向链表有公共节点，则两个链表会构成Y型结构，最后一个节点相同。

我们可以从头开始遍历两个链表，找到最后一个节点的指针，设为p_a，p_b。同时记录下两个链表的长度len_a，len_b（假设len_a >= len_b)。

如果p_a == p_b，则说明两个链表有公共节点，否则没有。

如果有公共节点，则第一个公共节点距起始节点的距离满足 len_a - start_a == len_b - start_b。

所以第一个可能的公共节点距起始节点的距离是 len_a - len_b, 0。我们从这两个节点开始比较，直到找到第一个公共节点。

微软面试题：如何在链表里如何发现循环链接?

解答:

从链表的开始处，由两个指针A和B同时开始遍历链表。指针A每向前移动一步，指针B都向前移动两步。如果在移动了N步以后，指针A和B指向了同一个节点，则此链表中存在循环链表。

分析：

当然还可以在遍历的过程中存储节点的地址，通过不断的比较地址来判断有没有循环链表。但这种算法会使用更多的内存。

如果考官比较变态，还可以直接考复制链表。如果复制前没有测试循环链表，那不好意思，只能扣分了

谷歌面试题：找到链表的倒数第m个节点

方法1：

首先遍历链表，统计链表的长度N。

然后再次遍历链表，找到第N-m个节点，即为倒数第m个节点。

方法2：

使用两个指针，并使它们指向的节点相距m-1个。

然后同时向前移动两个指针，当一个指针指最后一个节点时，第二个指针指向倒数第m个节点。

两个方法的复杂度都是O(n)。

但是当N较大而m较小时，方法2可能会更快一些。因为方法2能更好利用CPU的缓存。

谷歌面试题：给定一个排序数组，如何构造一个二叉排序树?

采用递归算法。

选取数组中间的一个元素作为根节点，左边的元素构造左子树，右边的节点构造有子树。

谷歌面试题：数组中是否有两个数的和为10

1.

比较任意两个数的和是否为10。如

for (int i = 0; i < n; ++i) { for (int j = i+1; j < n; ++j) { .... }}

复杂度为O(n*n)。

2.

将数组排序后，对每个数m，使用二分查找在数组中寻找10-m。

复杂度为O(nlogn)。

3.

将数组存储到hash_set中去，对每个数m，在hash_set中寻找10-m。

复杂度为O(n)。

4.

如果数组很大，超过内存的容量，可以按照hash(max(m, 10-m))%g，将数据分到g个小的group中。然后对每个小的group进行单独处理。

复杂度为O(n)。

谷歌面试题：找到两个字符串的公共字符，并按照其中一个的排序

写一函数f(a,b)，它带有两个字符串参数并返回一串字符，该字符串只包含在两个串中都有的并按照在a中的顺序。写一个版本算法复杂度O(N^2)和一个O(N) 。

O(N^2)：

对于a中的每个字符，遍历b中的每个字符，如果相同，则拷贝到新字符串中。

O(N)：

首先使用b中的字符建立一个hash_map，对于a中的每个字符，检测hash_map中是否存在，如果存在则拷贝到新字符串中。

在给定整数序列中，找出最大和的子序列

给定一个整数序列，其中有些是负数，有些是正数，从该序列中找出最大和的子序列。比如：-5，20，-4，10，-18，子序列[20，-4，10]具有最大和26。

```
 int GetMaxSubArraySum(int* array, int array_len) {
`    int current_sum = 0;
```

```
    int max_sum = 0;
    for (int i = 0; i < array_len; ++i) {
      current_sum += array[i];
      if (current_sum > max_sum) {
        max_sum = current_sum;
      } else if (current_sum < 0) {
        current_sum = 0;
      }
    }
    return max_sum;
}
```

谷歌面试题：将无向无环连通图转换成深度最小的树

已知一个无向无环连通图T的所有顶点和边的信息，现需要将其转换为一棵树，要求树的深度最小，请设计一个算法找到所有满足要求的树的根结点，并分析时空复杂度。

最简单直接的方法就是把每个节点都试一遍：

假设某个节点为根节点，计算树的深度。当遍历完所有节点后，也就找到了使树的深度最小的根节点。

但这个方法的复杂度很高。如果有n个节点，则时间复杂度为O(n^2)。

树的深度取决于根节点到最深叶节点的距离，所以我们可以从叶节点入手。

叶节点会且只会和某一个节点连通（反之不成立，因为根节点也可能只和一个节点连通），所以我们很容易找到所有可能的叶节点。

题目可以等价于找到了两个叶节点，使得两个叶节点之间的距离最远。根节点就是这两个叶节点路径的中间点（或者中间两个点的任意一个）。

我们可以每次都将连接度为1的节点删掉，直到最后只剩下1个或2个节点，则这一个节点，或者两个节点中的任意一个，就是我们要找的根节点。

谷歌面试题：将字符串中的小写字母排在大写字母的前面

有一个由大小写组成的字符串，现在需要对它进行修改，将其中的所有小写字母排在大写字母的前面（大写或小写字母之间不要求保持原来次序）。

初始化两个int变量A和B，代表字符串中的两个位置。开始时A指向字符串的第一个字符，B指向字符串的最后一个字符。

逐渐增加A的值使其指向一个大写字母，逐渐减小B使其指向一个小写字母，交换A，B所指向的字符，然后继续增加A，减小B....。

当A>=B时，就完成了重新排序。

谷歌面试题：如何拷贝特殊链表

有一个特殊的链表，其中每个节点不但有指向下一个节点的指针pNext，还有一个指向链表中任意节点的指针pRand，如何拷贝这个特殊链表?

拷贝pNext指针非常容易，所以题目的难点是如何拷贝pRand指针。

假设原来链表为A1 -> A2 ->... -> An，新拷贝链表是B1 -> B2 ->...-> Bn。

为了能够快速的找到pRand指向的节点，并把对应的关系拷贝到B中。我们可以将两个链表合并成

A1 -> B1 -> A2 -> B2 -> ... -> An -> Bn。

从A1节点出发，很容易找到A1的pRand指向的节点Ax,然后也就找到了Bx，将B1的pRand指向Bx也就完成了B1节点pRand的拷贝。依次类推。

当所有节点的pRand都拷贝完成后，再将合并链表分成两个链表就可以了。

谷歌面试题：10分钟内看到一辆车的概率是多少?

如果在高速公路上30分钟内看到一辆车开过的几率是0.95，那么在10分钟内看到一辆车开过的几率是多少？(假设为常概率条件下)

假设10分钟内看到一辆车开过的概率是x，那么没有看到车开过的概率就是1-x，30分钟没有看到车开过的概率是(1-x)^3，也就是0.05。所以得到方程

(1-x)^3 = 0.05

解方程得到x大约是0.63。

百度面试题：从输入url到显示网页，后台发生了什么?

简单来说有以下步骤：

1. 查找域名对应的IP地址。这一步会依次查找浏览器缓存，系统缓存，路由器缓存，

ISP DNS缓存，根域名服务器。

2. 向IP对应的服务器发送请求。

3. 服务器响应请求，发回网页内容。

4. 浏览器解析网页内容。

当然，由于网页可能有重定向，或者嵌入了图片，AJAX，其它子网页等等，这4个步骤可能反复进行多次才能将最终页面展示给用户。

百度面试题：设计DNS服务器中cache的数据结构

要求设计一个DNS的Cache结构，要求能够满足每秒5000以上的查询，满足IP数据的快速插入，查询的速度要快。（题目还给出了一系列的数据，比如：站点数总共为5000万，IP地址有1000万，等等）

DNS服务器实现域名到IP地址的转换。

每个域名的平均长度为25个字节（估计值），每个IP为4个字节，所以Cache的每个条目需要大概30个字节。

总共50M个条目，所以需要1.5G个字节的空间。可以放置在内存中。（考虑到每秒5000次操作的限制，也只能放在内存中。）

可以考虑的数据结构包括hash_map，字典树，红黑树等等。

百度面试题：将多个集合合并成没有交集的集合

给定一个字符串的集合，格式如：{aaa bbb ccc}， {bbb ddd}，{eee fff}，{ggg}，{ddd hhh}要求将其中交集不为空的集合合并，要求合并完成后的集合之间无交集，例如上例应输出{aaa bbb ccc ddd hhh}，{eee fff}， {ggg}。

（1）请描述你解决这个问题的思路；

（2）请给出主要的处理流程，算法，以及算法的复杂度

（3）请描述可能的改进。

集合使用hash_set来表示，这样合并时间复杂度比较低。

1. 给每个集合编号为0，1，2，3...

2. 创建一个hash_map，key为字符串，value为一个链表，链表节点为字符串所在集合的编号。

遍历所有的集合，将字符串和对应的集合编号插入到hash_map中去。

3. 创建一个长度等于集合个数的int数组，表示集合间的合并关系。例如，下标为5的元素值为3，表示将下标为5的集合合并到下标为3的集合中去。

开始时将所有值都初始化为-1，表示集合间没有互相合并。

在集合合并的过程中，我们将所有的字符串都合并到编号较小的集合中去。

遍历第二步中生成的hash_map，对于每个value中的链表，首先找到最小的集合编号（有些集合已经被合并过，需要顺着合并关系数组找到合并后的集合编号），然后将链表中所有编号的集合都合并到编号最小的集合中（通过更改合并关系数组）。

4.现在合并关系数组中值为-1的集合即为最终的集合，它的元素来源于所有直接或间接指向它的集合。

算法的复杂度为O(n)，其中n为所有集合中的元素个数。

题目中的例子：

0: {aaa bbb ccc}

1: {bbb ddd}

2: {eee fff}

3: {ggg}

4: {ddd hhh}

生成的hash_map，和处理完每个值后的合并关系数组分别为

aaa: 0。[-1, -1, -1, -1, -1]

bbb: 0, 1。[-1, 0, -1, -1, -1]

ccc: 0。[-1, 0, -1, -1, -1]

ddd: 1, 4。[-1, 0, -1, -1, 0]

eee: 2。[-1, 0, -1, -1, 0]

fff: 2。[-1, 0, -1, -1, 0]

ggg: 3。[-1, 0, -1, -1, 0]

hhh: 4。[-1, 0, -1, -1, 0]

所以合并完后有三个集合，第0，1，4个集合合并到了一起，

第2，3个集合没有进行合并。

百度面试题：用C语言将输入的字符串在原串上倒序

```
 void revert(char* str) {
`    char c;
`    for (int front = 0, int back = strlen(str) - 1;
`         front < back;
`         ++front, --back) {
`      c = str[back];
`      str[back] = str[front];
`      str[front] = c;
`    }
` }
```

百度面试题：找出给定字符串对应的序号

序列Seq=[a,b,…z,aa,ab…az,ba,bb,…bz,…,za,zb,…zz,aaa,…] 类似与excel的排列，任意给出一个字符串s=[a-z]+(由a-z字符组成的任意长度字符串），请问s是序列Seq的第几个。

注意到每满26个就会向前进一位，类似一个26进制的问题。

比如ab，则位置为26*1 + 2；

比如za，则位置为26*26 + 1；

比如abc，则位置为26*26*1 + 26*2 + 3

百度面试题：找出第k大的数字所在的位置

写一段程序，找出数组中第k大小的数，输出数所在的位置。例如{2，4，3，4，7}中，第一大的数是7，位置在4。第二大、第三大的数都是4，位置在1、3随便输出哪一个均可。

先找到第k大的数字，然后再遍历一遍数组找到它的位置。所以题目的难点在于如何最高效的找到第k大的数。

我们可以通过快速排序，堆排序等高效的排序算法对数组进行排序，然后找到第k大的

数字。这样总体复杂度为O(N logN)。

我们还可以通过二分的思想，找到第k大的数字，而不必对整个数组排序。

从数组中随机选一个数t，通过让这个数和其它数比较，我们可以将整个数组分成了两部分并且满足，{x, xx, ..., t} < {y, yy, ...}。

在将数组分成两个数组的过程中，我们还可以记录每个子数组的大小。这样我们就可以确定第k大的数字在哪个子数组中。

然后我们继续对包含第k大数字的子数组进行同样的划分，直到找到第k大的数字为止。

平均来说，由于每次划分都会使子数组缩小到原来1/2，所以整个过程的复杂度为O(N)。

百度面试题：找到满足条件的数组

给定函数d(n) = n + n的各位之和，n为正整数，如 d(78) = 78+7+8=93。 这样这个函数可以看成一个生成器，如93可以看成由78生成。

定义数A：数A找不到一个数B可以由d(B)=A，即A不能由其他数生成。现在要写程序，找出1至10000里的所有符合数A定义的数。

申请一个长度为10000的bool数组，每个元素代表对应的值是否可以有其它数生成。开始时将数组中的值都初始化为false。

由于大于10000的数的生成数必定大于10000，所以我们只需遍历1到10000中的数，计算生成数，并将bool数组中对应的值设置为true，表示这个数可以有其它数生成。

最后bool数组中值为false的位置对应的整数就是不能由其它数生成的。

百度面试题：对正整数，算得到1需要操作的次数

实现一个函数，对一个正整数n，算得到1需要的最少操作次数。

操作规则为：如果n为偶数，将其除以2；如果n为奇数，可以加1或减1；一直处理下去。

例子：

func(7) = 4，可以证明最少需要4次运算

n = 7

n-1 6

n/2 3

n-1 2

n/2 1

要求：实现函数(实现尽可能高效) int func(unsign int n)；n为输入，返回最小的运算次数。

给出思路(文字描述)，完成代码，并分析你算法的时间复杂度。

```
int func(unsign int n) {
if (n == 1) {
return 0;
}
if (n%2 == 0) {
return 1 + func(n/2);
}
int x = func(n+1);
int y = func(n-1);
if (x > y) {
return y + 1;
} else {
return x + 1;
}
}
```

假设n表示成二进制有x bit，可以看出计算复杂度为O(2^x)，也就是O(n)。

```
 int func(unsign int n) {
`    if (n == 1) {
`      return 0;
`    }
`    if (n % 2 == 0) {
`      return 1 + func(n/2);
```

```
    }
    if (n == 3) {
      return 2;
    }
    if ( n & 2) {
      return 1 + func(n + 1);
    } else {
      return 1 + func(n - 1);
    }
}
```

百度面试题：找出N！后面的0的个数

容易理解，题目等价于求因子2和因子5出现的次数。

对于因子2来说，数字2，4，6，8，10....2n...中存在因子2，这样就获得了 N/2 （其中N/2只取结果的整数部分）个因子2。这些数字去除因子2后，变成1，2，3....N/2，又可以提取N/4个因子2....这样一直到只剩下1 个数（1）。所以N！中总共可以获得N/2 + N/4 + N/8 +....个因子2。

同理，N！中可以获得N/5 + N/25 + ... 个因子5。

尾部连续0的个数就是因子2和因子5较少的那个。

对于题目中的例子，18！中包含9+4+2+1个因子2，包含3个因子5。所以尾部有3个连续0。

计算的复杂度为O(logN)。

百度面试题：找出被修改过的数字

n个空间（其中n<1M），存放a到a+n-1的数，位置随机且数字不重复，a为正且未知。现在第一个空间的数被误设置为-1。已经知道被修改的数不是最小的。请找出被修改的数字是多少。

例如：n=6，a=2，原始的串为5, 3, 7, 6, 2, 4。现在被别人修改为-1, 3, 7, 6, 2, 4。现在希望找到5。

由于修改的数不是最小的，所以遍历第二个空间到最后一个空间可以得到a的值。

a 到 a+n-1这 n个数的和是 total = na + (n - 1)n/2。

将第二个至最后一个空间的数累加获得 sub_total。

那么被修改的数就是 total - sub_total。

百度面试题：在100w个数中找最大的前100个数

应该使用某种数据结构保存迄今最大的100个数。每读到一个新数时，将新数和保存的100个数中的最小一个相比较，如果新数更大些，则替换。这样扫描一遍100w个数也就获得了最大的100个数。

对于保存的100个数的数据结构，应该在最小复杂度的条件下满足

1）可以获得最小的数；

2）将最小数替换为另一个数后可以重新调整，使其可以满足条件1。

可见小根堆可以满足这些条件。

所以应该采用小根堆+扫描的方法。

方法1：类似《算法导论》中用二分法求第K大数，理想TC是O(n)。

百度面试题：正向最大匹配分词，怎么做最快?

用所有词生成一个字典树，匹配的过程就是查字典的过程。

假设我们有两个词"百度"，"百家姓"，那么生成的字典树就是：

百---度*

|

|-----家----姓*

其中"度"和"姓"旁边的星号表示这是一个有效词。

对于句子"百度面试题"，首先在字典中找"百"，找到了；继续向下查找"度"，又找到了；继续向下查找"面"，没有找到。那么"百度"就是我们分出来的第一个词。

还可以用hash_map来做。

首先用所有的词生成一个hash_map，假设我们有两个词“百度，“百家姓”，那么生成hash_map如下：

{

百：0

百度：1

百家：0

百家姓：1

}

其中值为0表示对应的key不是一个词，但有更长的词包括这个key；值为1表示这是一个词。

对于句子“百度面试题”，首先在hash_map中查找“百”，找到对应值为0，继续；查找“百度”，找到对应值为1，说明这是一个词，记下来并继续；查找“百度面”，没有找到，说明没有更长的词包含“百度面”。所以“百度”就是我们分出来的第一个词。

和字典法相比，hash_map法可能会用到更多的存储空间（因为有些字，比如“百”字，都存储了多次。但这还取决于字典树的具体实现），但程序设计会更加简单，不容易出错。

session和cache的区别是什么？

session是针对单个连接（会话）来使用的，主要存储和连接相关的上下文信息，比如登录信息等等。

cache是应用程序级的，主要用来缓存计算结果，减轻服务器负担，并加快响应速度。

百度面试题：找出数组中出现次数超过一半的数

答案：

创建一个hash_map，key为数组中的数，value为此数出现的次数。遍历一遍数组，用hash_map统计每个数出现的次数，并用两个值存储目前出现次数最多的数和对应出现的次数。

这样可以做到O(n)的时间复杂度和O(n)的空间复杂度，满足题目的要求。

但是没有利用“一个数出现的次数超过了一半”这个特点。也许算法还有提高的空间。

答案2：

使用两个变量A和B，其中A存储某个数组中的数，B用来计数。开始时将B初始化为0。

遍历数组，如果B=0，则令A等于当前数，令B等于1；如果当前数与A相同，则B=B+1；如果当前数与A不同，则令B=B-1。遍历结束时，A中的数就是要找的数。

这个算法的时间复杂度是O(n)，空间复杂度为O(1)。

百度面试题：如何找出字典中的兄弟单词

给定一个单词a，如果通过交换单词中字母的顺序可以得到另外的单词b，那么定义b是a的兄弟单词。现在给定一个字典，用户输入一个单词，如何根据字典找出这个单词有多少个兄弟单词?

答案：

使用hash_map和链表。

首先定义一个key，使得兄弟单词有相同的key，不是兄弟的单词有不同的key。例如，将单词按字母从小到大重新排序后作为其key，比如bad的key为abd，good的key为dgoo。

使用链表将所有兄弟单词串在一起，hash_map的key为单词的key，value为链表的起始地址。

开始时，先遍历字典，将每个单词都按照key加入到对应的链表当中。当需要找兄弟单词时，只需求取这个单词的key，然后到hash_map中找到对应的链表即可。

这样创建hash_map时时间复杂度为O(n)，查找兄弟单词时时间复杂度是O(1)。

网易面试题：new/delete和malloc/free的区别

new/delete：给定数据类型，new/delete会自动计算内存大小，并进行分配或释放。如果是对类进行操作，new/delete还会自动调用相应的构造函数和析构函数。

malloc/free：没有进行任何数据类型检查，只负责分配和释放给定大小的内存空间。

有些情况下，new/delete和malloc/free都不能满足性能的要求，我们需要自建内存分配来提高效率。比如，如果程序需要动态分配大量很小 的对象，我们可以一次分配可以容纳很多小对象的内存，将这些小对象维护在链表中，当程序需要时直接从链表中返回一个。还有一点，new返回指定类型的指 针；而malloc返回void*，必须强制类型转化。

有个比较有意思的地方是:int *p=(void*)malloc(1);可以编译并运行。

网易面试题：没有拷贝构造函数和重载＝运算符的string类

c++中，一个没有拷贝构造函数和重载＝运算符的string类，会出现什么问题，如何解决?

如果没有定义拷贝构造函数和重载=运算符，则系统会自动生成逐位拷贝的函数。

当我们用string初始化string时，（比如 string a("abc"); string b = a;），两个对象会指向同样的内存地址。在两个对象的析构函数中，我们会对同一个内存块调用两次删除，导致不确定的结果。

当我们将一个string赋值给另外一个string时，（比如 string a("abc"); string b(“cde"); b = a;）除了上面的多次调用析构函数的问题外，由于原来对象b指向的数据没有被正确删除，会导致内存泄漏。

解决办法：

1. 添加这两个函数。

2. 不使用这两个函数。

- 不用string初始化string：可以使用string a（”abc"); string b(a.c_str()); 代替。

- 不用string给string赋值，包括不能通过传值方法传递string参数：尽量使用指针。

网易面试题：写一段程序，实现atoi(const char* s)方法

atoi用于将字符串转换成为整数。

比如“123” =》 123，“-246” =》 -246。

```
int atoi(const char*s) {
  int result = 0;
  bool is_plus = true;
  if (*s == '+') {
    ++s;
  } else if (*s == '-') {
    ++s;
    is_plus = false;
  }
```

```
    while (*s >= '0' && *s <= '9') {
      result = result * 10 + *s - '0';
      ++s;
    }
    if (is_plus) {
      return result;
    } else {
      return -result;
    }
  }
```

网易面试题：给出若干个单词，组成字典，要求查找速度最快。

为使查找速度最快，可以要使用hash_map。

如果每个单词还有对应的解释和例句，可以将解释和例句对应的指针存放在hash_map的值中。或许可以尝试使用 TRIE 结构。

迅雷面试题：门面模式的解释、适用场合?

门面模式又被称为外观模式，为子系统中的一组接口提供一个一致的界面，该模式定义了一个高层接口，使得这个子系统更加容易使用。

举个例子：在做项目或产品的过程中进行跨部门合作的时候，每个部门都有个相应的接口人，那么我们只需和对应部门的接口人交互即可。

适用场合：

为一个复杂子系统提供一个简单接口：子系统往往因为不断演化而变得越来越复杂，使用门面模式可以使得子系统更具有可复用性。

子系统的独立性：引入门面模式将一个子系统与它的客户端以及其他子系统分离，可以提高子系统的独立性和可移植性。

层次化结构：在构建一个层次化的系统时，可以使用 门面模式定义系统中每一层的入口。如果层与层之间是相互依赖的，则可以限定它们仅通过门面进行通信，简化层与层之间的依赖关系。

迅雷面试题：AJAX的原理、如何实现刷新及其优点

AJAX即“Asynchronous JavaScript and XML”（异步JavaScript和XML)，是指一种创建交互式网页应用的网页开发技术。

使用了AJAX技术的网页，利用Javascript和服务器通信，获取数据，然后再通过修改网页的DOM中的某些元素来实现刷新网页的特定部分。

使用了AJAX技术后，由于只需要更新网页的一部分，而不是全部，所以和服务器交互的数据比较少。这就降低了服务器的负载，并提高了用户端的响应速度。另外，AJAX并不需要在浏览器中安装插件。

迅雷面试题：数组与链表的区别?

在数组中，元素在内存中连续存放。对于访问操作，由于元素类型相同，占用内存相同，所以可以通过数组的下标计算出元素所在的内存地址，便于快速访问。但对于插入或删除操作，需要移动大量元素，所以速度比较慢。

在链表中，元素在内存中没有连续存放，而是通过元素中的指针将各个元素连接在一起。对于访问操作，需要从链表头部开始顺序遍历链表，直到找到需要的元素，所以速度比较慢。对于插入或删除操作，只需修改元素中的指针即可完成，速度比较快。

所以，如果需要频繁访问数据，很少插入删除操作，则使用数组；反之，如果频繁插入删除，则应使用链表。

迅雷面试题：最快的排序法的性能，并列举至少三个

最快的排序算法是O(N*lgN)。，快排序，堆排序，归并排序

迅雷面试题：合并用户基本信息和看电影的记录

如何有效合并两个文件：一个是1亿条的用户基本信息，另一个是用户每天看电影连续剧等的记录，5000万条。其中内存只有1G。

显然内存不能同时存下所有的数据，所以考虑分而治之的思想。

假设1K Byte可以保存一个用户的基本信息和看电影记录。我们可以将基本信息和看电影记录都按照hash(user_name)%100的余数各分成100个小文件。利用1G内存，我们可以每次只处理一对小文件，然后将结果输出到一个文件中即可。

在处理一对小文件时，可以利用key为用户名的hash_map将基本信息和看电影记录合并在一起。

迅雷面试题：c语言中不同include方法的差别

#include "filename.h" 首先在程序原文件所在目录下查找，如果找不到，再到系统目录中查找。

#include <filename.h> 直接去系统目录中查找。

在1亿条用户记录里，如何快速查询统计出看了5个电影以上的用户?

构建一个hash map，key为用户名，value为已经看过的电影数量。

遍历所有用户记录，然后根据用户名和已经看过电影数量的情况进行处理:

- 如果用户名不在hash map中，则添加对应用户名，并将值设为1。

- 如果用户名对应的值小于5，则将值加1。如果加1后值为5，则输出此用户名。

- 如果用户名对应的值等于5，则不进行任何操作。

oracle面试题：数据库冷备份和热备份的不同点以及各自的优点

热备份针对归档模式的数据库，在数据库仍旧处于工作状态时进行备份。而冷备份指在数据库关闭后，进行备份，适用于所有模式的数据库。热备份的优点在于当备 份时，数据库仍旧可以被使用并且可以将数据库恢复到任意一个时间点。冷备份的优点在于它的备份和恢复操作相当简单，并且由于冷备份的数据库可以工作在非归 档模式下，数据库性能会比归档模式稍好。（因为不必将archive log写入硬盘）

华为面试题：IP，TCP和UDP协议的定义和主要作用

IP协议是网络层的协议。IP协议规定每个互联网网上的电脑都有一个唯一的IP地址，这样数据包就可以通过路由器的转发到达指定的电脑。但IP协议并不保证数据传输的可靠性。

TCP协议是传输层的协议。它向下屏蔽了IP协议不能可靠传输的缺点，向上提供面向连接的可靠的数据传输。

UDP协议也是传输层的协议。它提供无连接的不可靠传输。

华为面试题：全局变量和局部变量有什么区别

全局变量是整个程序都可访问的变量，生存期从程序开始到程序结束；局部变量存在于模块中(比如某个函数)，只有在模块中才可以访问，生存期从模块开始到模块结束。

全局变量分配在全局数据段，在程序开始运行的时候被加载。局部变量则分配在程序的堆栈中。因此，操作系统和编译器可以通过内存分配的位置来知道来区分全局 变量和局部变量。全局变量和局部变量的区别是在存储器中位置不同，具体说，全局变量存储在数据段中，局部变量一般来说在堆栈段

华为面试题：析构函数和虚函数的用法和作用?

析构函数是在类对象消亡时由系统自动调用。主要用来做对象的清理工作，比如来释放对象申请的动态空间。

基类中用virtual修饰的函数称为虚函数。在派生类中可以对虚函数进行重新定义，这样同样的函数接口可以在不同的派生类中对应不同的实现。当通过基类的指针来调用虚函数时，程序会根据指针实际指向的对象来决定调用哪个实现。

华为面试题：如何引用一个已经定义过的全局变量?

可以用引用头文件的方式，也可以使用extern关键字。

用引用头文件方式，如果将那个变量写错了，那么在编译期间会报错。

用extern方式，如果将变量名写错了，那么在编译期间不会报错，而在连接期间报错。

华为面试题：c语言中局部变量能否和全局变量重名?

局部变量可以与全局变量同名。在函数内引用这个变量时，会用到同名的局部变量，而不会用到全局变量。要用全局变量，需要使用"::"。

对于有些编译器而言，在同一个函数内可以定义多个同名的局部变量，比如在两个循环体内都定义一个同名的局部变量，而那个局部变量的作用域就在那个循环体内。

完美时空面试题：memcpy 和 memmove 有什么区别?

memcpy和memmove都是将源地址的若干个字符拷贝到目标地址。

如果源地址和目标地址有重叠，则memcpy不能保证拷贝正确，但memmove可以保证拷贝正确。

例如：

char src[20];

// set src

char* dst = src + 5;

此时如果要从src拷贝10个字符到dst，则么memcpy不能保证拷贝正确，但是memmove可以保证。

雅虎面试题：HTTP中Get和Post的区别

Get和Post都是浏览器向网页服务器提交数据的方法。

Get把要提交的数据编码在url中，比如 http://hi.baidu.com/mianshiti?key1=value1&key2=value2 中就编码了键值对 key1，value1 和key2，value2。受限于url的长度限制，Get方法能传输的数据有限（不同浏览器对url长度限制不同，比如微软IE设为2048）。

Post把要提交的数据放在请求的body中，而不会显示在url中，因此，也没有数据大小的限制。

由于Get把数据编码在URL中，所以这些变量显示在浏览器的地址栏，也会被记录在服务器端的日志中。所以Post方法更加安全。

```
/*************************************************************************/
/* 从海量数据中找出重复次数最多的一个
   思路:先将海量数据通过哈希表统计出数据的频率并映射为100个小文件，小文件
      中的数据包括两项(数值，出现次数)，然后再对每一个小文件求出重复次数
  最多的一个数据然后将各个小文件出现最多的数据项目通过二路归并进行比
  较，找出频率最大的即为所求
   性能：时间复杂度:O(N)+100*O(N1)+O(nlogn)*/
/*************************************************************************/

#include<iostream>
#include<fstream>
#include<malloc.h>
#include<stdlib.h>
const int ERROR=0;
using namespace std;

struct LinkHash//哈希表
{
 LinkHash *next;
 int m_nValue;
 int count;//数据出现的次数
};
struct _Data//数据结构体
{
 int Value;
 int Count;
};
char *file[101]=
```

```
{"file1.txt","file2.txt","file3.txt","file4.txt","file5.txt","file6.txt","file7.txt","file8.txt","file9.txt","file10.txt",
"file11.txt","file12.txt","file13.txt","file14.txt","file15.txt","file16.txt","file17.txt","file18.txt","file19.txt","file20.
txt",
"file21.txt","file22.txt","file23.txt","file24.txt","file25.txt","file26.txt","file27.txt","file28.txt","file29.txt","file30.
txt",
"file31.txt","file32.txt","file33.txt","file34.txt","file35.txt","file36.txt","file37.txt","file38.txt","file39.txt","file40.
txt",
"file41.txt","file42.txt","file43.txt","file44.txt","file45.txt","file46.txt","file47.txt","file48.txt","file49.txt","file50.
txt",
"file51.txt","file52.txt","file53.txt","file54.txt","file55.txt","file56.txt","file57.txt","file58.txt","file59.txt","file60.
txt",
"file61.txt","file62.txt","file63.txt","file64.txt","file65.txt","file66.txt","file67.txt","file68.txt","file69.txt","file70.
txt",
"file71.txt","file72.txt","file73.txt","file74.txt","file75.txt","file76.txt","file77.txt","file78.txt","file79.txt","file80.
txt",
"file81.txt","file82.txt","file83.txt","file84.txt","file85.txt","file86.txt","file87.txt","file88.txt","file89.txt","file90.
txt",
"file91.txt","file92.txt","file93.txt","file94.txt","file95.txt","file96.txt","file97.txt","file98.txt","file99.txt","file10
0.txt"};
class CHashTable
{
private:
 LinkHash *HashTable[101];//10个空哈希表头
public:
 CHashTable();
 ~CHashTable();

 void HashCollision(int data);
 void WriteToFile();
 _Data GetMaxFreq(char *filename);

};
CHashTable::CHashTable()
{

 int i;
 for(i=0;i<100;i++)//初始化空链表
 {
  HashTable[i]=(LinkHash*)malloc(sizeof(LinkHash));
  if(!HashTable[i])
   exit(ERROR);
  HashTable[i]->count=0;
  HashTable[i]->next=NULL;
  HashTable[i]->m_nValue=-1;
 }
}
CHashTable::~CHashTable()
{

}
int HashFunc(int key)//哈希函数
{
 return key%100;
}

void CHashTable::HashCollision(int data)//链地址法处理冲突
{
 LinkHash *newNode;
 LinkHash *head;
 newNode=(LinkHash*)malloc(sizeof(LinkHash));
 if(!newNode)
```

```
 exit(ERROR);
newNode->next=NULL;
newNode->m_nValue=data;
newNode->count=0;

int p;
bool isRep=false;//重复出现
p=HashFunc(data);
head=HashTable[p];
while(head->next)
{
 head=head->next;
 if(head->m_nValue==data)
 {
  head->count++;//有重复的数据统计出现的次数
  isRep=true;
  break;
 }

}
if(isRep==false)//如果没有重复的数据，则将数据插入
{
 head->next=newNode;
    head=newNode;
    head->count++;
}


}
void CHashTable::WriteToFile()//将结果写入100个小文件中
{
 int i;
 ofstream fout;
 for(i=0;i<100;i++)
 {
  LinkHash *p;
  fout.open(file[i]);
   if(HashTable[i]->next)
   {
    p=HashTable[i]->next;
    while(p)
    {
     fout<<p->m_nValue<<" "<<p->count<<endl;
     p=p->next;
    }
   }
  fout.close();
  fout.clear();
 }
}

_Data CHashTable::GetMaxFreq(char *filename)//遍历文件中数据执行次数T(n)=O(N1)
{//从文件中获取出现频率最多的数据
 fstream fin;
 _Data InData;
 _Data MaxData;
 MaxData.Count=0;
 fin.open(filename);
 if(fin.is_open())
 {
  while(fin>>InData.Value>>InData.Count)
```

```
 {
  if(InData.Count>MaxData.Count)
   MaxData=InData;
 }
}
fin.close();
return MaxData;
}
void BiSearchMax(_Data Array[],int start,int end,_Data &Max)//二路归并数组中数据频率最大的值
{
 _Data Max1;
 Max1.Count=-1;
 int mid;
 if(start==end)
  Max=Array[start];
 else if(end-start+1==2)
 {
  if(Array[start].Count>Array[end].Count)
   Max=Array[start];
  else
   Max=Array[end];
 }
 else
 {
     mid=(start+end)/2;
   BiSearchMax(Array,start,mid,Max);
   BiSearchMax(Array,mid+1,end,Max1);

 }
 if(Max1.Count>Max.Count)
  Max=Max1;
}
int main()
{
 CHashTable HTable=CHashTable();
   fstream fin;
 ofstream fout;
 int i,data,indata;
 _Data FileData[101];
 _Data MaxFreq;
 MaxFreq.Count=0;

 fout.open("input.txt");
 for(i=0;i<1000000;i++)//生成100000个数据
 {
  data=1+rand()%1000;//从1到1000的随机数
  fout<<data<<" ";
 }
 fout.close();

 fin.open("input.txt");
 if(fin.is_open())
 {
  while(fin>>indata)//对海量数据进行遍历执行次数T(n)=N,时间复杂度O(N)
    {
      HTable.HashCollision(indata);
    }
 }

 HTable.WriteToFile();
   for(i=0;i<100;i++)//分别获取100个文件中频率最大的数据，执行次数100*N1(N1为100个文件平均长
```

```
度),时间复杂度100*O(N1)
{
 FileData[i]=HTable.GetMaxFreq(file[i]);
   cout<<FileData[i].Value<<" "<<FileData[i].Count<<endl;
}
BiSearchMax(FileData,0,99,MaxFreq);//对100个数据进行二路归并查找最大值，时间复杂度O(nlogn)
 cout<<"出现最多的是"<<MaxFreq.Value<<" "<<MaxFreq.Count<<endl;

return 1;
```