史上最全的iOS面试题及答案

史上最全的iOS面试题及答案

iOS面试小贴士

————————————————回答好下面的足够了----------

多线程、特别是NSOperation 和 GCD 的内部原理。

运行时机制的原理和运用场景。

SDWebImage的原理。实现机制。如何解决TableView卡的问题。

block和代理的，通知的区别。block的用法需要注意些什么。

strong，weak，retain，assign，copy nomatic 等的区别。

设计模式，mvc，单利，工厂，代理等的应用场景。

单利的写法。在单利中创建数组应该注意些什么。

NSString 的时候用copy和strong的区别。

响应值链。

NSTimer 在子线程中应该手动创建NSRunLoop ， 否则不能循环执行。

UIScrollView和NSTimer组合做循环广告图轮播的时候有一个属性可以控制当上下滚动tableview的时候广告轮播图依然正常滚动。

Xcode最新的自动布局。。。这个很多公司都用。尽量自学下。

git ， 和svn的用法。。。git的几个命令简单的记下。。。

友盟报错可以查到具体某一行的错误，原理是什么。

Instrument 可以检测 电池的耗电量、和内存的消耗。的用法。

动画CABaseAnimation CAKeyAni.... CATrans..... CAGoup.... 等熟悉。。

ARC的原理。

自己写过什么自定义控件就最好了。。

————————————————回答好上面的足够了----------

__block和__weak修饰符的区别其实是挺明显的：

1.__block不管是ARC还是MRC模式下都可以使用，可以修饰对象，还可以修饰基本数据类型。
2.__weak只能在ARC模式下使用，也只能修饰对象（NSString），不能修饰基本数据类型（int）。
3.__block对象可以在block中被重新赋值，__weak不可以。

tableView 滑动卡的问题主要是因为：从缓存中或者是从本地读取图片给UIImage的时候耗费的时间。需要把下面的两句话放到子线程里面：

1. **NSData** *imgData = [NSData dataWithContentsOfURL:[NSURL URLWithString:app.icon]];
2. **UIImage** *image = [UIImage imageWithData:imgData];

把UIImage赋值给图片的时候在主线程。

子线程不能更新UI 所有的UI跟新都是主线程执行了。手指滑动屏幕了。或者屏幕的某个方法执行了。

子线程里面加入NSTimer 的时候需要 手动添加NSRunloop 否则不能循环。

单利里面添加 NSMutableArray 的时候，防止多个地方对它同时便利和修改的话，需要加原子属性。并且用strong，，，并且写一个遍历和修改的方法。加上锁。Lock UnLock

__weak ViewController* weakSelf = self;

GCD里面用 __weak 防止内存释放不了，循环引用。

二、SDWebImage内部实现过程

1. 入口 setImageWithURL:placeholderImage:options: 会先把 placeholderImage 显示，然后 SDWebImageManager 根据 URL 开始处理图片。
2. 进入 SDWebImageManager-downloadWithURL:delegate:options:userInfo:，交给 SDImageCache 从缓存查找图片是否已经下载 queryDiskCacheForKey:delegate:userInfo:.

3. 先从内存图片缓存查找是否有图片，如果内存中已经有图片缓存，SDImageCacheDelegate 回调 imageCache:didFindImage:forKey:userInfo: 到 SDWebImageManager。

4. SDWebImageManagerDelegate 回调 webImageManager:didFinishWithImage: 到 UIImageView+WebCache 等前端展示图片。

5. 如果内存缓存中没有，生成 NSInvocationOperation 添加到队列开始从硬盘查找图片是否已经缓存。

6. 根据 URLKey 在硬盘缓存目录下尝试读取图片文件。这一步是在 NSOperation 进行的操作，所以回主线程进行结果回调 notifyDelegate:。

7. 如果上一操作从硬盘读取到了图片，将图片添加到内存缓存中（如果空闲内存过小，会先清空内存缓存）。SDImageCacheDelegate 回调 imageCache:didFindImage:forKey:userInfo:。进而回调展示图片。

8. 如果从硬盘缓存目录读取不到图片，说明所有缓存都不存在该图片，需要下载图片，回调 imageCache:didNotFindImageForKey:userInfo:。

9. 共享或重新生成一个下载器 SDWebImageDownloader 开始下载图片。

10. 图片下载由 NSURLConnection 来做，实现相关 delegate 来判断图片下载中、下载完成和下载失败。

11. connection:didReceiveData: 中利用 ImageIO 做了按图片下载进度加载效果。

12. connectionDidFinishLoading: 数据下载完成后交给 SDWebImageDecoder 做图片解码处理。

13. 图片解码处理在一个 NSOperationQueue 完成，不会拖慢主线程 UI。如果有需要对下载的图片进行二次处理，最好也在这里完成，效率会好很多。

14. 在主线程 notifyDelegateOnMainThreadWithInfo: 宣告解码完成，imageDecoder:didFinishDecodingImage:userInfo: 回调给 SDWebImageDownloader。

15. imageDownloader:didFinishWithImage: 回调给 SDWebImageManager 告知图片下载完成。

16. 通知所有的 downloadDelegates 下载完成，回调给需要的地方展示图片。

17. 将图片保存到 SDImageCache 中，内存缓存和硬盘缓存同时保存。写文件到硬盘也在以单独 NSInvocationOperation 完成，避免拖慢主线程。

18. SDImageCache 在初始化的时候会注册一些消息通知，在内存警告或退到后台的时候清理内存图片缓存，应用结束的时候清理过期图片。

19. SDWI 也提供了 UIButton+WebCache 和 MKAnnotationView+WebCache，方便使用。

20. SDWebImagePrefetcher 可以预先下载图片，方便后续使用。

从上面流程可以看出，当你调用setImageWithURL:方法的时候，他会自动去给你干这么多事，当你需要在某一具体时刻做事情的时候，你可以覆盖这些方法。比如在下载某个图片的过程中要响应一个事件，就覆盖这个方法：

```
    SDWebImageManager *manager = [SDWebImageManager sharedManager];

    [manager downloadImageWithURL:imagePath2 options:SDWebImageRetryFailed progress:^(NSInteger receivedSize, NSInteger expected

        NSLog(@"显示当前进度");

    } completed:^(UIImage *image, NSError *error, SDImageCacheType cacheType, BOOL finished, NSURL *imageURL) {

        NSLog(@"下载完成");
    }];
```

对于初级来说，用sd_setImageWithURL:的若干个方法就可以实现很好的图片缓存。

UIButton 的父类是UIControl UIControl的父类是UIView UIView的父类是 UIResponder

http状态吗： 302 是请求重定向。500以上是服务器错误。400以上是请求链接错误或者找不到服务器。200以上是正确。100以上是请求接受成功。

HTTP Keep-Alive详解[转]

HTTP是一个请求<->响应模式的典型范例，即客户端向服务器发送一个请求信息，服务器来响应这个信息。在老的HTTP版本中，每个请求都将被创建一个新的客户端->服务器的连接，在这个连接上发送请求，然后接收请求。这样的模式有一个很大的优点就是，它很简单，很容易理解和编程实现；它也有一个很大的缺点就是，它效率很低，因此Keep-Alive被提出用来解决效率低的问题。

Keep-Alive功能使客户端到服务器端的连接持续有效，当出现对服务器的后继请求时，Keep-Alive功能避免了建立或者重新建立连接。市场上 的大部分Web服务器，包括iPlanet、IIS和Apache，都支持HTTP Keep-Alive。对于提供静态内容的网站来说，这个功能通常很有用。但是，对于负担较重的网站来说，这里存在另外一个问题：虽然为客户保留打开的连 接有一定的好处，但它同样影响了性能，因为在处理暂停期间，本来可以释放的资源仍旧被占用。当Web服务器和应用服务器在同一台机器上运行时，Keep- Alive功能对资源利用的影响尤其突出。 此功能为HTTP 1.1预设的功能，HTTP 1.0加上Keep-Aliveheader也可以提供HTTP的持续作用功能。
Keep-Alive: timeout=5, max=100

timeout：过期时间5秒（对应httpd.conf里的参数是：KeepAliveTimeout），max是最多一百次请求，强制断掉连接
就是在timeout时间内又有新的连接过来，同时max会自动减1，直到为0，强制断掉。见下面的四个图，注意看Date的值（前后时间差都是在5秒之内）！

HTTP/1.0

在HTTP/1.0版本中，并没有官方的标准来规定Keep-Alive如何工作，因此实际上它是被附加到HTTP/1.0协议上，如果客户端浏览器支持Keep-Alive，那么就在HTTP请求头中添加一个字段 **Connection: Keep-Alive**，当服务器收到附带有Connection: Keep-Alive的请求时，它也会在响应头中添加一个同样的字段来使用Keep-Alive。这样一来，客户端和服务器之间的HTTP连接就会被保持，不会断开（超过Keep-Alive规定的时间，意外断电等情况除外），当客户端发送另外一个请求时，就使用这条已经建立的连接

HTTP/1.1

在HTTP/1.1版本中，官方规定的Keep-Alive使用标准和在HTTP/1.0版本中有些不同，默认情况下所在HTTP1.1中所有连接都被保持，除非在请求头或响应头中指明要关闭：Connection: Close ，这也就是为什么Connection: Keep-Alive字段再没有意义的原因。另外，还添加了一个新的字段Keep-Alive:，因为这个字段并没有详细描述用来做什么，可忽略它

Not reliable（不可靠）

HTTP是一个无状态协议，这意味着每个请求都是独立的，Keep-Alive没能改变这个结果。另外，Keep-Alive也不能保证客户端和服务器之间的连接一定是活跃的，在HTTP1.1版本中也如此。唯一能保证的就是当连接被关闭时你能得到一个通知，所以不应该让程序依赖于Keep-Alive的保持连接特性，否则会有意想不到的后果

Keep-Alive和POST

在HTTP1.1细则中规定了在一个POST消息体后面不能有任何字符，还指出了对于某一个特定的浏览器可能并不遵循这个标准（比如在POST消息体的后面放置一个CRLF符）。而据我所知，大部分浏览器在POST消息体后都会自动跟一个CRLF符再发送，如何解决这个问题呢？根据上面的说明在POST请求头中禁止使用Keep-Alive，或者由服务器自动忽略这个CRLF，大部分服务器都会自动忽略，但是在未经测试之前是不可能知道一个服务器是否会这样做。

1、常用的方法dispatch_async

为了避免界面在处理耗时的操作时卡死，比如读取网络数据，IO,数据库读写等，我们会在另外一个线程中处理这些操作，然后通知主线程更新界面。

用GCD实现这个流程的操作比前面介绍的NSThread NSOperation的方法都要简单。代码框架结构如下：

如果这样还不清晰的话，那我们还是用上两篇博客中的下载图片为例子，代码如下：

运行显示：

是不是代码比NSThread NSOperation简洁很多，而且GCD会自动根据任务在多核处理器上分配资源，优化程序。

系统给每一个应用程序提供了三个concurrent dispatch queues。这三个并发调度队列是全局的，它们只有优先级的不同。因为是全局的，我们不需要去创建。我们只需要通过使用函数dispath_get_global_queue去得到队列，如下：

这里也用到了系统默认就有一个串行队列main_queue

虽然dispatch queue是引用计数的对象，但是以上两个都是全局的队列，不用retain或release。

2、dispatch_group_async的使用

dispatch_group_async可以实现监听一组任务是否完成，完成后得到通知执行其他的操作。这个方法很有用，比如你执行三个下载任务，当三个任务都下载完成后你才通知界面说完成的了。下面是一段例子代码：

dispatch_group_async是异步的方法，运行后可以看到打印结果：

2012-09-25 16:04:16.737 gcdTest[43328:11303] group1
2012-09-25 16:04:17.738 gcdTest[43328:12a1b] group2
2012-09-25 16:04:18.738 gcdTest[43328:13003] group3
2012-09-25 16:04:18.739 gcdTest[43328:f803] updateUi

每个一秒打印一个，当第三个任务执行后，upadteUi被打印。

3、dispatch_barrier_async的使用

dispatch_barrier_async是在前面的任务执行结束后它才执行，而且它后面的任务等它执行完成之后才会执行

例子代码如下：

打印结果：

2012-09-25 16:20:33.967 gcdTest[45547:11203] dispatch_async1

2012-09-25 16:20:35.967 gcdTest[45547:11303] dispatch_async2

2012-09-25 16:20:35.967 gcdTest[45547:11303] dispatch_barrier_async

2012-09-25 16:20:40.970 gcdTest[45547:11303] dispatch_async3

请注意执行的时间，可以看到执行的顺序如上所述。

4、dispatch_apply

执行某个代码片段N次。

```
dispatch_apply(5, globalQ, ^(size_t index) {
    // 执行5次

});
```

copy与retain：
1、copy其实是建立了一个相同的对象，而retain不是；
2、copy是内容拷贝，retain是指针拷贝；
3、copy是内容的拷贝 ,对于像NSString,的确是这样，但是如果copy的是一个NSArray呢?这时只是copy了指向array中相对应元素的指针.这便是所谓的"浅复制".
4、copy的情况：NSString *newPt = [pt copy];
此时会在堆上重新开辟一段内存存放@"abc" 比如0X1122 内容为@"abc 同时会在栈上为newPt分配空间 比如地址：0Xaacc 内容为0X1122 因此retainCount增加1供newPt来管理0X1122这段内存；
assign与retain：
1、assign: 简单赋值，不更改索引计数；
2、assign的情况：NSString *newPt = [pt assing];

此时newPt和pt完全相同 地址都是0Xaaaa 内容为0X1111 即newPt只是pt的别名，对任何一个操作就等于对另一个操作， 因此retainCount不需要增加；

3、assign就是直接赋值；
4、retain使用了引用计数，retain引起引用计数加1, release引起引用计数减1，当引用计数为0时，dealloc函数被调用，内存被回收；

5、retain的情况：NSString *newPt = [pt retain];

此时newPt的地址不再为0Xaaaa，可能为0Xaabb 但是内容依然为0X1111。 因此newPt 和 pt 都可以管理"abc"所在的内存，因此 retainCount需要增加1 ；
readonly：
1、属性是只读的，默认的标记是读写，如果你指定了只读，在@implementation中只需要一个读取器。或者如果你使用@synthesize关键字，也是有读取器方法被解析
readwrite：
1、说明属性会被当成读写的，这也是默认属性。设置器和读取器都需要在@implementation中实现。如果使用@synthesize关键字，读取器和设置器都会被解析；
nonatomic：
1、非原子性访问，对属性赋值的时候不加锁，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；
weak and strong property (强引用和弱引用的区别)：
1、 weak 和 strong 属性只有在你打开ARC时才会被要求使用，这时你是不能使用retain release autorelease 操作的，因为ARC会自动为你做好这些操作，但是你需要在对象属性上使用weak 和strong,其中strong就相当于retain属性，而weak相当于assign。

2、只有一种情况你需要使用weak（默认是strong），就是为了避免retain cycles（就是父类中含有子类{父类retain了子类}，子类中又调用了父类{子类又retain了父类}，这样都无法release）

3、声明为weak的指针，指针指向的地址一旦被释放，这些指针都将被赋值为nil。这样的好处能有效的防止野指针。

ARC(Automatic Reference Counting):

1、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 以后可以使用该特性。

strong,weak,copy 具体用法：

1.具体一点：IBOutlet可以为weak，NSString为copy，Delegate一般为weak，其他的看情况。一般来说，类“内部”的属性设置为strong，类“外部”的属性设置为weak。说到底就是一个归属权的问题。小心出现循环引用导致内存无法释放。

2.不用ARC的话就会看到很多retian。

3.如果你写了@synthesize abc = _abc；的话，系统自动帮你声明了一个_abc的实例变量。

使用assign: 对基础数据类型 （NSInteger）和C数据类型（int, float, double, char,等）

使用copy： 对NSString

使用retain： 对其他NSObject和其子类

1. 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);
}
```

2static 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。

7objc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 是开发者自已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？

4oid、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。

46**ASIDownloadCache** 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 ];
```

47**HTTP**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：
http://host[":"port][abs_path]
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

50socket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 ---------- |A |A+1 |A+2 |A+3 |int a; | | | | ---------- |A |char b; | | ---------- 如果是小端如何存储c.a的呢？

地址A ----------

---------- |A |A+1 |A+2 |A+3 | int a;

|0x01 |0x00 |0x00 |0x00 | ---------- |A |char b; | | ---------

如果是大端如何存储c.a的呢？

地址A ----------

---------- |A |A+1 |A+2 |A+3 |int a; |0x00 |0x00 |0x00 |0x01 | ---------- |A |char b; | | --------

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

682.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