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
答： 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，一般情况下用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，如果是要在不修改原有类的基础上增加其他原有类没有的方法，就要用类别

虽然OC在语法上禁止类使用多继承，但是在协议的遵守上却允许使用多继承。所以可以用协议来实现多继承。但是协议只能提供接口，而没有提供实现方式，如果只是想多继承基类的接口，那么遵守多协议无疑是最好的方法，而既需要多继承接口，又要多继承其实现，那么协议是无能为力了

2. #import 跟#include 又什么区别，@class呢, #import< 跟 #import""又什么区别
答：#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
答：
- 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：

```
    {lock}
if (property != newValue) {
[property release];
property = [newValue retain];
}
{unlock}
```

nonatomic

禁止多线程，变量保护，提高性能。

- 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。

- 指出访问器不是原子操作，而默认地，访问器是原子操作。这也就是说，在多线程环境下，解析的访问器提供一个对属性的安全访问，从获取器得到的返回值或者通过设置器设置的值可以一次完成，即便是别的线程也正在对其进行访问。如果你不指定 nonatomic ，在自己管理内存的环境中，

解析的访问器保留并自动释放返回的值，如果指定了 nonatomic ，那么访问器只是简单地返回这个值。

– nonatomic非原子操作，决定编译器生成的setter getter是否为原子操作，atomic表示多线程安全（防止在写未完成的时候被另一个线程读取，造成数据错误，但使用同步锁的开销太大会严重影响性能），一般使用nonatomic；

– atomic并不能保证线程安全，若要实现线程安全的操作，还需采用更深层次的锁定机制才可以。例如，一个线程在连续多次读取某属性值的过程中 有别的线程在同时改写此值，那么即使将属性声明为atomic，也还是会读到不同的属性值，iOS程序中一般都会使用nonatomic属性 但在开发Mac OSX程序时 使用atomic属性一般不会有什么性能瓶颈

readwrite可读可写属性 生成getter setter方法

readonly只读属性 不会生成setter方法；不希望属性在类外发生改变

assign是赋值特性，简单赋值，不更改索引计数，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在赋值，传入参数的retaincount+1

copy是赋值特性，setter方法将传入对象复制一份

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
答：

```
– (void) setName:(NSString*) str
{
[str retain];
[name release];
name = str;
}
– (void)setName:(NSString *)str
{
id t = [str copy];
[name release];
name = t;
}
```

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
答： 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
答：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
答：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
答：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。
1). (Garbage Collection)自动内存计数：这种方式和Java类似，在你的程序的执行过程中。始终有一个高人在背后准确地帮你收拾垃圾，你不用考虑它什么时候开始工作，怎样工作。你只需要明白，我申请了一段内存空间，当我不再使用从而这段内存成为垃圾的时候，我就彻底的把它忘记掉，反正那个高人会帮我收拾垃圾。遗憾的是，那个高人需要消耗一定的资源，在携带设备里面，资源是紧俏商品所以iPhone不支持这个功能。所以“Garbage Collection”不是本入门指南的范围，对“Garbage Collection”内部机制感兴趣的同学可以参考一些其他的资料，不过说老实话“Garbage Collection”不大适合适初学者研究。
解决: 通过alloc 一 initial方式创建的, 创建后引用计数+1, 此后每retain一次引用计数+1, 那么在程序中做相应次数的release就好了.
2). (Reference Counted)手动内存计数：就是说，从一段内存被申请之后，就存在一个变量用于保存这段内存被使用的次数，我们暂时把它称为计数器，当计数器变为0的时候，那么就是释放这段内存的时候。比如说，当在程序A里面一段内存被成功申请完成之后，那么这个计数器就从0变成1(我们把这个过程叫做alloc)，然后程序B也需要使用这个内存，那么计数器就从1变成了2(我们把这个过程叫做retain)。紧接着程序A不再需要这段内存了，那么程序A就把这个计数器减1(我们把这个过程叫做release);程序B也不再需要这段内存的时候，那么也把计数器减1(这个过程还是release)。当系统(也就是Foundation)发现这个计数器变 成员了0，那么就会调用内存回收程序把这段内存回收(我们把这个过程叫做dealloc)。顺便提一句，如果没有Foundation，那么维护计数器，释放内存等等工作需要你手工来完成。
解决:一般是由类的静态方法创建的, 函数名中不会出现alloc或init字样, 如[NSString string]和[NSArray arrayWithObject:], 创建后引用计数+0, 在函数出栈后释放, 即相当于一个栈上的局部变量. 当然也可以通过retain延长对象的生存期.
3). (NSAutoRealeasePool)内存池：可以通过创建和释放内存池控制内存申请和回收的时机.
解决:是由autorelease加入系统内存池, 内存池是可以嵌套的, 每个内存池都需要有一个创建释放对, 就像main函数中写的一样. 使用也很简单, 比如[[[NSString alloc]initialWithFormat:@”Hey you!”]autorelease], 即将一个NSString对象加入到最内层的系统内存池, 当我们释放这个内存池时, 其中的对象都会被释放.
9. 原子(atomic)跟非原子(non-atomic)属性有什么区别
答：
1). atomic提供多线程安全。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
2). non-atomic:在自己管理内存的环境中，解析的访问器保留并自动释放返回的值，如果指定了nonatomic ，那么访问器只是简单地返回这个值。
10. 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

```
NSMutableArrayarr= [[NSMutableArray array] retain];+1
NSStringstr= [NSString stringWithFormat:@"test"];
[str retain];+1
[arr addObject:str];+1
NSLog(@"%@%d", str, [str retainCount]);
[str retain]; +1
[str release];-1
```

[str release];-1
NSLog(@"%@%d", str, [str retainCount]);
[arr removeAllObjects]; -1
NSLog(@"%@%d", str, [str retainCount]);
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
11. 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
答：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，

##7️⃣Objective−C内存管理
− arc
− mrc
− NSAutorelease

− 内存管理的几条原则
  − 谁申请谁释放
  − 遵循CocoaTouch的使用原则
  − 内存管理主要要避免过早释放和内存泄漏，对于过早释放是要注意@property 设置特性时，用对特性关键字，对于内存泄漏，有增有减，申请了要负责释放
  − 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
− 内存分析

内存分析的主要目的就是为了检测程序是否存在内存泄露

###静态内存分析(Analyze)

− 作用:

    逻辑错误：访问未初始化的变量, 野指针等;

    声明错误：从未使用过的对象;

    内存管理错误：如内存泄漏等;

− 分析方法:

静态内存分析是不运行程序,直接对代码进行分析.

根据代码的上下文的语法结构,来分析是否有内存泄露

- 缺点:

不一定准确,但是如果发现有提示,那么去结合上下文看一下,这里的代码是否有问题

- 场景演练:

MRC 下桥接 – Foundation 和 CoreFoundation框架的数据类型转换

ARC 下桥接 – Foundation 和 CoreFoundation框架的数据类型转换

###内存分配

- 作用:

查看是内存的分配情况

查看内存是否有释放

- 场景演示:

UIImage 的两种创建方法测试

imageNamed:

imageWithContentOfFile:

###动态内存分析

- 作用:

检测程序在运行过程中是否存在内存泄露

- 场景演示:

模拟循环引用, 测试内存泄露

12.如何对iOS设备进行性能测试
答： Profile– Instruments – Time Profiler
测试iOS版的 App 注意事项分享以下几点：

– app使用过程中，接听电话。可以测试不同的通话时间的长短，对于通话结束后，原先打开的app的响应，比如是否停留在原先界面，继续操作时的相应速度等。

– app使用过程中，有推送消息时，对app的使用影响

– 设备在充电时，app的响应以及操作流畅度

– 设备在不同电量时(低于10%，50%，95%)，app的响应以及操作流畅度

– 意外断电时，app数据丢失情况

– 网络环境变化时，app的应对情况如何：是否有适当提示?从有网络环境到无网络环境时，app的反馈如何?从无网络环境回到有网络环境时，是否能自动加载数据，多久才能开始加载数据

– 多点触摸的情况

– 跟其他app之间互相切换时的响应

– 进程关闭再重新打开的反馈

– iOS系统语言环境变化时

平时用第三方服务的话可以去DevStore（开发者服务商店）找，集成了很多的第三方服务，有服务评测还可以源码下载，是个不错的工具网站。

13. 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
答：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:
14. MVC设计模式是什么 你还熟悉什么设计模式
答：
设计模式：并不是一种新技术，而是一种编码经验，使用比如java中的接口，iphone中的协议，继承关系等基本手段，用比较成熟的逻辑去处理某一种类型的事情，总结为所谓设计模式。面向对象编程中，java已经归纳了23种设计模式。

mvc设计模式 ： 模型，视图，控制器，可以将整个应用程序在思想上分成三大块，对应是的数据的存储或处理，前台的显示，业务逻辑的控制。 Iphone本身的设计思想就是遵循mvc设计模式。其不属于23种设计模式范畴。
代理模式：代理模式给某一个对象提供一个代理对象，并由代理对象控制对源对象的引用.比如一个工厂生产了产品，并不想直接卖给用户，而是搞了很多代理商，用户可以直接找代理商买东西，代理商从工厂进货.常见的如QQ的自动回复就属于代理拦截，代理模式在iphone中得到广泛应用.
单例模式：说白了就是一个类不通过alloc方式创建对象，而是用一个静态方法返回这个类的对象。系统只需要拥有一个的全局对象，这样有利于我们协调系统整体的行为，比如想获得[UIApplication sharedApplication];任何地方调用都可以得到 UIApplication的对象，这个对象是全局唯一的。
观察者模式： 当一个物体发生变化时，会通知所有观察这个物体的观察者让其做出反应。实现起来无非就是把所有观察者的对象给这个物体，当这个物体的发生改变，就会调用遍历所有观察者的对象调用观察者的方法从而达到通知观察者的目的。
工厂模式：

```
public class Factory{
public static Sample creator(int which){
if (which==1)
return new SampleA();
else if (which==2)
return new SampleB();
}
}
```

15 浅复制和深复制的区别
答：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。
16. 类别的作用 继承和类别在实现中有何区别
答：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。

- 分类中也可以声明属性，但这种做法要尽量避免
  - 通过“class-continuation分类”向类中新增实例变量。

– 如果某属性在主接口中声明为“只读”，而类的内部又要用设置方法修改此属性，那么就在“class-continuation分类”中将其扩展为“可读写”。

– 把私有方法的原型声明在“class-continuation分类”里面。

– 若想使用类所遵循的协议不为人所知，则可于“class-continuation分类”中声明。

– “class-continuation分类”和普通的分类不同，它必须定义在其所接续的那个类的实现文件里。其重要之处在于，这是唯一能声明实例变量的分类，而且此分类没有特定的实现文件，其中的方法应该定义在类的主实现文件里。与其他分类不同，“class-continuation分类”没有名字。比如：

```
@interface EOCPerson()
//methods here
@end
```

– 私有方法不能在其他类或子类中使用，如强制使用，Xcode编译器直接报错。
– 如果非要在其他类中调用私有方法，需要给MyClass类添加一个分类(Category)，在分类.h中写上该声
明，分类.m中不用写实现。在要用的类中同时导入类.h和分类.h即可使用。这就是传说中的私有方法的前向引用。
– 对于以下情况1，无法使用类别，须使用继承。
1）新扩展的方法与原方法同名，但是还需要使用父类的实现。因为使用类别，会覆盖原类的实现，无法访问到原来的方法。
2）扩展类的属性，这个类别无法做到。如果不继承的话就用类扩展

17. 类别和类扩展的区别。
答：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。
18. oc中的协议和java中的接口概念有何不同
答：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)”。
19. 什么是KVO和KVC

答：KVC:键 一 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

```
[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。
20. 代理的作用
答：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。
21. oc中可修改和不可以修改类型。
答：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。
22. 我们说的oc是动态运行时语言是什么意思
答：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 ~~~
23. 通知和协议的不同之处
答：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。
24. 什么是推送消息
答：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。 测试push的实现可以查看该博文。
25. 关于多态性
答：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
26. 对于单例的理解
答：在objective-c中要实现一个单例类，至少需要做以下四个步骤：
1).为单例对象实现一个静态实例，并初始化，然后设置成nil，
2).实现一个实例构造方法检查上面声明的静态实例是否为nil，如果是则新建并返回一个本类的实例，

3).重写allocWithZone方法，用来保证其他人直接使用alloc和init试图获得一个新实力的时候不产生一个新实例，
4).适当实现allocWitheZone，copyWithZone，release和autorelease。
27. 说说响应链
答： 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。
可以从责任链模式，来讲通过事件响应链处理，其拥有的扩展性
28. frame和bounds有什么不同
答: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)

29. 方法和选择器有何不同
答：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。
30. OC的垃圾回收机制
答： 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。
31. NSOperation queue
答：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。
32. 什么是延迟加载
答：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。
33. 是否在一个视图控制器中嵌入两个tableview控制器
答：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。
34. 一个tableView是否可以关联两个不同的数据源 你会怎么处理
答：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
35. 什么时候使用NSMutableArray，什么时候使用NSArray
答：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。
36. 给出委托方法的实例，并且说出UITableVIew的Data Source方法

答：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;
37. 在应用中可以创建多少autorelease对象，是否有限制
答案：无
38. 如果我们不创建内存池，是否有内存池提供给我们
答: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
39. 什么时候需要在程序中创建内存池
答：用户自己创建的数据线程，则需要创建该线程的内存池
40. 类NSObject的那些方法经常被使用
答：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
41. 什么是简便构造方法
答：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。
42. 如何使用Xcode设计通用应用
答：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。
43. UIView的动画效果有那些
答：有很多，如 UIViewAnimationOptionCurveEaseInOut UIViewAnimationOptionCurveEaseIn UIViewAnimationOptionCurveEaseOut UIViewAnimationOptionTransitionFlipFromLeft UIViewAnimationOptionTransitionFlipFromRight UIViewAnimationOptionTransitionCurlUpUIViewAnimationOptionTransitionCurlDown
44. 在iPhone应用中如何保存数据
答：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
45. 什么是coredata
答：coredata是苹果提供一套数据保存框架，其基于SQlite
46. 什么是NSManagedObject模型
答：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。
47. 什么是NSManagedobjectContext

答：NSManagedobjectContext对象负责应用和数据库之间的交互。
48. 什么是谓词
答：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。
1
2
predicate = [NSPredicate predicateWithFormat:@"customerID == %d",n];
a = [customers filteredArrayUsingPredicate:predicate];
49. 和coredata一起有哪几种持久化存储机制
答：存入到文件、 存入到NSUserDefaults(系统plist文件中)、存入到Sqlite文件数据库
50. 谈谈对Block 的理解 并写出一个使用Block执行UIVew动画
答：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)

[UIView transitionWithView:self.view
duration:0.2
options:UIViewAnimationOptionTransitionFlipFromLeft
animations:^{ [[blueViewController view] removeFromSuperview]; [[self view]
insertSubview:yellowViewController.view atIndex:0]; }
completion:NULL];
51. 写出上面代码的Block的定义。
答：
1
2
typedef void(^animations)(void);
typedef void(^completion)(BOOL finished);
52. 试着使用+ beginAnimations:context:以及上述Block的定义，写出一个可以完成
1
+ (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
53. 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：ASIHTTPRequest与NSURLConnection
54. 简单介绍下NSURLConnection类及+ sendSynchronousRequest:returningResponse:error:
与－ initWithRequest:delegate:两个方法的区别
答: 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。
55. 多线程是什么
答: 多线程是个复杂的概念，按字面意思是同步完成多项任务，提高了资源的使用效率，从硬件、操作系统、应用软件不同的角度去看，多线程被赋予不同的内涵，对于硬件，现在市面上多数的CPU都是多核的，多核的CPU运算多线程更为出色;从操作系统角度，是多任务，现在用的主流操作系统都是多任务的，可以一边听歌、一边写博客;对于应用来说，多线程可以让应用有更快的回应，可以

在网络下载时，同时响应用户的触摸操作。在iOS应用中，对多线程最初的理解，就是并发，它的含义是原来先做烧水，再摘菜，再炒菜的工作，会变成烧水的同时去摘菜，最后去炒菜。
56. iOS 中的多线程
答: iOS中的多线程，是Cocoa框架下的多线程，通过Cocoa的封装，可以让我们更为方便的使用线程，做过C++的同学可能会对线程有更多的理解，比如线程的创立，信号量、共享变量有认识，Cocoa框架下会方便很多，它对线程做了封装，有些封装，可以让我们创建的对象，本身便拥有线程，也就是线程的对象化抽象，从而减少我们的工程，提供程序的健壮性。
GCD是(Grand Central Dispatch)的缩写 ，从系统级别提供的一个易用地多线程类库，具有运行时的特点，能充分利用多核心硬件。GCD的API接口为C语言的函数，函数参数中多数有Block，关于Block的使用参看这里，为我们提供强大的“接口”，对于GCD的使用参见本文
NSOperation与Queue
NSOperation是一个抽象类，它封装了线程的细节实现，我们可以通过子类化该对象，加上NSQueue来同面向对象的思维，管理多线程程序。具体可参看这里：一个基于NSOperation的多线程网络访问的项目。
NSThread
NSThread是一个控制线程执行的对象，它不如NSOperation抽象，通过它我们可以方便的得到一个线程，并控制它。但NSThread的线程之间的并发控制，是需要我们自己来控制的，可以通过NSCondition实现。
参看 iOS多线程编程之NSThread的使用
其他多线程
在Cocoa的框架下，通知、Timer和异步函数等都有使用多线程，(待补充).
57. 在项目什么时候选择使用GCD，什么时候选择NSOperation
答:
1. GCD是底层的C语言构成的API，而NSOperationQueue及相关对象是Objc的对象。在GCD中，在队列中执行的是由block构成的任务，这是一个轻量级的数据结构；而Operation作为一个对象，为我们提供了更多的选择；
2. 在NSOperationQueue中，我们可以随时取消已经设定要准备执行的任务(当然，已经开始的任务就无法阻止了)，而GCD没法停止已经加入queue的block(其实是有的，但需要许多复杂的代码)；
3. NSOperation能够方便地设置依赖关系，我们可以让一个Operation依赖于另一个Operation，这样的话尽管两个Operation处于同一个并行队列中，但前者会直到后者执行完毕后再执行；
4. 我们能将KVO应用在NSOperation中，可以监听一个Operation是否完成或取消，这样子能比GCD更加有效地掌控我们执行的后台任务；
5. 在NSOperation中，我们能够设置NSOperation的priority优先级，能够使同一个并行队列中的任务区分先后地执行，而在GCD中，我们只能区分不同任务队列的优先级，如果要区分block任务的优先级，也需要大量的复杂代码；
6. 我们能够对NSOperation进行继承，在这之上添加成员变量与成员方法，提高整个代码的复用度，这比简单地将block任务排入执行队列更有自由度，能够在其之上添加更多自定制的功能。
7. GCD 是严格的队列，先进先出 FIFO；NSOperation可以改动 优先级（或者说服务质量）改变执行顺序
NSOperation的高级：最大并发数，控制线程个数，优化了线程的暂停、继续、取消功能（GCD实现起来太难，可以用 KVO ），依赖关系，可以让异步任务同步执行
项目中使用NSOperation的优点是NSOperation是对线程的高度抽象，在项目中使用它，会使项目的程序结构更好，子类化NSOperation的设计思路，是具有面向对象的优点(复用、封装)，使得实现是多线程支持，而接口简单，建议在复杂项目中使用。

项目中使用GCD的优点是GCD本身非常简单、易用，对于不复杂的多线程操作，会节省代码量，而Block参数的使用，会是代码更为易读，建议在简单项目中使用。
58. 什么是block
答: 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。
59. block 实现原理
答: 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)。
使用实例
cocoaTouch框架下动画效果的Block的调用
使用typed声明block
1
2
typedef void(^didFinishBlock)(NSObject *ob);
这就声明了一个didFinishBlock类型的block,
然后便可用
1
@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。
60.关于block
答: 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设计模式一-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
1
dispatch_async(dispatch_get_main_queue(),^(void){finishBlock();});
62.谈谈Object-C的内存管理方式及过程
答: 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.
63.Object-C有私有方法吗 私有变量呢
答: objective-c 一 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
64.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
答: 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.
65.内存管理 Autorelease、retain、copy、assign的set方法和含义
答: 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)
66. C和obj-c 如何混用
答: 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
67. Objective-C堆和栈的区别
答: 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。

68. ViewController的didReceiveMemoryWarning怎么被调用：
答:[supper didReceiveMemoryWarning];
69.什么时候用delegate,什么时候用Notification
答: delegate针对one-to-one关系，用于sender接受到reciever的某个功能反馈值。
notification针对one-to-one/many/none,reciver,用于通知多个object某个事件。
70.用预处理指令#define声明一个常数，用以表明1年中有多少秒（忽略闰年问题）
答:
#define SECONDS_PER_YEAR (60 * 60 * 24 * 365)UL
我在这想看到几件事情：
#define 语法的基本知识（例如：不能以分号结束，括号的使用，等等）
懂得预处理器将为你计算常数表达式的值，因此，直接写出你是如何计算一年中有多少秒而不是计算出实际的值，是更清晰而没有代价的。
意识到这个表达式将使一个16位机的整型数溢出-因此要用到长整型符号L,告诉编译器这个常数是的长整型数。
如果你在你的表达式中用到UL（表示无符号长整型），那么你有了一个好的起点。记住，第一印象很重要。
71.写一个”标准"宏MIN ，这个宏输入两个参数并返回较小的一个。
答：
1
#define MIN(A,B) ((A) <= (B) (A) : (B))
这个测试是为下面的目的而设的：
标识#define在宏中应用的基本知识。这是很重要的，因为直到嵌入(inline)操作符变为标准C的一部分，宏是方便产生嵌入代码的唯一方
法，
对于嵌入式系统来说，为了能达到要求的性能，嵌入代码经常是必须的方法。
三重条件操作符的知识。这个操作符存在C语言中的原因是它使得编译器能产生比 if-then-else 更优化的代码，了解这个用法是很重要的。
懂得在宏中小心地把参数用括号括起来
我也用这个问题开始讨论宏的副作用，例如：当你写下面的代码时会发生什么事
1
least = MIN(*p++, b);
结果是：
1
((*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。
72.关键字const有什么含意 修饰类呢 static的作用,用于类呢 还有extern c的作用
答：
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。
1).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”。
73. 关键字volatile有什么含意　并给出三个不同的例子。
答：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
74. 一个参数既可以是const还可以是volatile吗　一个指针可以是volatile 吗 解释为什么。
答：1).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。
2).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。
75 . static 关键字的作用：
答：
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。
76. 线程与进程的区别和联系
答：
1). 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

2). 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。
3). 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。
4.)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。

5). 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。
77. 列举几种进程的同步机制，并比较其优缺点。
答： 原子操作 信号量机制 自旋锁 管程，会合，分布式系统
78. 进程之间通信的途径
答：共享存储系统消息传递系统管道：以文件系统为基础
79. 进程死锁的原因
答：资源竞争及进程推进顺序非法
80. 死锁的4个必要条件
答：互斥、请求保持、不可剥夺、环路
81. 死锁的处理
答：鸵鸟策略、预防策略、避免策略、检测与解除死锁
82. cocoa touch框架
答：iPhone OS 应用程序的基础 Cocoa Touch 框架重用了许多 Mac 系统的成熟模式，但是它更多地专注于触摸的接口和优化。
UIKit 为您提供了在 iPhone OS 上实现图形，事件驱动程序的基本工具，其建立在和 Mac OS X 中一样的 Foundation 框架上，包括文件处理，网络，字符串操作等。
Cocoa Touch 具有和 iPhone 用户接口一致的特殊设计。有了 UIKit，您可以使用 iPhone OS 上的独特的图形接口控件，按钮，以及全屏视图的功能，您还可以使用加速仪和多点触摸手势来控制您的应用。
各色俱全的框架 除了UIKit 外，Cocoa Touch 包含了创建世界一流 iPhone 应用程序需要的所有框架，从三维图形，到专业音效，甚至提供设备访问 API 以控制摄像头，或通过 GPS 获知当前位置。

Cocoa Touch 既包含只需要几行代码就可以完成全部任务的强大的 Objective-C 框架，也在需要时提供基础的 C 语言 API 来直接访问系统。这些框架包括：
Core Animation：通过 Core Animation，您就可以通过一个基于组合独立图层的简单的编程模型来创建丰富的用户体验。
Core Audio：Core Audio 是播放，处理和录制音频的专业技术，能够轻松为您的应用程序添加强大的音频功能。
Core Data：提供了一个面向对象的数据管理解决方案，它易于使用和理解，甚至可处理任何应用或大或小的数据模型。
功能列表：框架分类
下面是 Cocoa Touch 中一小部分可用的框架：
音频和视频：Core Audio ， OpenAL ， Media Library ， AV Foundation
数据管理 ： Core Data ， SQLite
图形和动画 ： Core Animation ， OpenGL ES ， Quartz 2D
网络：Bonjour ， WebKit ， BSD Sockets
用户应用：Address Book ， Core Location ， Map Kit ， Store Kit
83. 自动释放池是什么,如何工作

答：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.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。
84. Objective-C的优缺点。
答：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
objc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。
85. sprintf,strcpy,memcpy使用上有什么要注意的地方。
答：
1). 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。
2).strcpy是一个字符串拷贝的函数，它的函数原型为strcpy(char *dst, const 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。
3). memcpy是具备字符串拷贝功能的函数，这是一个内存拷贝函数，它的函数原型为memcpy(char *dst, const 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。
86. readwrite，readonly，assign，retain，copy，nonatomic 属性的作用
答：@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。
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。
87. http和scoket通信的区别。
答： http是客户端用http协议进行请求，发送请求时候需要封装http请求头，并绑定请求的数据，服务器一般有web服务器配合（当然也非绝对）。 http请求方式为客户端主动发起请求，服务器才能给响应，一次请求完毕后则断开连接，以节省资源。服务器不能主动给客户端响应（除非采取http长连接 技术）。iphone主要使用类是NSUrlConnection。

scoket是客户端跟服务器直接使用socket“套接字”进行连接，并没有规定连接后断开，所以客户端和服务器可以保持连接通道，双方 都可以主动发送数据。一般在游戏开发或股票开发这种要求即时性很强并且保持发送数据量比较大的场合使用。主要使用类是CFSocketRef。
88. TCP和UDP的区别
答： TCP全称是Transmission Control Protocol，中文名为传输控制协议，它可以提供可靠的、面向连接的网络数据传递服务。传输控制协议主要包含下列任务和功能：
* 确保IP数据报的成功传递。
* 对程序发送的大块数据进行分段和重组。
* 确保正确排序及按顺序传递分段的数据。
* 通过计算校验和，进行传输数据的完整性检查。
TCP提供的是面向连接的、可靠的数据流传输，而UDP提供的是非面向连接的、不可靠的数据流传输。
简单的说，TCP注重数据安全，而UDP数据传输快点，但安全性一般
89. 你了解svn,cvs等版本控制工具么
答： 版本控制 svn,cvs 是两种版控制的器,需要配套相关的svn，cvs服务器。
scm是xcode里配置版本控制的地方。版本控制的原理就是a和b同时开发一个项目，a写完当天的代码之后把代码提交给服务器，b要做的时候先从服务器得到最新版本，就可以接着做。 如果a和b都要提交给服务器，并且同时修改了同一个方法，就会产生代码冲突，如果a先提交，那么b提交时，服务器可以提示冲突的代码，b可以清晰的看到，并做出相应的修改或融合后再提交到服务器。
90. 什么是push。
答： 客户端程序留下后门端口，客户端总是监听针对这个后门的请求，于是 服务器可以主动像这个端口推送消息。
91. 静态链接库
答：此为.a文件，相当于java里的jar包，把一些类编译到一个包中，在不同的工程中如果导入此文件就可以使用里面的类，具体使用依然是#import “ xx.h”。
92. fmmpeg框架
答： 音视频编解码框架，内部使用UDP协议针对流媒体开发，内部开辟了六个端口来接受流媒体数据，完成快速接受之目的。
93. fmdb框架
答：数据库框架，对sqllite的数据操作进行了封装，使用着可把精力都放在sql语句上面。
94. 320框架
答： ui框架，导入320工程作为框架包如同添加一个普通框架一样。cover(open) flower框架 (2d仿射技术)，内部核心类是CATransform3D.
94. 什么是沙盒模型 哪些操作是属于私有api范畴
答：某个iphone工程进行文件操作有此工程对应的指定的位置，不能逾越。
iphone沙箱模型的有四个文件夹documents，tmp，app，Library，永久数据存储一般放documents文件夹，得到模拟器的路径的可使用NSHomeDirectory()方法。Nsuserdefaults保存的文件在tmp文件夹里。
95. 在一个对象的方法里面：self.name=“object”；和 name =”object”有什么不同吗
答：self.name =”object”：会调用对象的setName()方法；
name =“object”：会直接把object赋值给当前对象的name属性。
96. 请简要说明viewDidLoad和viewDidUnload何时调用
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 。
97. 简述内存分区情况
答：
1).代码区：存放函数二进制代码
2).数据区：系统运行时申请内存并初始化，系统退出时由系统释放。存放全局变量、静态变量、常量
3).堆区：通过malloc等函数或new等操作符动态申请得到，需程序员手动申请和释放
4).栈区：函数模块内申请，函数结束时由系统自动释放。存放局部变量、函数参数
98. 队列和栈有什么区别：
答：队列和栈是两种不同的数据容器。从”数据结构”的角度看，它们都是线性结构，即数据元素之间的关系相同。
队列是一种先进先出的数据结构，它在两端进行操作，一端进行入队列操作，一端进行出列队操作。

栈是一种先进后出的数据结构，它只能在栈顶进行操作，入栈和出栈都在栈顶操作。
99. HTTP协议中，POST和GET的区别是什么
答：
1).GET 方法
GET 方法提交数据不安全，数据置于请求行，客户端地址栏可见;
GET 方法提交的数据大小有限
GET 方法不可以设置书签
2).POST 方法
POST 方法提交数据安全，数据置于消息主体内，客户端不可见
POST 方法提交的数据大小没有限制
POST 方法可以设置书签
100. iOS的系统架构
答： iOS的系统架构分为（ 核心操作系统层 theCore OS layer ）、（ 核心服务层theCore Services layer ）、（ 媒体层 theMedia layer ）和（ Cocoa 界面服务层 the Cocoa Touch layer ）四个层次。
101. 控件主要响应3种事件
答：1). 基于触摸的事件； 2). 基于值的事件； 3).基于编辑的事件。
102. xib文件的构成分为哪3个图标 都具有什么功能。
答： File’s Owner 是所有 nib 文件中的每个图标，它表示从磁盘加载 nib 文件的对象；
First Responder 就是用户当前正在与之交互的对象；
View 显示用户界面；完成用户交互；是 UIView 类或其子类。
103. 简述视图控件器的生命周期。
答： loadView 尽管不直接调用该方法，如多手动创建自己的视图，那么应该覆盖这个方法并将它们赋值给试图控制器的 view 属性。
viewDidLoad 只有在视图控制器将其视图载入到内存之后才调用该方法，这是执行任何其他初始化操作的入口。
viewDidUnload 当试图控制器从内存释放自己的方法的时候调用，用于清楚那些可能已经在试图控制器中创建的对象。
viewVillAppear 当试图将要添加到窗口中并且还不可见的时候或者上层视图移出图层后本视图变成顶级视图时调用该方法，用于执行诸如改变视图方向等的操作。实现该方法时确保调用 [super viewWillAppear:

viewDidAppear 当视图添加到窗口中以后或者上层视图移出图层后本视图变成顶级视图时调用，用于放置那些需要在视图显示后执行的代码。确保调用 [super viewDidAppear：]。
104. 动画有基本类型有哪几种；表视图有哪几种基本样式。
答：动画有两种基本类型：隐式动画和显式动画。
105. 实现简单的表格显示需要设置UITableView的什么属性、实现什么协议
答：实现简单的表格显示需要设置 UITableView 的 dataSource 和 delegate 属性，实现 UITableViewDataSource 和 UITableViewDelegate 协议。
106. Cocoa Touch提供了哪几种Core Animation过渡类型
答： Cocoa Touch 提供了 4 种 Core Animation 过渡类型，分别为：交叉淡化、推挤、显示和覆盖。
107. UIView与CLayer有什么区别
答：
1).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，访问它的跟绘图和跟坐标有关的属性。
2).UIView 有个重要属性 layer ，可以返回它的主 CALayer 实例。
3).UIView 的 CALayer 类似 UIView 的子 View 树形结构，也可以向它的 layer 上添加子layer ，来完成某些特殊的表示。即 CALayer 层是可以嵌套的。
4).UIView 的 layer 树形在系统内部，被维护着三份 copy 。分别是逻辑树，这里是代码可以操纵的；动画树，是一个中间层，系统就在这一层上更改属性，进行各种渲染操作；显示树，其内容就是当前正被显示在屏幕上得内容。
5).动画的运作：对 UIView 的 subLayer （非主 Layer ）属性进行更改，系统将自动进行动画生成，动画持续时间的缺省值似乎是 0.5 秒。
6).坐标系统： CALayer 的坐标系统比 UIView 多了一个 anchorPoint 属性，使用CGPoint 结构表示，值域是 0~1 ，是个比例值。这个点是各种图形变换的坐标原点，同时会更改 layer 的 position 的位置，它的缺省值是 {0.5,0.5} ，即在 layer 的中央。
7).渲染：当更新层，改变不能立即显示在屏幕上。当所有的层都准备好时，可以调用 setNeedsDisplay 方法来重绘显示。
8).变换：要在一个层中添加一个 3D 或仿射变换，可以分别设置层的 transform 或affineTransform 属性。
9).变形： Quartz Core 的渲染能力，使二维图像可以被自由操纵，就好像是三维的。图像可以在一个三维坐标系中以任意角度被旋转，缩放和倾斜。 CATransform3D 的一套方法提供了一些魔术般的变换效果。
108. Quatrz 2D的绘图功能的三个核心概念是什么并简述其作用。
答：上下文：主要用于描述图形写入哪里；
路径：是在图层上绘制的内容；
状态：用于保存配置变换的值、填充和轮廓， alpha 值等。
109. iPhone OS主要提供了几种播放音频的方法
答： SystemSound Services
AVAudioPlayer 类
Audio Queue Services
OpenAL
110. 使用AVAudioPlayer类调用哪个框架、使用步骤
答： AVFoundation.framework
步骤：配置 AVAudioPlayer 对象；

实现 AVAudioPlayer 类的委托方法；
控制 AVAudioPlayer 类的对象；
监控音量水平；
回放进度和拖拽播放。
111. 有哪几种手势通知方法、写清楚方法名
答：
–(void)touchesBegan:(NSSet*)touchedwithEvent:(UIEvent*)event;
–(void)touchesMoved:(NSSet*)touched withEvent:(UIEvent*)event;
–(void)touchesEnded:(NSSet*)touchedwithEvent:(UIEvent*)event;
–(void)touchesCanceled:(NSSet*)touchedwithEvent:(UIEvent*)event;
112. CFSocket使用有哪几个步骤。
答：创建 Socket 的上下文；创建 Socket ；配置要访问的服务器信息；封装服务器信息；连接服务器；
113. Core Foundation中提供了哪几种操作Socket的方法
答： CFNetwork 、 CFSocket 和 BSD Socket 。
114. 解析XML文件有哪几种方式
答：以 DOM 方式解析 XML 文件；以 SAX 方式解析 XML 文件；
115. 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
答：iOS 中可以有四种持久化数据的方式：属性列表(plist)、对象归档、 SQLite3 和 Core Data；core data 可以使你以图形界面的方式快速的定义 app 的数据模型，同时在你的代码中容易获取到它。 coredata 提供了基础结构去处理常用的功能，例如保存，恢复，撤销和重做，允许你在 app 中继续创建新的任务。在使用 core data 的时候，你不用安装额外的数据库系统，因为 core data 使用内置的 sqlite 数据库。 core data 将你 app 的模型层放入到一组定义在内存中的数据对象。coredata 会追踪这些对象的改变，同时可以根据需要做相反的改变，例如用户执行撤销命令。当 core data 在对你 app 数据的改变进行保存的时候， core data 会把这些数据归档，并永久性保存。 mac os x 中sqlite 库，它是一个轻量级功能强大的关系数据引擎，也很容易嵌入到应用程序。可以在多个平台使用， sqlite 是一个轻量级的嵌入式 sql 数据库编程。与 core data 框架不同的是， sqlite 是使用程序式的， sql 的主要的 API 来直接操作数据表。 Core Data 不是一个关系型数据库，也不是关系型数据库管理系统 (RDBMS) 。虽然 Core Dta 支持SQLite 作为一种存储类型，但它不能使用任意的 SQLite 数据库。 Core Data 在使用的过程种自己创建这个数据库。 Core Data 支持对一、对多的关系。
116. tableView 的重用机制
答：UITableView 通过重用单元格来达到节省内存的目的: 通过为每个单元格指定一个重用标识符(reuseIdentifier),即指定了单元格的种类,以及当单元格滚出屏幕时,允许恢复单元格以便重用.对于不同种类的单元格使用不同的ID,对于简单的表格,一个标识符就够了.
首先我们需要搞明白为什么要使用重用机制,它的原理是什么.
无论是UITableView还是UICollectionView，都有许多需要显示的cell (item), 但是屏幕的大小是有限的，一次只能显示那么几个，如果我们把所有的数据全部都加载进去,暂时又看不到，就会非常浪费内存.
那么该如何避免这种不必要的内存消耗呢 就是每次只显示屏幕能放得下的cell的数据，在用户滑动屏幕的过程中，再去加载新的数据，于是就有了cell的重用机制
重用机制实现了数据和显示的分离,并不会为每个要显示的数据都创建一个Cell,一般情况下只创建屏幕可显示的最大的cell个数+1,每当有一个cell从屏幕消失，就将其放到缓存池中，如果有新的cell出现，就去缓存池中取，如果缓存池中没有，再创建。

这种机制下系统默认有一个可变数组 NSMutableArray* visiableCells, 用来保存当前显示的cell. 还有一个可变字典 NSMutableDictnery* reusableTableCells, 用来保存可重复利用的cell. 之所以用字典是因为可重用的cell有不止一种样式,我们需要根据它的reuseIdentifier(重用标识符)来查找是否有可重用的该样式的cell.
重用的写法如下:

```
//设置单元格（cell）  indexPath :单元格当前所在位置-哪一组的哪一行
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:
(NSIndexPath *)indexPath
{   //定义重用标识
    static NSString *identifier = @"cell" ;
    //identifier: 因为一个表视图中可能存在多种样式的单元格（cell）,所以要相同样式的单元格放到同一个集合里面,并且为这个                集合绑定标识符,当我们需要用到某种样式的单元格的时候,就根据不同的标识符,从不同的集合中找寻单元格.
  //该方法会先去缓存池中寻找对应的cell 如果缓存池中没有，就看有没有注册对应的cell，如果也没有注册，就看storyboard中有没          有绑定对应标识的cell 都没有的话就创建
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:identifier] ;

    if (!cell) {
  //创建cell的时候需要标示符(Identifier)是因为,当该cell离开屏幕的时候需要根据标示符放到对应的集合中.
    cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleSubtitle
reuseIdentifier:@"cell"];

  return cell ;
}
```

系统第一次执行 cellForRowAtIndexPath:(NSIndexPath *)indexPath的时候, reusableTableCells为空,
[tableView dequeueReusableCellWithIdentifier:identifier] 的返回值为nil，我们需要通过 initWithStyle:UITableViewCellStyleSubtitle reuseIdentifier: identifier] 方式来创建.
当我们的数据过多,整个屏幕的cell显示不完全时,这个方法的执行情况是 :
(1) 先执行initWithStyle:UITableViewCellStyleSubtitle reuseIdentifier: identifier]创建整个屏幕能显示的cell数+1的cell (当我们拖动屏幕的时候,第一个cell没有移出屏幕,最下面的cell就已经存在), 并指定相同或者不同的标示符identifier.把创建出的屏幕能显示的cell全部都加入到visiableCells数组中(最后一个创建的先不加入数组)，reusableTableCells为空.
(2)当我们拖动屏幕时,顶端的cell移出屏幕并加入到reusableTableCells字典中,键为identifier ,并把之前已经创建的但是没有加入到visiableCells的cell加入到visiableCells数组中.
(3)当我们接着拖动的时候,因为reusableTableCells中已经有值，当需要显示新的cell, cellForRowAtIndexPath再次被调用执行[tableView dequeueReusableCellWithIdentifier: identifier]，返回一个标示符为identifier的cell。该cell移出reusableTableCells之后加入到visiableCells；顶端的cell移出visiableCells并加入到reusableTableCells.如果visiableCells数组中没有找到identifier类型的cell,则再次重新alloc一个.

在iOS6之后系统加入了一种单元格注册的方法.
[self.tableView registerClass:[UITableViewCell class] forCellReuseIdentifier: identifier] ;

这个方法的作用是,当我们从重用队列中取cell的时候,如果没有,系统会帮我们创建我们给定类型的cell,如果有,则直接重用. 这种方式cell的样式为系统默认样式.
在设置cell的方法中只需要:
```
– (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    // 重用队列中取单元格 由于上面已经注册过单元格,系统会帮我们做判断,不用再次手动判断单元格是否存在
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier: identifier forIndexPath:indexPath] ;
    return cell ;
}
```
117问:lldb（gdb）常用的调试命令
答:breakpoint 设置断点定位到某一个函数
n 断点指针下一步
po打印对象
118问:如何调试BAD_ACCESS错误
答:1. 重写object的respondsToSelector方法，现实出现EXEC_BAD_ACCESS前访问的最后一个object

2. 通过 Zombie

3. 设置全局断点快速定位问题代码所在行

4. Xcode 7 已经集成了BAD_ACCESS捕获功能：Address Sanitizer。 用法如下：在配置中勾选/Enable Address Sanitizer
119
问:IB中User Defined Runtime Attributes如何使用
答:它能够通过KVC的方式配置一些你在interface builder 中不能配置的属性。当你希望在IB中作尽可能多得事情，这个特性能够帮助你编写更加轻量级的viewcontroller
120问:IBOutlet连出来的视图属性为什么可以被设置成weak
答:因为既然有外链那么视图在xib或者storyboard中肯定存在，视图已经对它有一个强引用了。

不过这个回答漏了个重要知识，使用storyboard（xib不行）创建的vc，会有一个叫_topLevelObjectsToKeepAliveFromStoryboard的私有数组强引用所有top level的对象，所以这时即便outlet声明成weak也没关系
121问: apple用什么方式实现对一个对象的KVO
答:当你观察一个对象时，一个新的类会被动态创建。这个类继承自该对象的原本的类，并重写了被观察属性的 setter 方法。重写的 setter 方法会负责在调用原 setter 方法之前和之后，通知所有观察对象：值的更改。最后通过 isa 混写（isa-swizzling） 把这个对象的 isa 指针（ isa 指针告诉 Runtime 系统这个对象的类是什么 ）指向这个新创建的子类，对象就神奇的变成了新创建的子类的实例。
122问:KVC和KVO的keyPath一定是属性么
答:KVO支持实例变量
123问:KVC的keyPath中的集合运算符如何使用

答:1.必须用在集合对象上或普通对象的集合属性上
2.简单集合运算符有@avg， @count ， @max ， @min ，@sum，
3.格式 @"@sum.age"或 @“集合属性.@max.age"
124问:若一个类有实例变量 NSString *_foo ，调用setValue:forKey:时，可以以foo还是 _foo 作为key
答:都可以。
125问:如何手动触发一个value的KVO
答:所谓的“手动触发”是区别于“自动触发”:

自动触发是指类似这种场景：在注册 KVO 之前设置一个初始值，注册之后，设置一个不一样的值，就可以触发了。

想知道如何手动触发，必须知道自动触发 KVO 的原理:

键值观察通知依赖于 NSObject 的两个方法: willChangeValueForKey: 和 didChangevlueForKey: 。在一个被观察属性发生改变之前， willChangeValueForKey: 一定会被调用，这就 会记录旧的值。而当改变发生后， didChangeValueForKey: 会被调用，继而 observeValueForKey:ofObject:change:context: 也会被调用。如果可以手动实现这些调用，就可以实现“手动触发”了。

那么“手动触发”的使用场景是什么 一般我们只在希望能控制“回调的调用时机”时才会这么做。

具体做法如下:

如果这个 value 是 表示时间的 self.now ，那么代码如下：最后两行代码缺一不可。

```
//  .m文件
//  Created by https://github.com/ChenYilong
//  微博@iOS程序犭袁(http://weibo.com/luohanchenyilong/).
//  手动触发 value 的KVO，最后两行代码缺一不可。

//@property (nonatomic, strong) NSDate *now;
- (void)viewDidLoad
{
    [super viewDidLoad];
    [self willChangeValueForKey:@"now"]; //“手动触发self.now的KVO”，必写。
    [self didChangeValueForKey:@"now"]; //“手动触发self.now的KVO”，必写。
}
```

但是平时我们一般不会这么干，我们都是等系统去“自动触发”。“自动触发”的实现原理:

比如调用 setNow: 时，系统还会以某种方式在中间插入 wilChangeValueForKey: 、didChangeValueForKey: 和 observeValueForKeyPath:ofObject:change:context: 的调用。

大家可能以为这是因为 setNow: 是合成方法，有时候我们也能看到人们这么写代码:

```
- (void)setNow:(NSDate *)aDate {
    [self willChangeValueForKey:@"now"]; // 没有必要
    _now = aDate;
    [self didChangeValueForKey:@"now"];// 没有必要
}
```
这是完全没有必要的代码，不要这么做，这样的话，KVO代码会被调用两次。KVO在调用存取方法之前总是调用 willChangeValueForKey: ，之后总是调用 didChangeValueForkey: 。怎么做到的呢答案是通过 isa 混写（isa-swizzling）。

126问:addObserver:forKeyPath:options:context:各个参数的作用分别是什么，observer中需要实现哪个方法才能获得KVO回调
答:// 添加键值观察
```
/*
1 观察者，负责处理监听事件的对象
2 观察的属性
3 观察的选项
4 上下文
*/
[self.person addObserver:self forKeyPath:@"name" options:NSKeyValueObservingOptionNew | NSKeyValueObservingOptionOld context:@"Person Name"];
```
observer中需要实现一下方法：

```
// 所有的 kvo 监听到事件，都会调用此方法
/*
 1. 观察的属性
 2. 观察的对象
 3. change 属性变化字典（新 / 旧）
 4. 上下文，与监听的时候传递的一致
 */
- (void)observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object change:
   (NSDictionary *)change context:(void *)context;
```

127问:以下代码运行结果如何
答:
```
- (void)viewDidLoad
{
    [super viewDidLoad];
    NSLog(@"1");
    dispatch_sync(dispatch_get_main_queue(), ^{
        NSLog(@"2");
    });
    NSLog(@"3");
}
```
只输出：1 。发生主线程锁死。

128问:苹果为什么要废弃dispatch_get_current_queue
答:dispatch_get_current_queue容易造成死锁

129问:dispatch_barrier_async的作用是什么
答:在并行队列中，为了保持某些任务的顺序，需要等待一些任务完成后才能继续进行，使用 barrier 来等待之前任务完成，避免数据竞争等问题。 dispatch_barrier_async 函数会等待追加到Concurrent Dispatch Queue并行队列中的操作全部执行完之后，然后再执行dispatch_barrier_async 函数追加的处理，等 dispatch_barrier_async 追加的处理执行结束之后，Concurrent Dispatch Queue才恢复之前的动作继续执行。

打个比方：比如你们公司周末跟团旅游，高速休息站上，司机说：大家都去上厕所，速战速决，上完厕所就上高速。超大的公共厕所，大家同时去，程序猿很快就结束了，但程序媛就可能会慢一些，即使你第一个回来，司机也不会出发，司机要等待所有人都回来后，才能出发。
dispatch_barrier_async 函数追加的内容就如同“上完厕所就上高速”这个动作。

130问:如何用GCD同步若干个异步调用 （如根据若干个url异步加载多张图片，然后在都下载完成后合成一张整图）
答:使用Dispatch Group追加block到Global Group Queue,这些block如果全部执行完毕，就会执行Main Dispatch Queue中的结束处理的block。

```
dispatch_queue_t queue =
dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);
dispatch_group_t group = dispatch_group_create();
dispatch_group_async(group, queue, ^{ /*加载图片1 */ });
dispatch_group_async(group, queue, ^{ /*加载图片2 */ });
dispatch_group_async(group, queue, ^{ /*加载图片3 */ });
dispatch_group_notify(group, dispatch_get_main_queue(), ^{
        // 合并图片
});
```

131
问:GCD的队列（dispatch_queue_t）分哪两种类型
答:1.串行队列Serial Dispatch Queue
2.并行队列Concurrent Dispatch Queue

132问:使用系统的某些block api（如UIView的block版本写动画时），是否也考虑引用循环问题
答:系统的某些block api中，UIView的block版本写动画时不需要考虑，但也有一些api 需要考虑：

所谓“引用循环”是指双向的强引用，所以那些“单向的强引用”（block 强引用 self ）没有问题，比如这些：

```
[UIView animateWithDuration:duration animations:^{ [self.superview layoutIfNeeded]; }];
[[NSOperationQueue mainQueue] addOperationWithBlock:^{ self.someProperty = xyz; }];
```

```
[[NSNotificationCenter defaultCenter] addObserverForName:@"someNotification"
                                                  object:nil
                                                   queue:[NSOperationQueue mainQueue]
usingBlock:^(NSNotification * notification) {
                                                    self.someProperty = xyz; }];
```

这些情况不需要考虑“引用循环”。

但如果你使用一些参数中可能含有 ivar 的系统 api ，如 GCD 、NSNotificationCenter就要小心一点：比如GCD 内部如果引用了 self，而且 GCD 的其他参数是 ivar，则要考虑到循环引用：

```
__weak __typeof__(self) weakSelf = self;
dispatch_group_async(_operationsGroup, _operationsQueue, ^
{
__typeof__(self) strongSelf = weakSelf;
[strongSelf doSomething];
[strongSelf doSomethingElse];
} );
```

类似的：

```
__weak __typeof__(self) weakSelf = self;
  _observer = [[NSNotificationCenter defaultCenter] addObserverForName:@"testKey"
                                                                 object:nil
                                                                  queue:nil
                                                             usingBlock:^(NSNotification *note) {
      __typeof__(self) strongSelf = weakSelf;
      [strongSelf dismissModalViewControllerAnimated:YES];
  }];
```

self -- _observer -- block -- self 显然这也是一个循环引用。

133问:在block内如何修改block外部变量
答:默认情况下，在block中访问的外部变量是复制过去的，即：写操作不对原变量生效。但是你可以加上__block来让其写操作生效，示例代码如下:

```
block int a = 0;
void  (^foo)(void) = ^{
    a = 1;
}
f00();
//这里，a的值被修改为1
```

134问:使用block时什么情况会发生引用循环，如何解决
答:一个对象中强引用了block，在block中又使用了该对象，就会发射循环引用。 解决方法是将该对象使用__weak或者__block修饰符修饰之后再在block中使用。

id weak weakSelf = self; 或者 weak __typeof(&*self)weakSelf = self该方法可以设置宏
id __block weakSelf = self;
135问:苹果是如何实现autoreleasepool的
答:autoreleasepool以一个队列数组的形式实现,主要通过下列三个函数完成.

objc_autoreleasepoolPush
objc_autoreleasepoolPop
objc_aurorelease
看函数名就可以知道，对autorelease分别执行push，和pop操作。销毁对象时执行release操作。

136问:BAD_ACCESS在什么情况下出现
答:访问了野指针，比如对一个已经释放的对象执行了release、访问已经释放对象的成员变量或者发消息。 死循环

137问:不手动指定autoreleasepool的前提下，一个autorealese对象在什么时刻释放 （比如在一个vc的viewDidLoad中创建）
答:分两种情况：手动干预释放时机、系统自动去释放。

手动干预释放时机--指定autoreleasepool 就是所谓的：当前作用域大括号结束时释放。
系统自动去释放--不手动指定autoreleasepool
Autorelease对象会在当前的 runloop 迭代结束时释放。
如果在一个vc的viewDidLoad中创建一个 Autorelease对象，那么该对象会在 viewDidAppear 方法执行前就被销毁了。

138
问:ARC通过什么方式帮助开发者管理内存
答:编译时根据代码上下文，插入 retain/release

139问:objc使用什么机制管理对象内存
答:通过 retainCount 的机制来决定对象是否需要释放。 每次 runloop 的时候，都会检查对象的 retainCount，如果retainCount 为 0，说明该对象没有地方需要继续使用了，可以释放掉了。

140问:猜想runloop内部是如何实现的
答:一般来讲，一个线程一次只能执行一个任务，执行完成后线程就会退出。如果我们需要一个机制，让线程能随时处理事件但并不退出，通常的代码逻辑 是这样的：

```
function loop() {
    initialize();
    do {
        var message = get_next_message();
        process_message(message);
    } while (message != quit);
}
```

或使用伪代码来展示下:

```
//
// http://weibo.com/luohanchenyilong/ (微博@iOS程序犭袁)
// https://github.com/ChenYilong
int main(int argc, char * argv[]) {
 //程序一直运行状态
 while (AppIsRunning) {
     //睡眠状态，等待唤醒事件
     id whoWakesMe = SleepForWakingUp();
     //得到唤醒事件
     id event = GetEvent(whoWakesMe);
     //开始处理事件
     HandleEvent(event);
 }
 return 0;
}
```

141问:以+ scheduledTimerWithTimeInterval...的方式触发的timer，在滑动页面上的列表时，timer会暂定回调，为什么 如何解决

答:RunLoop只能运行在一种mode下，如果要换mode，当前的loop也需要停下重启成新的。利用这个机制，ScrollView滚动过程中NSDefaultRunLoopMode（kCFRunLoopDefaultMode）的mode会切换到UITrackingRunLoopMode来保证ScrollView的流畅滑动：只能在NSDefaultRunLoopMode模式下处理的事件会影响scrllView的滑动。

如果我们把一个NSTimer对象以NSDefaultRunLoopMode（kCFRunLoopDefaultMode）添加到主运行循环中的时候, ScrollView滚动过程中会因为mode的切换，而导致NSTimer将不再被调度。

同时因为mode还是可定制的，所以：

Timer计时会被scrollView的滑动影响的问题可以通过将timer添加到NSRunLoopCommonModes（kCFRunLoopCommonModes）来解决。代码如下：

```
//
// http://weibo.com/luohanchenyilong/ (微博@iOS程序犭袁)
// https://github.com/ChenYilong

//将timer添加到NSDefaultRunLoopMode中
[NSTimer scheduledTimerWithTimeInterval:1.0
    target:self
    selector:@selector(timerTick:)
    userInfo:nil
    repeats:YES];
//然后再添加到NSRunLoopCommonModes里
NSTimer *timer = [NSTimer timerWithTimeInterval:1.0
    target:self
```

```
    selector:@selector(timerTick:)
    userInfo:nil
    repeats:YES];
[[NSRunLoop currentRunLoop] addTimer:timer forMode:NSRunLoopCommonModes];
```

142问:runloop的mode作用是什么
答:model 主要是用来指定事件在运行循环中的优先级的，分为：

NSDefaultRunLoopMode（kCFRunLoopDefaultMode）：默认，空闲状态
UITrackingRunLoopMode：ScrollView滑动时
UIInitializationRunLoopMode：启动时
NSRunLoopCommonModes（kCFRunLoopCommonModes）：Mode集合
苹果公开提供的 Mode 有两个：

NSDefaultRunLoopMode（kCFRunLoopDefaultMode）
NSRunLoopCommonModes（kCFRunLoopCommonModes）

143问:runloop和线程有什么关系
答:总的说来，Run loop，正如其名，loop表示某种循环，和run放在一起就表示一直在运行着的循环。实际上，run loop和线程是紧密相连的，可以这样说run loop是为了线程而生，没有线程，它就没有存在的必要。Run loops是线程的基础架构部分， Cocoa 和 CoreFundation 都提供了 run loop 对象方便配置和管理线程的 run loop （以下都以 Cocoa 为例）。每个线程，包括程序的主线程（ main thread ）都有与之相应的 run loop 对象。

runloop 和线程的关系：

1. 主线程的run loop默认是启动的。

iOS的应用程序里面，程序启动后会有一个如下的main()函数

```
int main(int argc, char * argv[]) {
@autoreleasepool {    return UIApplicationMain(argc, argv, nil,
NSStringFromClass([AppDelegate class]));
}
}
```

重点是UIApplicationMain()函数，这个方法会为main thread设置一个NSRunLoop对象，这就解释了：为什么我们的应用可以在无人操作的时候休息，需要让它干活的时候又能立马响应。

2. 对其它线程来说，run loop默认是没有启动的，如果你需要更多的线程交互则可以手动配置和启动，如果线程只是去执行一个长时间的已确定的任务则不需要。

3. 在任何一个 Cocoa 程序的线程中，都可以通过以下代码来获取到当前线程的 run loop 。

NSRunLoop *runloop = [NSRunLoop currentRunLoop];

144问:能否向编译后得到的类中增加实例变量 能否向运行时创建的类中添加实例变量 为什么
答:不能向编译后得到的类中增加实例变量；
能向运行时创建的类中添加实例变量；
解释下：

因为编译后的类已经注册在 runtime 中，类结构体中的 objc_ivar_list 实例变量的链表 和 instance_size 实例变量的内存大小已经确定，同时runtime 会调用 class_setIvarLayout 或 class_setWeakIvarLayout 来处理 strong weak 引用。所以不能向存在的类中添加实例变量；
运行时创建的类是可以添加实例变量，调用 class_addIvar 函数。但是得在调用 objc_allocateClassPair 之后，objc_registerClassPair 之前，原因同上。
145问:runtime如何实现weak变量的自动置nil
答:runtime 对注册的类， 会进行布局，对于 weak 对象会放入一个 hash 表中。 用 weak 指向的对象内存地址作为 key，当此对象的引用计数为0的时候会 dealloc，假如 weak 指向的对象内存地址是a，那么就会以a为键， 在这个 weak 表中搜索，找到所有以a为键的 weak 对象，从而设置为 nil。

146问:_objc_msgForward函数是做什么的，直接调用它将会发生什么
答:_objc_msgForward是 IMP 类型，用于消息转发的：当向一个对象发送一条消息，但它并没有实现的时候，_objc_msgForward会尝试做消息转发
147问:objc中的类方法和实例方法有什么本质区别和联系
答:类方法：
类方法是属于类对象的
类方法只能通过类对象调用
类方法中的self是类对象
类方法可以调用其他的类方法
类方法中不能访问成员变量
类方法中不定直接调用对象方法

实例方法：
实例方法是属于实例对象的
实例方法只能通过实例对象调用
实例方法中的self是实例对象
实例方法中可以访问成员变量
实例方法中直接调用实例方法
实例方法中也可以调用类方法(通过类名)

148问:使用runtime Associate方法关联的对象，需要在主对象dealloc的时候释放么
答:无论在MRC下还是ARC下均不需要
对象的内存销毁时间表，分四个步骤：

// 对象的内存销毁时间表

// http://weibo.com/luohanchenyilong/ (微博@iOS程序犭袁)
// https://github.com/ChenYilong
// 根据 WWDC 2011, Session 322 (36分22秒)中发布的内存销毁时间表

1. 调用 -release ： 引用计数变为零
   * 对象正在被销毁，生命周期即将结束.
   * 不能再有新的 __weak 弱引用， 否则将指向 nil.
   * 调用 [self dealloc]
2. 父类 调用 -dealloc
   * 继承关系中最底层的父类 在调用 -dealloc
   * 如果是 MRC 代码 则会手动释放实例变量们（iVars）
   * 继承关系中每一层的父类 都在调用 -dealloc
3. NSObject 调 -dealloc
   * 只做一件事：调用 Objective-C runtime 中的 object_dispose() 方法
4. 调用 object_dispose()
   * 为 C++ 的实例变量们（iVars）调用 destructors
   * 为 ARC 状态下的 实例变量们（iVars） 调用 -release
   * 解除所有使用 runtime Associate方法关联的对象
   * 解除所有 __weak 引用
   * 调用 free()

149问: runtime如何通过selector找到对应的IMP地址 （分别考虑类方法和实例方法）
答:每一个类对象中都一个方法列表,方法列表中记录着方法的名称,方法实现,以及参数类型,其实selector本质就是方法名称,通过这个方法名称就可以在方法列表中找到对应的方法实现.
150问:一个objc对象的isa的指针指向什么 有什么作用
答:指向他的类对象,从而可以找到对象上的方法
151问:一个objc对象如何进行内存布局 （考虑有父类的情况）
答:所有父类的成员变量和自己的成员变量都会存放在该对象所对应的存储空间中.
每一个对象内部都有一个isa指针,指向他的类对象,类对象中存放着本对象的
1）对象方法列表（对象能够接收的消息列表，保存在它所对应的类对象中）

2）成员变量的列表

3）属性列表

它内部也有一个isa指针指向元对象(meta class),元对象内部存放的是类方法列表,类对象内部还有一个superclass的指针,指向他的父类对象。

1）根对象就是NSobject，它的superclass指针指向nil。

2）类对象既然称为对象，那它也是一个实例。类对象中也有一个isa指针指向它的元类(meta class)，即类对象是元类的实例。元类内部存放的是类方法列表，根元类的isa指针指向自己，superclass指针指向NSObject类。

152问:什么时候会报unrecognized selector的异常

答:简单来说：当该对象上某个方法,而该对象上没有实现这个方法的时候， 可以通过“消息转发”进行解决。

简单的流程如下，在上一题中也提到过：objc是动态语言，每个方法在运行时会被动态转为消息发送，即：objc_msgSend(receiver, selector)。

objc在向一个对象发送消息时，runtime库会根据对象的isa指针找到该对象实际所属的类，然后在该类中的方法列表以及其父类方法列表中寻找方法运行，如果，在最顶层的父类中依然找不到相应的方法时，程序在运行时会挂掉并抛出异常unrecognized selector sent to XXX 。但是在这之前，objc的运行时会给出三次拯救程序崩溃的机会：

Method resolution
objc运行时会调用+resolveInstanceMethod:或者 +resolveClassMethod:，让你有机会提供一个函数实现。如果你添加了函数并返回 YES，那运行时系统就会重新启动一次消息发送的过程，如果 resolve 方法返回 NO ，运行时就会移到下一步，消息转发（Message Forwarding）。

Fast forwarding
如果目标对象实现了–forwardingTargetForSelector:，Runtime 这时就会调用这个方法，给你把这个消息转发给其他对象的机会。 只要这个方法返回的不是nil和self，整个消息发送的过程就会被重启，当然发送的对象会变成你返回的那个对象。否则，就会继续Normal Fowarding。 这里叫Fast，只是为了区别下一步的转发机制。因为这一步不会创建任何新的对象，但下一步转发会创建一个NSInvocation对象，所以相对更快点。

Normal forwarding
这一步是Runtime最后一次给你挽救的机会。首先它会发送–methodSignatureForSelector:消息获得函数的参数和返回值类型。如果–methodSignatureForSelector:返回nil，Runtime则会发出–doesNotRecognizeSelector:消息，程序这时也就挂掉了。如果返回了一个函数签名，Runtime就会创建一个NSInvocation对象并发送–forwardInvocation:消息给目标对象。

153问:objc中向一个对象发送消息[obj foo]和objc_msgSend()函数之间有什么关系
答:具体原因同上题：该方法编译之后就是objc_msgSend()函数调用.如果我没有记错的大概是这样的：

((void ()(id, SEL))(void)objc_msgSend)((id)obj, sel_registerName("foo"));
也就是说：

[obj foo];在objc动态编译时，会被转意为：objc_msgSend(obj, @selector(foo));。

154问:objc中向一个nil对象发送消息将会发生什么
答:在Objective-C中向nil发送消息是完全有效的——只是在运行时不会有任何作用:

如果一个方法返回值是一个对象，那么发送给nil的消息将返回0(nil)。例如：

Person * motherInlaw = [[aPerson spouse] mother];
如果spouse对象为nil，那么发送给nil的消息mother也将返回nil。

1）如果方法返回值为指针类型，其指针大小为小于或者等于sizeof(void*)，float，double，long double 或者long long的整型标量，发送给nil的消息将返回0。

2）如果方法返回值为结构体,发送给nil的消息将返回0。结构体中各个字段的值将都是0。

3）如果方法的返回值不是上述提到的几种情况，那么发送给nil的消息的返回值将是未定义的。

具体原因如下：

objc是动态语言，每个方法在运行时会被动态转为消息发送，即：objc_msgSend(receiver, selector)。

那么，为了方便理解这个内容，还是贴一个objc的源代码：

```
struct objc_class {
  Class isa OBJC_ISA_AVAILABILITY; //isa指针指向Meta Class，因为Objc的类的本身也是一个Object，为了处理这个关系，runtime就创造了Meta Class，当给类发送[NSObject alloc]这样消息时，实际上是把这个消息发给了Class Object
  #if !__OBJC2__
  Class super_class OBJC2_UNAVAILABLE; // 父类
  const char *name OBJC2_UNAVAILABLE; // 类名
  long version OBJC2_UNAVAILABLE; // 类的版本信息，默认为0
  long info OBJC2_UNAVAILABLE; // 类信息，供运行期使用的一些位标识
  long instance_size OBJC2_UNAVAILABLE; // 该类的实例变量大小
  struct objc_ivar_list *ivars OBJC2_UNAVAILABLE; // 该类的成员变量链表
  struct objc_method_list **methodLists OBJC2_UNAVAILABLE; // 方法定义的链表
  struct objc_cache *cache OBJC2_UNAVAILABLE; // 方法缓存，对象接到一个消息会根据isa指针查找消息对象，这时会在method Lists中遍历，如果cache了，常用的方法调用时就能够提高调用的效率。
  struct objc_protocol_list *protocols OBJC2_UNAVAILABLE; // 协议链表
  #endif
  } OBJC2_UNAVAILABLE;
```

objc在向一个对象发送消息时，runtime库会根据对象的isa指针找到该对象实际所属的类，然后在该类中的方法列表以及其父类方法列表中寻找方法运行，然后在发送消息的时候，objc_msgSend方法不会返回值，所谓的返回内容都是具体调用时执行的。 那么，回到本题，如果向一个nil对象发送消息，首先在寻找对象的isa指针时就是0地址返回了，所以不会出现任何错误。

155问:在有了自动合成属性实例变量之后，@synthesize还有哪些使用场景

答:回答这个问题前，我们要搞清楚一个问题，什么情况下不会autosynthesis（自动合成）

同时重写了setter和getter时
重写了只读属性的getter时
使用了@dynamic时
在 @protocol 中定义的所有属性

在 category 中定义的所有属性
重载的属性
当你在子类中重载了父类中的属性，你必须 使用@synthesize来手动合成ivar。

除了后三条，对其他几个我们可以总结出一个规律：当你想手动管理@property的所有内容时，你就会尝试通过实现@property的所有“存取方法”（the accessor methods）或者使用@dynamic来达到这个目的，这时编译器就会认为你打算手动管理@property，于是编译器就禁用了autosynthesis（自动合成）。

因为有了autosynthesis（自动合成），大部分开发者已经习惯不去手动定义ivar，而是依赖于autosynthesis（自动合成），但是一旦你需要使用ivar，而autosynthesis（自动合成）又失效了，如果不去手动定义ivar，那么你就得借助@synthesize来手动合成ivar。
156问:@synthesize合成实例变量的规则是什么 假如property名为foo，存在一个名为_foo的实例变量，那么还会自动合成新变量么
答:如果使用了属性的话，那么编译器就会自动编写访问属性所需的方法，此过程叫做“自动合成”(auto synthesis)。需要强调的是，这个过程由编译器在编译期执行，所以编辑器里看不到这些“合成方法”(synthesized method)的源代码。除了生成方法代码之外，编译器还要自动向类中添加适当类型的实例变量，并且在属性名前面加下划线，以此作为实例变量的名字。

@interface CYLPerson : NSObject
@property NSString *firstName;
@property NSString *lastName;
@end
在上例中，会生成两个实例变量，其名称分别为 _firstName与_lastName。也可以在类的实现代码里通过@synthesize语法来指定实例变量的名字:

@implementation CYLPerson
@synthesize firstName = _myFirstName;
@synthesize lastName = _myLastName;
@end
上述语法会将生成的实例变量命名为_myFirstName与_myLastName，而不再使用默认的名字。一般情况下无须修改默认的实例变量名，但是如果你不喜欢以下划线来命名实例变量，那么可以用这个办法将其改为自己想要的名字。笔者还是推荐使用默认的命名方案，因为如果所有人都坚持这套方案，那么写出来的代码大家都能看得懂。

总结下@synthesize合成实例变量的规则，有以下几点：

1）如果指定了成员变量的名称,会生成一个指定的名称的成员变量,

2）如果这个成员已经存在了就不再生成了.

3）如果是 @synthesize foo; 还会生成一个名称为foo的成员变量，也就是说：如果没有指定成员变量的名称会自动生成一个属性同名的成员变量。

4）如果是 @synthesize foo = _foo; 就不会生成成员变量了.

假如property名为foo，存在一个名为_foo的实例变量，那么还会自动合成新变量么 不会。

157问:用@property声明的NSString（或NSArray，NSDictionary）经常使用copy关键字，为什么如果改用strong关键字，可能造成什么问题
答:1）因为父类指针可以指向子类对象,使用copy的目的是为了让本对象的属性不受外界影响,使用copy无论给我传入是一个可变对象还是不可对象,我本身持有的就是一个不可变的副本.

2）如果我们使用是strong,那么这个属性就有可能指向一个可变对象,如果这个可变对象在外部被修改了,那么会影响该属性.

copy此特质所表达的所属关系与strong类似。然而设置方法并不保留新值，而是将其“拷贝”(copy)。 当属性类型为NSString时，经常用此特质来保护其封装性，因为传递给设置方法的新值有可能指向一个NSMutableString类的实例。这个类是NSString的子类，表示一种可修改其值的字符串，此时若是不拷贝字符串，那么设置完属性之后，字符串的值就可能会在对象不知情的情况下遭人更改。所以，这时就要拷贝一份“不可变”(immutable)的字符串，确保对象中的字符串值不会无意间变动。只要实现属性所用的对象是“可变的”(mutable)，就应该在设置新属性值时拷贝一份。

为了理解这种做法，首先要知道，对非集合类对象的copy操作：

在非集合类对象中：对immutable对象进行copy操作，是指针复制，mutableCopy操作时内容复制；对mutable对象进行copy和mutableCopy都是内容复制。用代码简单表示如下：

[immutableObject copy] // 浅复制
[immutableObject mutableCopy] //深复制
[mutableObject copy] //深复制
[mutableObject mutableCopy] //深复制
比如以下代码：

NSMutableString *string = [NSMutableString stringWithString:@"origin"];//copy
NSString *stringCopy = [string copy];
查看内存，会发现 string、stringCopy 内存地址都不一样，说明此时都是做内容拷贝、深拷贝。即使你进行如下操作：

1
[string appendString:@"origion!"]
stringCopy的值也不会因此改变，但是如果不使用copy，stringCopy的值就会被改变。 集合类对象以此类推。 所以，
用@property声明 NSString、NSArray、NSDictionary 经常使用copy关键字，是因为他们有对应的可变类型：NSMutableString、NSMutableArray、NSMutableDictionary，他们之间可能进行赋值操作，为确保对象中的字符串值不会无意间变动，应该在设置新属性值时拷贝一份。

158问:ARC下，不显式指定任何属性关键字时，默认的关键字都有哪些
答:对应基本数据类型默认关键字是

atomic,readwrite,assign

对于普通的OC对象
atomic,readwrite,strong
159问:@synthesize和@dynamic分别有什么作用
答:1）@property有两个对应的词，一个是@synthesize，一个是@dynamic。如果@synthesize和@dynamic都没写，那么默认的就是@syntheszie var = _var;

2）@synthesize的语义是如果你没有手动实现setter方法和getter方法，那么编译器会自动为你加上这两个方法。

3）@dynamic告诉编译器：属性的setter与getter方法由用户自己实现，不自动生成。（当然对于readonly的属性只需提供getter即可）。假如一个属性被声明为@dynamic var，然后你没有提供@setter方法和@getter方法，编译的时候没问题，但是当程序运行到instance.var = someVar，由于缺setter方法会导致程序崩溃；或者当运行到 someVar = var时，由于缺getter方法同样会导致崩溃。编译时没问题，运行时才执行相应的方法，这就是所谓的动态绑定。
160问:weak属性需要在dealloc中置nil么
答:不需要。

在ARC环境无论是强指针还是弱指针都无需在deallco设置为nil，ARC会自动帮我们处理。

即便是编译器不帮我们做这些，weak也不需要在dealloc中置nil：

正如上文的：runtime 如何实现 weak 属性 中提到的：

我们模拟下weak的setter方法，应该如下：

```
- (void)setObject:(NSObject *)object
{
    objc_setAssociatedObject(self, "object", object, OBJC_ASSOCIATION_ASSIGN);
    [object cyl_runAtDealloc:^{
        _object = nil;
    }];
}
```

也即:在属性所指的对象遭到摧毁时，属性值也会清空(nil out)。
161问:@property中有哪些属性关键字 / @property 后面可以有哪些修饰符
答:属性可以拥有的特质分为四类:

原子性---nonatomic特质
在默认情况下，由编译器合成的方法会通过锁定机制确保其原子性(atomicity)。如果属性具备nonatomic特质，则不使用同步锁。请注意，尽管没有名为“atomic”的特质(如果某属性不具备nonatomic特质，那它就是“原子的”(atomic))，但是仍然可以在属性特质中写明这一点，编译器不会报错。若是自己定义存取方法，那么就应该遵从与属性特质相符的原子性。

读/写权限---readwrite(读写)、readooly (只读)
内存管理语义---assign、strong、 weak、unsafe_unretained、copy

方法名---getter=、setter=
getter=的样式：

1
@property (nonatomic, getter=isOn) BOOL on;
（ setter=这种不常用，也不推荐使用。故不在这里给出写法。）

不常用的：nonnull,null_resettable,nullable
162问:runtime 如何实现 weak 属性
答:要实现weak属性，首先要搞清楚weak属性的特点：

weak 此特质表明该属性定义了一种“非拥有关系”(nonowning relationship)。为这种属性设置新值时，设置方法既不保留新值，也不释放旧值。此特质同assign类似， 然而在属性所指的对象遭到摧毁时，属性值也会清空(nil out)。

那么runtime如何实现weak变量的自动置nil

runtime 对注册的类， 会进行布局，对于 weak 对象会放入一个 hash 表中。 用 weak 指向的对象内存地址作为 key，当此对象的引用计数为0的时候会 dealloc，假如 weak 指向的对象内存地址是a，那么就会以a为键， 在这个 weak 表中搜索，找到所有以a为键的 weak 对象，从而设置为 nil。
163问: @protocol 和 category 中如何使用 @property
答:1）在protocol中使用property只会生成setter和getter方法声明,我们使用属性的目的,是希望遵守我协议的对象能实现该属性

2）category 使用 @property 也是只会生成setter和getter方法的声明,如果我们真的需要给category增加属性的实现,需要借助于运行时的两个函数：

①objc_setAssociatedObject

②objc_getAssociatedObject
164问:@property 的本质是什么 ivar、getter、setter 是如何生成并添加到这个类中的
答:@property 的本质是什么

@property = ivar + getter + setter;

下面解释下：

“属性”(property)有两大概念：ivar（实例变量）、存取方法（access method = getter + setter）。

“属性”(property)作为 Objective-C 的一项特性，主要的作用就在于封装对象中的数据。 Objective-C 对象通常会把其所需要的数据保存为各种实例变量。实例变量一般通过“存取方法”(access method)来访问。其中，“获取方法”(getter)用于读取变量值，而“设置方法”(setter)用于写入变量值。这个概念已经定型，并且经由“属性”这一特性而成为Objective-C 2.0的一部分。而在正规的 Objective-C 编码风格中，存取方法有着严格的命名规范。 正因为有了这种严格的命名

规范，所以 Objective-C 这门语言才能根据名称自动创建出存取方法。其实也可以把属性当做一种关键字，其表示:

编译器会自动写出一套存取方法，用以访问给定类型中具有给定名称的变量。 所以你也可以这么说:

@property = getter + setter;

例如下面这个类:

```
@interface Person : NSObject
@property NSString *firstName;
@property NSString *lastName;
@end
```

上述代码写出来的类与下面这种写法等效:

```
@interface Person : NSObject
- (NSString *)firstName;
- (void)setFirstName:(NSString *)firstName;
- (NSString *)lastName;
- (void)setLastName:(NSString *)lastName;
@end
```

ivar、getter、setter 是如何生成并添加到这个类中的

“自动合成”(autosynthesis)

完成属性定义后，编译器会自动编写访问这些属性所需的方法，此过程叫做“自动合成”(autosynthesis)。需要强调的是，这个过程由编译 器在编译期执行，所以编辑器里看不到这些“合成方法”(synthesized method)的源代码。除了生成方法代码 getter、setter 之外，编译器还要自动向类中添加适当类型的实例变量，并且在属性名前面加下划线，以此作为实例变量的名字。在前例中，会生成两个实例变量，其名称分别为 _firstName与_lastName。也可以在类的实现代码里通过@synthesize语法来指定实例变量的名字.

```
@implementation Person
@synthesize firstName = _myFirstName;
@synthesize lastName = myLastName;
@end
```

我为了搞清属性是怎么实现的,曾经反编译过相关的代码,大致生成了五个东西:

1）OBJC_IVAR_$类名$属性名称 ： 该属性的“偏移量” (offset)，这个偏移量是“硬编码” (hardcode)，表示该变量距离存放对象的内存区域的起始地址有多远。

2）setter与getter方法对应的实现函数

3）ivar_list ： 成员变量列表

4）method_list ： 方法列表

5）prop_list ： 属性列表

也就是说我们每次在增加一个属性,系统都会在ivar_list中添加一个成员变量的描述,在method_list中增加setter与getter方法的描述,在属性列表中增加一个属性的描述,然后计算该属性在对象中的偏移量,然后给出setter与getter方法对应的实现,在setter方法中从偏移量的位置开始赋值,在getter方法中从偏移量开始取值,为了能够读取正确字节数,系统对象偏移量的指针类型进行了类型强转.
165问:如何让自己的类用 copy 修饰符 如何重写带 copy 关键字的 setter
答:若想令自己所写的对象具有拷贝功能，则需实现NSCopying协议。如果自定义的对象分为可变版本与不可变版本，那么就要同时实现NSCopyiog与NSMutableCopying协议。

具体步骤：

1）需声明该类遵从NSCopying协议

2）实现NSCopying协议。该协议只有一个方法:

– (id)copyWithZone: (NSZone*) zone
注意：一提到让自己的类用 copy 修饰符，我们总是想覆写copy方法，其实真正需要实现的却是“copyWithZone”方法。
166问:这个写法会出什么问题： @property (copy) NSMutableArray *array;
答:两个问题：
1、添加,删除,修改数组内的元素的时候,程序会因为找不到对应的方法而崩溃.因为copy就是复制一个不可变NSArray的对象；
2、使用了atomic属性会严重影响性能。

第1条的同上一个问题。

第2条原因，如下：

该属性使用了同步锁，会在创建时生成一些额外的代码用于帮助编写多线程程序，这会带来性能问题，通过声明nonatomic可以节省这些虽然很小但是不必要额外开销。

在默认情况下，由编译器所合成的方法会通过锁定机制确保其原子性(atomicity)。如果属性具备nonatomic特质，则不使用同步锁。请注意，尽管没有名为“atomic”的特质(如果某属性不具备nonatomic特质，那它就是“原子的”(atomic))。

在iOS开发中，你会发现，几乎所有属性都声明为nonatomic。

一般情况下并不要求属性必须是“原子的”，因为这并不能保证“线程安全”（ thread safety)，若要实现“线程安全”的操作，还需采用更为深层的锁定机制才行。例如，一个线程在连续多次读取某属性值的过程中有别的线程在同时改写该值，那么即便将属性声明为atomic，也还是会读到不同的属性值。

因此，开发iOS程序时一般都会使用nonatomic属性。但是在开发Mac OS X程序时，使用 atomic属性通常都不会有性能瓶颈。
167问:怎么用 copy 关键字
答:用途：

1）NSString、NSArray、NSDictionary 等等经常使用copy关键字，是因为他们有对应的可变类型：NSMutableString、NSMutableArray、NSMutableDictionary；

2）block也经常使用copy关键字，具体原因见官方文档：Objects Use Properties to Keep Track of Blocks：

block使用copy是从MRC遗留下来的“传统”,在MRC中,方法内部的block是在栈区的,使用copy可以把它放到堆区.在ARC中写不写都行：对于block使用copy还是strong效果是一样的，但写上copy也无伤大雅，还能时刻提醒我们：编译器自动对block进行了copy操作。

copy此特质所表达的所属关系与strong类似。然而设置方法并不保留新值，而是将其“拷贝”(copy)。 当属性类型为NSString时，经常用此特质来保护其封装性，因为传递给设置方法的新值有可能指向一个NSMutableString类的实例。这个类是NSString的子类，表示一种可修改其值的字符串，此时若是不拷贝字符串，那么设置完属性之后，字符串的值就可能会在对象不知情的情况下遭人更改。所以，这时就要拷贝一份“不可变”(immutable)的字符串，确保对象中的字符串值不会无意间变动。只要实现属性所用的对象是“可变的”(mutable)，就应该在设置新属性值时拷贝一份。

用@property声明 NSString、NSArray、NSDictionary 经常使用copy关键字，是因为他们有对应的可变类型：NSMutableString、NSMutableArray、NSMutableDictionary，他们之间可能进行赋值操作，为确保对象中的字符串值不会无意间变动，应该在设置新属性值时拷贝一份。
168问:什么情况使用 weak 关键字，相比 assign 有什么不同
答:1）在ARC中,在有可能出现循环引用的时候,往往要通过让其中一端使用weak来解决,比如:delegate代理属性

2）自身已经对它进行一次强引用,没有必要再强引用一次,此时也会使用weak,自定义IBOutlet控件属性一般也使用weak；当然，也可以使用strong。

不同点：

1）weak 此特质表明该属性定义了一种“非拥有关系”(nonowning relationship)。为这种属性设置新值时，设置方法既不保留新值，也不释放旧值。此特质同assign类似， 然而在属性所指的对象遭到摧毁时，属性值也会清空(nil out)。 而 assign 的“设置方法”只会执行针对“纯量类型”(scalar type，例如 CGFloat 或 NSInteger 等)的简单赋值操作。

2）assigin 可以用非OC对象,而weak必须用于OC对象
169问:定义一个OC标准的枚举命令
答:官方示例1：

```
typedef NS_ENUM(NSInteger, UITableViewCellStyle) {
    UITableViewCellStyleDefault,
```

```
    UITableViewCellStyleValue1,
    UITableViewCellStyleValue2,
    UITableViewCellStyleSubtitle
};
```

官方示例2：

```
typedef NS_OPTIONS(NSUInteger, UIViewAutoresizing) {
    UIViewAutoresizingNone                 = 0,
    UIViewAutoresizingFlexibleLeftMargin   = 1 << 0,
    UIViewAutoresizingFlexibleWidth        = 1 << 1,
    UIViewAutoresizingFlexibleRightMargin  = 1 << 2,
    UIViewAutoresizingFlexibleTopMargin    = 1 << 3,
    UIViewAutoresizingFlexibleHeight       = 1 << 4,
    UIViewAutoresizingFlexibleBottomMargin = 1 << 5
};
```

170问:对比函数指针和Blocks使用方法

答:函数指针定义：int func (int count)(return count +1) ； int (*funcptr)(int) = &func;

函数指针使用：int result = (*funcptr)(10);

Blocks定义：int (^blk)(int) = ^(int count){return count+1;};

Blocks使用：int result = blk(10);

171问:const常量与宏定义区别

答:(1) 编译器处理方式不同

define宏是在预处理阶段展开。

const常量是编译运行阶段使用。

(2) 类型和安全检查不同

define宏没有类型，不做任何类型检查，仅仅是展开。

const常量有具体的类型，在编译阶段会执行类型检查。

(3) 存储方式不同

define宏仅仅是展开，有多少地方使用，就展开多少次，不会分配内存。

const常量会在内存中分配(可以是堆中也可以是栈中)。

(4)const 可以节省空间，避免不必要的内存分配。 例如：

```
#define PI 3.14159 //常量宏
const doulbe Pi=3.14159; //此时并未将Pi放入ROM中 ......
double i=Pi; //此时为Pi分配内存，以后不再分配！
double I=PI; //编译期间进行宏替换，分配内存
double j=Pi; //没有内存分配
double J=PI; //再进行宏替换，又一次分配内存！
```

const定义常量从汇编的角度来看，只是给出了对应的内存地址，而不是象#define一样给出的是立即数，所以，const定义的常量在程序运行过程中只有一份拷贝，而 #define定义的常量在内存中有若干个拷贝。

(5) 提高了效率。 编译器通常不为普通const常量分配存储空间，而是将它们保存在符号表中，这使得它成为一个编译期间的常量，没有了存储与读内存的操作，使得它的效率也很高。

const 与 #define的比较

C++ 语言可以用const来定义常量，也可以用 #define来定义常量。但是前者比后者有更多的优点：

（1） const常量有数据类型，而宏常量没有数据类型。编译器可以对前者进行类型安全检查。而对后者只进行字符替换，没有类型安全检查，并且在字符替换可能会产生意料不到的错误（边际效应）。

（2） 有些集成化的调试工具可以对const常量进行调试，但是不能对宏常量进行调试。

172问:讲讲OC和其他语言好在哪里,不好在哪里

答: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) 不是一一个过度复杂的 C 衍生生语言言

7) Objective-C++ 与 C++ 可混合编程

缺点:

1) 不支支援命名空間

2) 不支支持运算符重载

3)不支支持多重继承

173问:const的用法

答:(1)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”。174问:把你知道的framework写出来，写出功能

答:UIKit.framework 包含iOS应用程序用户界面层使用的类和方法，CoreGraphics.framework 包含 Quartz 2D 绘图API 接口，Foundation.framework 为CoreFoundation 框架的许多功能提供 Objective-C 封装，CoreFoundation.framework 一组C 语言接口,它们为iOS 应用程序提供基本数据管理和服务功能，MapKit.framework 该框架供一个可被嵌入到应用程序的地图界面,该界面包含一个可以滚动的地图视图，CoreLocation.framework 可用于定位某个设备当前经纬度，CFNetwork.framework提供一组高性能基于C 语言的接口,它们为使用网络协议提供面向对象抽象。

175问:project workspace scheme的区别

答:Project:

一一般的某个应用用单独新建一一个project就可以了,然后把所有的程序文文件都放在里里面面,这个可以满足足大大部分普通的需求。

Workspace:

项罒目目有可能要使用用其他的项罒目目文文件,或者引入入其他的静态库文文件,这个时候workspace就派上用场了,workspace既可以单独管理多个项罒目目,又又可以通过配置,让各个项罒目目相互依赖。

Scheme:

Xcode scheme定义了编译集合中的若干target,编译时的一一些设置以及要执行行的测试集合。可以自自定义若干个scheme,但是同一一时刻只能运行行一一个。

176

问:SEL和一个函数指针调用同一个方法的结果是不是相同

答:得到了SEL变量之后,可以通过下面面的调用用来给一一个对象发送消息:

[对象performSelector:SEL变量withObject:参数1 withObject:参数2];

结论:SEL只是方方法名标识,实际运行行时需要通过消息发送来调用用,IMP是"implementation"的缩 写,它是objetive-C 方方法 (method)实现代码块的地址,类似函数指针,通过它可以 直接访问任意一一个方方法。免去发送消息的代价。

177问:对于Objective-C，你认为它最大的优点和最大的不足是什么 对于不足之处，现在有没有可用的方法绕过这些不足来实现需求。如果可以的话，你有没有考虑或者实践过重新实现OC的一些功能，如果有，具体会如何做

答:最大的优点是它的运行时特性，不足是没有命名空间，对于命名冲突，可以使用长命名法或特殊前缀解决，如果是引入的第三方库之间的命名冲突，可以使用link命令及flag解决冲突。

178问:是否做过异步的网络处理和通讯方面的工作 如果有，能具体介绍一些实现策略么

答:使用NSOperation发送异步网络请求，使用NSOperationQueue管理线程数目及优先级，底层是用NSURLConnetion。

179问:使用GCD以及block时要注意些什么 它们两是一回事儿么 block在ARC中和传统的MRC中的行为和用法有没有什么区别，需要注意些什么 如何避免循环引用

答:使用block是要注意，若将block做函数参数时，需要把它放到最后，GCD是Grand Central Dispatch，是一个对线程开源类库，而Block是闭包，是能够读取其他函数内部变量的函数。

180问:你用过NSOperationQueue么 如果用过或者了解的话，你为什么要使用NSOperationQueue，实现了什么 请描述它和GCD的区别和类似的地方

答:使用NSOperationQueue用来管理子类化的NSOperation对象，控制其线程并发数目。GCD和NSOperation都可以实现对线程的管理，区别是 NSOperation和NSOperationQueue是多线程的面向对象抽象。项目中使用NSOperation的优点是NSOperation是对线程的高度抽象，在项目中使用它，会使项目的程序结构更好，子类化NSOperation的设计思路，是具有面向对象的优点（复用、封装），使得实现是多线程支持，而接口简单，建议在复杂项目中使用。

项目中使用GCD的优点是GCD本身非常简单、易用，对于不复杂的多线程操作，会节省代码量，而Block参数的使用，会是代码更为易读，建议在简单项目中使用。

181问:NSNotification和KVO的区别和用法是什么 什么时候应该使用通知，什么时候应该使用KVO，它们的实现上有什么区别吗 如果用protocol和delegate（或者delegate的Array）来实现类似的功能可能吗 如果可能，会有什么潜在的问题 如果不能，为什么

答:NSNotification是通知模式在iOS的实现，KVO的全称是键值观察(Key-value observing),其是基于KVC（key-value coding）的，KVC是一个通过属性名访问属性变量的机制。例如将Module层的变化，通知到多个Controller对象时，可以使用NSNotification；如果是只需要观察某个对象的某个属性，可以使用KVO。

对于委托模式，在设计模式中是对象适配器模式，其是delegate是指向某个对象的，这是一对一的关系，而在通知模式中，往往是一对多的关系。委托模式，从技术上可以现在改变delegate指向的对象，但不建议这样做，会让人迷惑，如果一个delegate对象不断改变，指向不同的对象。

182问:是否使用过CoreText或者CoreImage等 如果使用过，请谈谈你使用CoreText或者CoreImage的体验

答:CoreText可以解决复杂文字内容排版问题。CoreImage可以处理图片，为其添加各种效果。体验是很强大，挺复杂的。

183问:是否使用过CoreAnimation和CoreGraphics。UI框架和CA，CG框架的联系是什么 分别用CA和CG做过些什么动画或者图像上的内容
答:UI框架的底层有CoreAnimation，CoreAnimation的底层有CoreGraphics。
UIKit |
---------- |
Core Animation |
Core Graphics |
Graphics Hardware|
使用CA做过menu菜单的展开收起

184问:使用过Objective-C的运行时编程（Runtime Programming）么 如果使用过，你用它做了什么 你还能记得你所使用的相关的头文件或者某些方法的名称吗
答:Objecitve-C的重要特性是Runtime（运行时）,在#import <objc/runtime.h 下能看到相关的方法，用过objc_getClass()和class_copyMethodList()获取过私有API;使用
```objective-c
Method method1 = class_getInstanceMethod(cls, sel1);
Method method2 = class_getInstanceMethod(cls, sel2);
method_exchangeImplementations(method1, method2);
```
代码交换两个方法，在写unit test时使用到。

185问:iOS 创建单例的两种方法(MRC ARC)
答:MRC
```
static AccountManager *DefaultManager = nil;

+ (AccountManager *)defaultManager {
    if (!DefaultManager) DefaultManager = [[self allocWithZone:NULL] init];
    return DefaultManager;
}
```

ARC
```
+ (AccountManager *)sharedManager
{
        static AccountManager *sharedAccountManagerInstance = nil;
        static dispatch_once_t predicate;
        dispatch_once(&predicate, ^{
                sharedAccountManagerInstance = [[self alloc] init];
        });
    return sharedAccountManagerInstance;
}
```

186问:写一个NSString类的实现
答:+ (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);
}
```

187问:目标–动作机制是什么
答: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。

188问:什么是键–值,键路径是什么
答: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。

189问:自动释放池是什么,如何工作
答: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.

190问:方法和选择器有何不同
答:selector是一个方法的名字，method是一个组合体，包含了名字和实现.

191问:id、nil代表什么
答: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。

192问: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。

193问:APNS 实现步骤
答:APNS 是Apple Push Notification Service（Apple Push服务器）的缩写，是苹果的服务器。

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

194问:ASIDownloadCache 设置下载缓存
答: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];

195问:HTTP协议详解
答: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。

196问:TCP/UDP区别联系
答: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

197问:socket连接和http连接的区别
答: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。

198问:什么是TCP连接的三次握手
答: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）

199问:利用Socket建立网络连接的步骤
答: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。

200问:多线程 之 NSOperation
答:如果需要让线程同时并行运行多个，可以将线程加入队列（Queue）中，NSOperationQueue类就是一个线程队列管理类，他提供了线程并行、队列的管理。可以认为NSOperationQueue就是一个线程管理器，通过addOperations方法，我们可以一次性把多个（数组形式）线程添加到队列中。同时，NSOperationQueue允许通过setMaxConcurrentOperationCount方法设置队列的并行（同一时间）运行数量

201问:多线程 之 NSThread
答:NSThree是官方推荐的线程处理方式，它在处理机制上，需要开发者负责手动管理Thread的生命周期，包括子线程与主线程之间的同步等。线程共享同一应用程序的部分内存空间，它们拥有对数据相同的访问权限。你得协调多个线程 对同一数据的访问，一般做法是在访问之前加锁，这会导致一定的性能开销。在 iOS 中我们可以使用多种形式的 thread。 比其他两个轻量级 需要自己管理线程的生命周期，线程同步。 线程同步对数据的加锁会有一定的系统开销

202问:多线程 之 GCD
答:Grand Central Dispatch (GCD)是Apple开发的一个多核编程的解决方法。该方法在Mac OS X 10.6雪豹中首次推出，并随后被引入到了iOS4.0中。GCD是一个替代诸如NSThread, NSOperationQueue, NSInvocationOperation等技术的很高效和强大的技术，它看起来象就其它语言的闭包(Closure)一样，但苹果把它叫做blocks。

203问:Objective-C的优点
答:Objective-C语言有C++ Java等面向对象的特点，那是远远不能体现它的优点的。Objective-C的优点是它是动态的。动态能力有三种：
动态类-运行时确定类的对象
动态绑定-运行时确定要调用的方法
动态加载--运行时为程序加载新的模块

204问:Socket连接和Http连接的区别
答: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。

205问:TCP/UDP区别联系
答: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

206问:GCD高级理解
答: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];
    });
```

207问:obj-c的优缺点
答: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。

208问:用 C++设计一个不能被继承的类
答:template <typename T class A {
friend T; private:
A() {} ~A() {}
};
class B : virtual public A<B {
public:
B() {}
~B() {} };
class C : virtual public B
{ public:
C() {}
~C() {} };
void main(void) {
B b; //C c; return;
} 注意:构造函数是继承实现的关键,每次子类对象构造时,首先调用的是父类的构造函数,然后才是自己的。

209问:不同屏幕怎么适配
答:iphone, iphone3G, iphone3GS 320x480
iphone4, iphoen4S 640x960 retina
iphone5, iphone5S, 640x1136
写程序需要有 2 套图片 demo.png demo@2x.png iphone5 适配.
1136/2-44-49

[[UIScreen mainScreen] applicationFrame] = (320x460, 320x548) [[UIScreen mainScreen] bounds] = (320x480, 320x568)
iPad, iPad2, iPad Mini 1024x768
iPad3, iPad4, 2048x1563

210问:数据库能不能存图片,怎么存
答:可以存图片, nsdata 存,但是我们一般不这样存,我们存路径把 图片存在沙盒中

211问:成员变量的作用域有哪几种
答:@public, @protected, @private

212问: OC 中有私有成员变量吗
答:在.m 文件中实现匿名类别 Category

213问: OC 中有私有方法吗
答:在.m 文件中实现 Category,只是在.m 中方法

214问:解释关键字static、const、inline、 static inline、volatile
答:static 修饰变量表示静态变量,根据作用域 不同
extern 和 static 是相反的。
Static 修饰函数是表示函数在本文件有效
Static inline 联合在一起表示内联函数。类似于宏
Const 修饰变量和变量的内容只读
Volatile 表示每次都从内存中真正的读取。主要是用于嵌入式中, 读取硬件。

215问:什么是通知中心
答:通知中心是多对多的平等的通讯模式。主要用在多个对象之间松 耦合的通讯模式。对象和对象之间通过通知的名字就可以进行关 联通讯。

216问:什么是委托代理模式
答:委托代理是 2 个对象之间的一种通讯方式。
一方使用协议,代理，另外一方实现协议,类似于回调,blocks 等语法。
目的是为了软件设计的低耦合

217问:ScrollView 的复用机制
答:滚动过程中把 scrollview 里面的 超出屏幕的 view 放在重用队列中, 在滚动过程中如果有新的 view 进来,首先从重用队列中取得 view, 如果没有就新创建一个。

218问:Frame 和 bounds 有什么区别
答:Frame 是子视图相对于父视图的坐标系统,x y width height, bounds 是 uiview 自己的坐标系统 bounds x y=0

219问:Blocks 语法有什么优点
答:Blocks 类似于 c 中回调函数,和代理类似的,类似于 java 中的 listener 匿名函数

220
C++怎么调用 C 的方法
答:extern “C” int foo(){ };

221问:NSArray, NSDictionary 这些是如何做的
答:NSArray 使用链表做的 NSDictionary 使用 hash 表做出来的

222问:遍历数组的三种方式有什么区别
答:for forin iterator
forin 也叫快速循环。为什么他叫快速循环,原因是速度快 为什么速度快 可以通过优化提前把 forin 里面的 array 数组数组全部导入到 cpu cache 中。因为 cache 访问速度把内存快至少 100 倍

223问:HTTP 协议、Socket 协议的区别
答:HTTP 启动一个 NSUrlConnection 在 didFinish 长连接,长期的和服务器保持连接。目的是为了避免每次都要连 接。因为和服务器连接的过程非常复杂。也耗时。所以连接一次。 HTTP 短连接只要处理完就断掉,下次重新连接 对服务器的负担 小.
长连接对服务器压力很大。

224问:怎么实现对象的本地存储
答:归档 NSArchieve

225问:copy 和 mutableCopy 区别
答:copy 是拷贝,mutableCopy 是可变拷贝,比如把 NSString 通过 mutableCopy 变 成 NSMutableString, 把 NSArray 通 过 mutableCopy 变成 NSMutableArray.
NSString –à?NSMutableString, NSArrayà?NSMutableArray,
NSDictionaryà?NSMutableDictionary, NSData– NSMutableData;

226copy 是深拷贝还是浅拷贝
答:copy 缺省是浅拷贝.但是 copy 也可以做成深拷贝

227问: retain、copy、readonly、readwrite 的区别
答:retain对计数器+1 copy是创建一个新对象,readonly只产生getter 函数,readwrite 产生 setter, getter 函数

228问:解释OC里的id类型
答:id 是泛型指针,可以指向任何 oc 类型,
id =~ NSObject *
id 是 objc_object 的 typedef,执行 Class 对象

229问:本地存储方式有哪几种
答:数据库,nsuserdefauts, file , plist,archieve 归档 比较小的就放在 nsuserdefaults(以文件设计的存放在 Libraray 中) 数据库里面适合存放一条一条的记录
file 一般存普通文件,图片,视频,音频等

plist 存放比 nsuserdefaults 大一些的
归档 nsarchieve 可以存对象

230问:POST 和 GET 有何区别
答:GET 和 POST 都是 HTTP 请求方式的 2 中。
POST 是安全的。GET 是不安全的。GET 是放在浏览器中地址暴露 出来了。POST 不会。但是在 App 上 GET 和 POST 都看不见。 GET 和 POST 都是和服务器提交参数/通讯的一种方式。
GET 参数不能太长<1024B POST 没有限制<4G
GET 不能上传文件, POST 可以上传文件

231问:POST 请求的链接参数怎么拼接
答:POST 参数有 2 中,一种文件 POST 一种非文件 POST,对于非文 件 POST 格 式 (form-data/ x-www-urlencoded) 是 name=xxx&id=22&sxx=33
对于文件 POST(multiple/form-data)

232 C/C++和 OC 怎么混用
答:.m - .mm

233问:NSString *name = @"1000phone.com",[name release]会出现什么 情况
答:这个代码不满足 objective-c 的内存管理黄金法则。没有 alloc 就
release

234问:什么是单例模式
答:单例就是在多个对象之间共享数据,类似于全局变量,比如数据 库打开一次,多个界面都可以使用

235问:解释KVC和KVO
答:Key value coding, Key value observer.
Kvc 是路径访问的规范,kvo 是观察某个变量的变化过程
KVO 可以观察某个对象的变量变化过程,KVC 是满足被观察的编 码规范。
KVC/KVO 类似于代理,通知中心。都是一种通讯方法。

236问:OC 里怎么实现多继承
答:OC 没有多继承。用协议实现多继承,把协议中的方法在实现的类 中重写 C++有多继承,多继承其实用好了很方便。C++多继承很容易出现 二一性比如:
类 A { int a;}
类B: 类A,类C:类A
类D : B, C那么 int a到底是继承自谁的 单继承

237问:解释 const, static, inline 关键字
答:const 修饰指针,或者常量,比如不可变,
static 修饰变量表示作用域,比如全局的私有变量,函数内部的 static 是内部的私有变量。
Static 修饰函数表示函数是文件作用域
Inline 表示内联。一般来说 inline 需要和 static 联合用 一般用法是 static inline int max(int a, int b) {

return a b a:b; }
static inline作用是和宏类似,只不过是方便调试(宏不能断掉调 试,static inline 可以)。运行时候是一样的。
一般 c/c++短小的函数都用 static inline 内联函数

238问:下拉刷新需要实现哪几个方法,刷新流程
答:下来一般使用 EGORefresh 进行。原理是利用 scrollview 的反弹效 果把刷新 view 加载 scrollview 的负坐标上。通过代理方法去触发。

239问:如何调用 iOS 打电话,发短信
答:[[UIApplication sharedApplication] openURL:[NSURL urlwithString:@"tel://1543434"]];
[[UIApplication sharedApplication] openURL:[NSURL urlwithString:@"sms://1572234"]];
[[UIApplication sharedApplication] openURL:[NSURL urlwithString:@"mail://hello@hello.com"]];
[[UIApplication sharedApplication] openURL:[NSURL urlwithString:@"http://1000phone.net"]];

240问:Objective-C 如何和 javascript 通讯
答:Objective-c --à?javascript ---à?html 里面内容
Objective-c 是通过 stringByEvaluatingJavaScriptFromString 函数 来执行 html 中的 javascript
Javascript -à?url --à?objective-c 本地方法
需要通过 uiwebview 中的代理函数
-(BOOL)webView:(UIWebView *)webView shouldStartLoadWithRequest:(NSURLRequest *)request navigationType:(UIWebViewNavigationType)navigationType;

241问:怎么解析 HTML 源码
答:一般 HTML 是通过 webview 来显示的。一般简单的 html 是通过 解析字符串来解析。没有成熟现成 html 解析器。 一般服务器去抓取html解析html形成json xml文件供手机端读 取。

242UITextField、UITextView 的区别
答:UITextField不能换行,UITextView可以换行。TextView可以点击
字体放大,可以复制。相同的是他们都有代理类似

243问:代理和通知中心/广播/NoficationCenter 什么区别
答:代理主要是反向传值,一般用来 1:1 的两个对象通讯上。
通知中心是通过注册然后接收事件的一种 n:n(多对对) 的方法

244问:代理和协议什么区别
答:代理是一种概念,协议是一种技术,代理是用协议来实现的,代理 是 2 个对象之间通讯的一种方式。代理主要做反向传值的。实现系统的一些回调方法,比如 scrollview 滑动事件,选择照片,asi 网络下载完成等。

245问: 什么是异步
答:相对于同步来说,单独起一个或者多线程去处理 异步是一个概念 线程还是一个技术,异步就是用线程这种技术 实现的

比如界面下载数据,我们启动一个异步任务 ASI 去网络下载数据, 然后异步刷新界面,我们无需等待网络数据下载完成.

246问:解释多线程、NSThread 、NSOperation、GCD

答:多线程在 iOS 用的很多,比如每个 asi 请求,sdwebimage 请求,数 据请求等待等网络数据处理,多线程/异步就是主要是为了界面流 畅,防止界面假死。
每一个 ASI 请求就是一个 NSOperation
每一个 NSUrlConnnection 也是一个线程
Nsthread 是创建线程的一个通用的类。比如线程创建,取消,开 始等。
Nsoperation 就是一个简单的以任务为导向的多线程模型。目的是 为了不懂操作系统,不懂线程的人使用的
GCD 类似于 NSOperation, 是一个 blocks 版本的线程模型。

247 C 和 OC 有哪些基本数据类型
答:C char, short, int, struct, union, enum
NSString, nsarray, nsdictionary.

248问:类别和继承什么区别
答:类别/类目/Category 很方便给现有类添加方法。但是不能添加成 员变量,匿名类除外,比如可以给 NSString 增加方法,给 UINavigationBar 增加方法,比如 SDWebImage 给 UIImageView 增 加了 setImageWithURL:方法。类别对于使用者很方便
继承可以给现有的类增加方法和成员变量。继承对于使用者来 说不如类别方便 ,比如对 SDWebImage 如果用继承的话,那么就 需要写一个类似于 QFImageView 然后把所有的 UIImageView 改成
QFImageView,这里不如类别方便

249问:类别的作用
答:类别是给已有的类添加方法,但是不能添加变量,匿名类别除外 ()

250问:你如何理解复用机制
答:一般是对 uiscrollview 做复用机制,因为 uiscrollview 滚动窗口没 有复用,所以要做,原理就是超过屏幕的 view 不能销毁,而要放 在复用队列/池里面存放起来,然后以后要在scrollview显示view 首先不要 alloc 创建,而要首先去复用池里面找有没有可复用的 view,如果没有就 alloc 如果有就直接用。

251问:瀑布流怎么理解和实现的
答:1. 如果简单的来说,用 3 个 tableview 就可以实现瀑布流,3 个 tableview 实现联动滚动。
2. 其实最好的做法在 scrollview 上使用 3 个复用队列,如果一种 一个 cell 超过屏幕,不能 release,而是把它回收到复用队列中, 如果要创建一个 cell,首先从复用队列中取一个,然后使用。

252常见的点击手势有哪几种
答:UIPinchGuesture, UITapGuesture, UISwipGuesture,
UILongPressGuesture, UIRotationGuesture

253问:什么是 MVC 模式

答:Model, View, Controller, 是 iOS 开发中的典型的设计模式。比如通 过 http 把网络数据下载并解析然后存于自己的数据模型 Model 中, 然后通知 controller 去刷新界面,通过 controller 讲 Model 和 View 进行关联,这种模式就叫 mvc。
这样的好处是,可以隔离数据模型 model 和 View 界面。遵循了低 耦合的设计思想.

254问:界面之间传值有哪几种方法
答:单例,代理,直接赋值,通知中心/广播, 数据库等多种

255问:解释 TCP/IP 协议
答:TCP/IP 是网络开发中常见的传输协议,他传输和 udp 相比是可靠 的。http 是基于 tcp/ip 的主要用户互联网的协议
所谓可靠是 tcp 传输对方会给一个 ACK 信号(确认信号)
tcp 传输不如 udp 快,吞吐量不如 udp 大
tcp 是顺序的,udp 是无序的
tcp 会保持连接,udp 不会保持连接

256问:浅拷贝与深拷贝的区别 或者什么是深拷贝 什么是浅拷贝
答:copy, mutableCopy
@interface A {
B *b; }
浅拷贝只是拷贝对象本身,不会对里面的子对象进一步拷贝
深拷贝会对子对象以及子对象的子对象进一步拷贝

257问:C语言里的数组与OC数组的区别
答:OC 数组是一个对象,有大量的方法,c 没有都需要自己写 C 数组删除是需要后面往前移动,oc 数组自动处理

258问:你如何理解 iOS 内存管理
答:1. new alloc copy retain这些对象我们都要主动的release或者 autorelease
2. 如果是类方法创建的对象,那么系统自动释放池自动在适当的 时候会帮我们 release
3. ARC xcode 自动会帮我们人工智能的添加 release autorelease 操 作

259问:在项目什么时候选择使用GCD，什么时候选择NSOperation
答:项目中使用NSOperation的优点是NSOperation是对线程的高度抽象，在项目中使用它，会使项目的程序结构更好，子类化NSOperation的设计思路，是具有面向对象的优点（复用、封装），使得实现是多线程支持，而接口简单，建议在复杂项目中使用。
项目中使用GCD的优点是GCD本身非常简单、易用，对于不复杂的多线程操作，会节省代码量，而Block参数的使用，会是代码更为易读，建议在简单项目中使用。

260问:ASI 原理是什么 请举例来说明你平时用的那些文件
答:ASI 使用apple底层 CFNetworking 框架实现的，而不是用 Socket 套接字实现的。 他是一个基于NSOperation（抽象类）的线程处理网络框架 CFNetwork 是基于InputSteam / Outstream 流的方式管理数据，它内部使用了多线程异步模式进行数据的通讯，比如数据上传进度，下载进度，缓存的管理机制，大文件下载，大文件上传，安全机制。

261问:什么是block block 实现原理 多线程与block有什么关系 写个例子。
答: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();});
```

262问:属性readwrite，readonly，assign，retain，copy，nonatomic 各是什么作用，在那种情况下用
答: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。

263问: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
答:编译时是NSString的类型；运行时是NSData类型的对象。

264问:如何理解线程
答:（1）iOS 线程是为了界面流程，防止页面假死。比如解析一个大文件，比如一个地区的详情文件，解析时间比如是10秒，那么不用线程的话，界面就会假死、卡顿10秒左右。用户体验差，那么解决方法就是很快把界面暂时 启动线程去后台解析;

（2）再比如做图片滤镜，图像操作会耗费很多时间，比如5秒左右，界面就会假死;
（3）网络下载数据，用于网络状态好坏时间不确定，那么我们需要使用线程 来处理这中不确定的关系，一定数据接收完成，那么就可以通知主线程处理;
（4）再比如读取数据全国电话号码数据库，时间很长，那么就需要通过线程来做;
（5）特别强调的是，线程不是快，就是为了给用户一个快的假象.

265问:描述一下iOS SDK中如何实现MVC的开发模式
答: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。

266问:Objective-C如何对内存管理的,说说你的看法和解决方法
答: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。

267问:说说为什么要用RAC，它是如何简化代码的
答:ReactiveCocoa（其简称为RAC）是由GitHub开源的一个应用于iOS和OS X开发的新框架。RAC具有函数式编程和响应式编程的特性，可以大大简化你的代码，并且目前看来安全可靠。
RAC通过如下几点来达到简化代码的作用：

1、事件监听
RAC最基本的入门使用技巧就是对事件的监听，在iOS开发中，我们所说的点击事件其实就是target-action，接触过iOS开发的人都不会陌生UIControlEventTouchUpInside，这就是按下并松开的动作，不仅仅是UIButton，还有UITextField也有目标-动作模式。

2、代理
用RAC写代理是有局限的，它只能实现返回值为void的代理方法；
为什么要用RAC写代理 还是四个字“简化代码”。为什么在这里强调这个，是因为RAC的优点在代理方法中体现得淋漓尽致。

3、通知
开发中通知是一个比较常用的功能，主要的应用场景是某个页面进行数据重传需要更新model但是点击返回栈时不会刷新返回界面的数据，这时可以用通知来更新另一个页面的数据，我们也可以在另一个页面的ViewDidAppear方法中刷新数据。

4、KVO
RAC中的KVO大部分都是宏定义，所以代码异常简洁，简单来说就是RACObserve(TARGET, KEYPATH)这种形式，TARGET是监听目标，KEYPATH是要观察的属性值，举个例子，如果UIScrollView滚动则输出success。

268问:runtime如何实现weak变量的自动置nil
答:runtime对注册的类会进行布局，对于weak对象会放入一个hash表中。 用weak指向的对象内存地址作为key，当此对象的引用计数为0的时候会dealloc。假如weak指向的对象内存地址是a，那么就会以a为键，在这个 weak 表中搜索，找到所有以a为键的weak对象，从而设置为nil。
weak修饰的指针默认值是nil（在Objective-C中向nil发送消息是安全的）

269问:objc_msgForward函数是做什么的，直接调用它将会发生什么
答:_objc_msgForward是IMP类型，用于消息转发的：当向一个对象发送一条消息，但它并没有实现的时候，_objc_msgForward会尝试做消息转发。
IMP msgForward = _objc_msgForward;
如果手动调用objcmsgForward，将跳过查找IMP的过程，而是直接触发“消息转发”，进入如下流程：

- 第一步：+ (BOOL)resolveInstanceMethod:(SEL)sel实现方法，指定是否动态添加方法。若返回NO，则进入下一步，若返回YES，则通过class_addMethod函数动态地添加方法，消息得到处理，此流程完毕。

- 第二步：在第一步返回的是NO时，就会进入–(id)forwardingTargetForSelector:(SEL)aSelector方法，这是运行时给我们的第二次机会，用于指定哪个对象响应这个selector。不能指定为self。若返回nil，表示没有响应者，则会进入第三步。若返回某个对象，则会调用该对象的方法。

- 第三步：若第二步返回的是nil，则我们首先要通过– (NSMethodSignature *)methodSignatureForSelector:(SEL)aSelector指定方法签名，若返回nil，则表示不处理。若返回方法签名，则会进入下一步。

- 第四步：当第三步返回方法方法签名后，就会调用– (void)forwardInvocation:(NSInvocation *)anInvocation方法，我们可以通过anInvocation对象做很多处理，比如修改实现方法，修改响应对象等
- 第五步：若没有实现– (void)forwardInvocation:(NSInvocation *)anInvocation方法，那么会进入– (void)doesNotRecognizeSelector:(SEL)aSelector方法。若我们没有实现这个方法，那么就会crash，然后提示打不到响应的方法。到此，动态解析的流程就结束了。

270问:runtime如何通过selector找到对应的IMP地址
答:每个selector都与对应的IMP是一一对应的关系，通过selector就可以直接找到对应的IMP：

271问:为什么其他语言里叫函数调用， objective c里则是给对象发消息（或者谈下对runtime的理解）
答:先来看看怎么理解发送消息的含义：
[receiver message]会被编译器转化为：
objc_msgSend(receiver, selector)
如果消息含有参数，则为：
objc_msgSend(receiver, selector, arg1, arg2, ...)
如果消息的接收者能够找到对应的selector，那么就相当于直接执行了接收者这个对象的特定方法；否则，消息要么被转发，或是临时向接收者动态添加这个selector对应的实现内容，要么就干脆玩完崩溃掉。
现在可以看出[receiver message]真的不是一个简简单单的方法调用。因为这只是在编译阶段确定了要向接收者发送message这条消息，而receive将要如何响应这条消息，那就要看运行时发生的情况来决定了。
OC 的 Runtime 铸就了它动态语言的特性，Objc Runtime使得C具有了面向对象能力，在程序运行时创建，检查，修改类、对象和它们的方法。可以使用runtime的一系列方法实现。

顺便附上OC中一个类的数据结构 /usr/include/objc/runtime.h
struct objc_class {

Class isa OBJC_ISA_AVAILABILITY; //isa指针指向Meta Class，因为Objc的类的本身也是一个Object，为了处理这个关系，runtime就创造了Meta Class，当给类发送[NSObject alloc]这样消息时，实际上是把这个消息发给了Class Object

#if !__OBJC2__
    Class super_class
OBJC2_UNAVAILABLE; // 父类
    const char *name
OBJC2_UNAVAILABLE; // 类名
    long version
OBJC2_UNAVAILABLE; // 类的版本信息，默认为0
    long info
OBJC2_UNAVAILABLE; // 类信息，供运行期使用的一些位标识
    long instance_size
OBJC2_UNAVAILABLE; // 该类的实例变量大小
    struct objc_ivar_list *ivars
OBJC2_UNAVAILABLE; // 该类的成员变量链表
    struct objc_method_list **methodLists
OBJC2_UNAVAILABLE; // 方法定义的链表
    struct objc_cache *cache
OBJC2_UNAVAILABLE; // 方法缓存，对象接到一个消息会根据isa指针查找消息对象，这时会在method Lists中遍历，如果cache了，常用的方法调用时就能够提高调用的效率。
    struct objc_protocol_list *protocols
OBJC2_UNAVAILABLE; // 协议链表
#endif

} OBJC2_UNAVAILABLE;
OC中一个类的对象实例的数据结构（/usr/include/objc/objc.h）：
typedef struct objc_class *Class; /// Represents an instance of a class. struct objc_object {
    Class isa
OBJC_ISA_AVAILABILITY;
}; /// A pointer to an instance of a class.
typedef struct objc_object *id;

向object发送消息时，Runtime库会根据object的isa指针找到这个实例object所属于的类，然后在类的方法列表以及父类方法列表寻找对应的方法运行。id是一个objc_object结构类型的指针，这个类型的对象能够转换成任何一种对象。
然后再来看看消息发送的函数：objc_msgSend函数
在引言中已经对objc_msgSend进行了一点介绍，看起来像是objc_msgSend返回了数据，其实objc_msgSend从不返回数据而是你的方法被调用后返回了数据。下面详细叙述下消息发送步骤：

检测这个 selector 是不是要忽略的。比如 Mac OS X 开发，有了垃圾回收就不理会 retain,release 这些函数了。
检测这个 target 是不是 nil 对象。ObjC 的特性是允许对一个 nil 对象执行任何一个方法不会 Crash，因为会被忽略掉。
如果上面两个都过了，那就开始查找这个类的 IMP，先从 cache 里面找，完了找得到就跳到对应的函数去执行。
如果 cache 找不到就找一下方法分发表。
如果分发表找不到就到超类的分发表去找，一直找，直到找到NSObject类为止。
如果还找不到就要开始进入动态方法解析了，后面会提到。
后面还有：
动态方法解析resolveThisMethodDynamically
消息转发forwardingTargetForSelector

272问:能否向编译后得到的类中增加实例变量 能否向运行时创建的类中添加实例变量 为什么
答:- 不能向编译后得到的类中增加实例变量；
- 能向运行时创建的类中添加实例变量；

解释如下：
因为编译后的类已经注册在 runtime 中，类结构体中的 objc_ivar_list 实例变量的链表 和 instance_size 实例变量的内存大小已经确定，同时runtime 会调用 class_setIvarLayout 或 class_setWeakIvarLayout 来处理 strong weak 引用。所以不能向存在的类中添加实例变量；
运行时创建的类是可以添加实例变量，调用 class_addIvar 函数。但是得在调用 objc_allocateClassPair 之后，objc_registerClassPair 之前，原因同上。

273问:Toll-Free Bridging 是什么 什么情况下会使用
答:Toll-Free Bridging用于在Foundation对象与Core Foundation对象之间交换数据,俗称桥接；
在ARC环境下,Foundation对象转成 Core Foundation对象；
使用__bridge桥接以后ARC会自动管理2个对象；
使用__bridge_retained桥接需要手动释放Core Foundation对象；
在ARC环境下, Core Foundation对象转成 Foundation对象；
使用__bridge桥接,如果Core Foundation对象被释放,Foundation对象也同时不能使用了,需要手动管理Core Foundation对象；
使用__bridge_transfer桥接,系统会自动管理2个对象。

274问:Runtime的相关术语有哪些
答:1.SEL
　它是selector在 Objc 中的表示(Swift 中是 Selector 类)。selector 是方法选择器，其实作用就和名字一样，日常生活中，我们通过人名辨别谁是谁，注意 Objc 在相同的类中不会有命名相同的两个方法。selector 对方法名进行包装，以便找到对应的方法实现。它的数据结构是：
typedef struct objc_selector *SEL;
我们可以看出它是个映射到方法的 C 字符串，你可以通过 Objc 编译器器命令@selector() 或者 Runtime 系统的 sel_registerName 函数来获取一个 SEL 类型的方法选择器。
注意：不同类中相同名字的方法所对应的 selector 是相同的，由于变量的类型不同，所以不会导致它们调用方法实现混乱。

2.id
id 是一个参数类型，它是指向某个类的实例的指针。定义如下：
typedef struct objc_object *id;
struct objc_object { Class isa; };
以上定义，看到 objc_object 结构体包含一个 isa 指针，根据 isa 指针就可以找到对象所属的类。
注意：isa 指针在代码运行时并不总指向实例对象所属的类型，所以不能依靠它来确定类型，要想确定类型还是需要用对象的 –class 方法。PS:KVO 的实现机理就是将被观察对象的 isa 指针指向一个中间类而不是真实类型。

3.Class
typedef struct objc_class *Class;
Class 其实是指向 objc_class 结构体的指针。objc_class 的数据结构如下：

```
struct objc_class {
    Class isa  OBJC_ISA_AVAILABILITY;
#if !__OBJC2__
    Class super_class                                       OBJC2_UNAVAILABLE;
    const char *name                                        OBJC2_UNAVAILABLE;
    long version                                            OBJC2_UNAVAILABLE;
    long info                                               OBJC2_UNAVAILABLE;
    long instance_size                                      OBJC2_UNAVAILABLE;
    struct objc_ivar_list *ivars                            OBJC2_UNAVAILABLE;
    struct objc_method_list **methodLists                   OBJC2_UNAVAILABLE;
    struct objc_cache *cache                                OBJC2_UNAVAILABLE;
    struct objc_protocol_list *protocols                    OBJC2_UNAVAILABLE;
#endif
} OBJC2_UNAVAILABLE;
```

从 objc_class 可以看到，一个运行时类中关联了它的父类指针、类名、成员变量、方法、缓存以及附属的协议。

其中 objc_ivar_list 和 objc_method_list 分别是成员变量列表和方法列表：

```
// 成员变量列表
struct objc_ivar_list {
    int ivar_count                                          OBJC2_UNAVAILABLE;
#ifdef __LP64__
    int space                                               OBJC2_UNAVAILABLE;
#endif
    /* variable length structure */
    struct objc_ivar ivar_list[1]                           OBJC2_UNAVAILABLE;
}                                                           OBJC2_UNAVAILABLE;

// 方法列表
struct objc_method_list {
    struct objc_method_list *obsolete                       OBJC2_UNAVAILABLE;
```

```
    int method_count                                    OBJC2_UNAVAILABLE;
#ifdef __LP64__
    int space                                           OBJC2_UNAVAILABLE;
#endif
    /* variable length structure */
    struct objc_method method_list[1]                   OBJC2_UNAVAILABLE;
}
```

由此可见，我们可以动态修改 *methodList 的值来添加成员方法，这也是 Category 实现的原理，同样解释了 Category 不能添加属性的原因。

objc_ivar_list 结构体用来存储成员变量的列表，而 objc_ivar 则是存储了单个成员变量的信息；同理，objc_method_list 结构体存储着方法数组的列表，而单个方法的信息则由 objc_method 结构体存储。

值得注意的时，objc_class 中也有一个 isa 指针，这说明 Objc 类本身也是一个对象。为了处理类和对象的关系，Runtime 库创建了一种叫做 Meta Class(元类) 的东西，类对象所属的类就叫做元类。Meta Class 表述了类对象本身所具备的元数据。

我们所熟悉的类方法，就源自于 Meta Class。我们可以理解为类方法就是类对象的实例方法。每个类仅有一个类对象，而每个类对象仅有一个与之相关的元类。

当你发出一个类似 [NSObject alloc](类方法) 的消息时，实际上，这个消息被发送给了一个类对象(Class Object)，这个类对象必须是一个元类的实例，而这个元类同时也是一个根元类(Root Meta Class)的实例。所有元类的 isa 指针最终都指向根元类。

所以当 [NSObject alloc] 这条消息发送给类对象的时候，运行时代码 objc_msgSend() 会去它元类中查找能够响应消息的方法实现，如果找到了，就会对这个类对象执行方法调用。

最后 objc_class 中还有一个 objc_cache ，缓存，它的作用很重要，后面会提到。

4.Method
Method 代表类中某个方法的类型

```
typedef struct objc_method *Method;
struct objc_method {
    SEL method_name                                     OBJC2_UNAVAILABLE;
    char *method_types                                  OBJC2_UNAVAILABLE;
    IMP method_imp                                      OBJC2_UNAVAILABLE;
}
```

objc_method 存储了方法名，方法类型和方法实现：

方法名类型为 SEL
方法类型 method_types 是个 char 指针，存储方法的参数类型和返回值类型

method_imp 指向了方法的实现，本质是一个函数指针
Ivar
Ivar 是表示成员变量的类型。

```
typedef struct objc_ivar *Ivar;
struct objc_ivar {
    char *ivar_name                                  OBJC2_UNAVAILABLE;
    char *ivar_type                                 OBJC2_UNAVAILABLE;
    int ivar_offset                                OBJC2_UNAVAILABLE;
#ifdef __LP64__
    int space                                      OBJC2_UNAVAILABLE;
#endif
}
```

其中 ivar_offset 是基地址偏移字节

5.IMP
IMP在objc.h中的定义是：

```
typedef id (*IMP)(id, SEL, ...);
```

它就是一个函数指针，这是由编译器生成的。当你发起一个 ObjC 消息之后，最终它会执行的那段代码，就是由这个函数指针指定的。而 IMP 这个函数指针就指向了这个方法的实现。

如果得到了执行某个实例某个方法的入口，我们就可以绕开消息传递阶段，直接执行方法，这在后面 Cache 中会提到。

你会发现 IMP 指向的方法与 objc_msgSend 函数类型相同，参数都包含 id 和 SEL 类型。每个方法名都对应一个 SEL 类型的方法选择器，而每个实例对象中的 SEL 对应的方法实现肯定是唯一的，通过一组 id和 SEL 参数就能确定唯一的方法实现地址。

而一个确定的方法也只有唯一的一组 id 和 SEL 参数。

6.Cache
Cache 定义如下：

```
typedef struct objc_cache *Cache
struct objc_cache {
    unsigned int mask /* total = mask + 1 */                OBJC2_UNAVAILABLE;
    unsigned int occupied                                  OBJC2_UNAVAILABLE;
    Method buckets[1]                                     OBJC2_UNAVAILABLE;
};
```

Cache 为方法调用的性能进行优化，每当实例对象接收到一个消息时，它不会直接在 isa 指针指向的类的方法列表中遍历查找能够响应的方法，因为每次都要查找效率太低了，而是优先在 Cache 中查找。

Runtime 系统会把被调用的方法存到 Cache 中，如果一个方法被调用，那么它有可能今后还会被调用，下次查找的时候就会效率更高。就像计算机组成原理中 CPU 绕过主存先访问 Cache 一样。

7.Property
typedef struct objc_property *Property;
typedef struct objc_property *objc_property_t;//这个更常用
可以通过class_copyPropertyList 和 protocol_copyPropertyList 方法获取类和协议中的属性：

objc_property_t *class_copyPropertyList(Class cls, unsigned int *outCount)
objc_property_t *protocol_copyPropertyList(Protocol *proto, unsigned int *outCount)
注意：
返回的是属性列表，列表中每个元素都是一个 objc_property_t 指针

```
#import <Foundation/Foundation.h
@interface Person : NSObject
/** 姓名 */
@property (strong, nonatomic) NSString *name;
/** age */
@property (assign, nonatomic) int age;
/** weight */
@property (assign, nonatomic) double weight;
@end
```

以上是一个 Person 类，有3个属性。让我们用上述方法获取类的运行时属性。

```
    unsigned int outCount = 0;

    objc_property_t *properties = class_copyPropertyList([Person class], &outCount);

    NSLog(@"%d", outCount);

    for (NSInteger i = 0; i < outCount; i++) {
        NSString *name = @(property_getName(properties[i]));
        NSString *attributes = @(property_getAttributes(properties[i]));
        NSLog(@"%@--------%@", name, attributes);
    }
```

打印结果如下：

```
test[2321:451525] 3
test[2321:451525] name--------T@"NSString",&,N,V_name
test[2321:451525] age--------Ti,N,V_age
test[2321:451525] weight--------Td,N,V_weight
```

property_getName 用来查找属性的名称，返回 c 字符串。property_getAttributes 函数挖掘属性的真实名称和 @encode 类型，返回 c 字符串。

objc_property_t class_getProperty(Class cls, const char *name)
objc_property_t protocol_getProperty(Protocol *proto, const char *name, BOOL isRequiredProperty, BOOL isInstanceProperty)
class_getProperty 和 protocol_getProperty 通过给出属性名在类和协议中获得属性的引用。

275问:为什么需要Runtime
答:Objective-C 是一门动态语言，它会将一些工作放在代码运行时才处理而并非编译时。也就是说，有很多类和成员变量在我们编译的时是不知道的，而在运行时，我们所编写的代码会转换成完整的确定的代码运行。

因此，编译器是不够的，我们还需要一个运行时系统(Runtime system)来处理编译后的代码。

Runtime 基本是用 C 和汇编写的，由此可见苹果为了动态系统的高效而做出的努力。苹果和 GNU 各自维护一个开源的 Runtime 版本，这两个版本之间都在努力保持一致。

276问:Runtime是什么
答:Runtime 又叫运行时，是一套底层的 C 语言 API，其为 iOS 内部的核心之一，我们平时编写的 OC 代码，底层都是基于它来实现的。比如：

```
[receiver message];
// 底层运行时会被编译器转化为：
objc_msgSend(receiver, selector)
// 如果其还有参数比如：
[receiver message:(id)arg...];
// 底层运行时会被编译器转化为：
objc_msgSend(receiver, selector, arg1, arg2, ...)
```

277问:两个APP之间如何互调传值
答:两个APP之间的跳转是通过[[UIApplication sharedApplication] openURL:url]这种方式来实现的。

1.首先设置第一个APP的url地址
2.接着设置第二个APP的url地址
3.跳转代码：

```
NSString *urlString = [NSString stringWithFormat:@"AppJumpSecond://%@",textField.text];
[[UIApplication sharedApplication] openURL:[NSURL URLWithString:urlString]];
```

例如：将textField的文字也传过去，同样的，在第二个页面也是如此

```
NSString *urlString = [NSString stringWithFormat:@"AppJumpFirst://%@",textField.text];
[[UIApplication sharedApplication] openURL:[NSURL URLWithString:urlString]];
```

4.处理传过去的数据，例如传了textField的数据，接收时在AppDelegate的
- (BOOL)application:(UIApplication *)application openURL:(NSURL *)url sourceApplication:(NSString *)sourceApplication annotation:(id)annotation方法里。
在AppDelegate里设置属性

```
@property (nonatomic, strong) RootViewController *rvc;
```

在didFinishLaunchingWithOptions方法里添加

```
self.rvc = [[RootViewController alloc] init];
UINavigationController *nc = [[UINavigationController alloc]
initWithRootViewController:self.rvc];
self.window.rootViewController = nc;
```

添加代码块
```
- (BOOL)application:(UIApplication *)application openURL:(NSURL *)url sourceApplication:
(NSString *)sourceApplication annotation:(id)annotation {
 self.rvc.textField.text = [[url host]
stringByReplacingPercentEscapesUsingEncoding:NSUTF8StringEncoding];
return YES;
}
```
使得textField显示另一个页面传过来的数据。

278问:如何用objective-c 实现常用算法（冒泡、选择、快速、插入）
答:前置条件：
```
- (void)initArr
{
    for (int i = 0; i < arr.count; i++) {
        for (int j = 0; j < arr.count - i - 1;j++) {
            if ([arr[j+1]integerValue] < [arr[j] integerValue]) {
                int temp = [arr[j] integerValue];
                arr[j] = arr[j + 1];
                arr[j + 1] = [NSNumber numberWithInt:temp];
            }
        }
    }
    NSLog(@"冒泡排序后：%@",arr);
}
```

一.冒泡排序
```
- (void)sort:(NSMutableArray *)arr
{
    for (int i = 0; i < arr.count; i++) {
        for (int j = 0; j < arr.count - i - 1;j++) {
            if ([arr[j+1]integerValue] < [arr[j] integerValue]) {
                int temp = [arr[j] integerValue];
                arr[j] = arr[j + 1];
                arr[j + 1] = [NSNumber numberWithInt:temp];
            }
        }
    }
    NSLog(@"冒泡排序后：%@",arr);
}
```
二.选择排序
```
- (void)sort:(NSMutableArray *)arr
{
    for (int i = 0; i < arr.count; i ++) {
        for (int j = i + 1; j < arr.count; j ++) {
```

```
            if ([arr[i] integerValue]     [arr[j] integerValue]) {
                int temp = [arr[i] integerValue];
                arr[i] = arr[j];
                arr[j] = [NSNumber numberWithInt:temp];
            }
        }
    }

    NSLog(@"选择排序后：%@",arr);
}

三.快速排序
- (void)quickSort:(NSMutableArray *)arr leftIndex:(int)left rightIndex:(int)right
{
    if (left < right) {
        int temp = [self getMiddleIndex:arr leftIndex:left rightIndex:right];
        [self quickSort:arr leftIndex:left rightIndex:temp - 1];
        [self quickSort:arr leftIndex:temp + 1 rightIndex:right];
    }
}

- (int)getMiddleIndex:(NSMutableArray *)arr leftIndex:(int)left rightIndex:(int)right
{
    int tempValue = [arr[left] integerValue];
    while (left < right) {
        while (left < right && tempValue <= [arr[right] integerValue]) {
            right --;
        }
        if (left < right) {
            arr[left] = arr[right];
        }

        while (left < right && [arr[left] integerValue] <= tempValue) {
            left ++;
        }
        if (left < right) {
            arr[right] = arr[left];
        }
    }
    arr[left] = [NSNumber numberWithInt:tempValue];
    return left;
}

四.插入排序
- (void)sort:(NSMutableArray *)arr
```

```
{
    for (int i = 1; i < arr.count; i ++) {
        int temp = [arr[i] integerValue];

        for (int j = i - 1; j     = 0 && temp < [arr[j] integerValue]; j --) {
            arr[j + 1] = arr[j];
             arr[j] = [NSNumber numberWithInt:temp];
        }


    }
    NSLog(@"插入排序后：%@",arr);
}
```

279问:SDWebImage内部实现过程是什么
答: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。

280问:__block和__weak修饰符的区别是什么
答: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。