# 精选 iOS 面试题

**iOS 技术交流群：624212887 （欢迎加入交流，学习）**

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

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

**2.#import 跟 #include 有什么区别， @class 呢，#import<> 跟 #import”” 有什么区别?**

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

**3.属性 readwrite，readonly，assign，retain，copy，nonatomic 各是**

什么作用，在那种情况下用?

①. readwrite 是可读可写特性;需要生成 getter 方法和 setter 方法时

②. 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

③. 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;

④. 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;

⑤. 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。

⑥. nonatomic 非原子操作，决定编译器生成的 setter getter 是否是原子操作，atomic 表示多线程安全，一般使用 nonatomic

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

```
- (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?

编译时是 NSString 的类型;运行时是 NSData 类型的对象

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

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

位还是 64 位来决定是本身是 int 还是 Long。

**7.id** 声明的对象有什么特性?

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

**8.Objective-C** 如何对内存管理的,说说你的看法和解决方法?

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

①.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,在携带的移动设备里面,资源是紧俏商品所以 iPhone 不支持这个功能。

解决:通过 alloc － init 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**?**

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

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

**11.** 浅复制和深复制的区别**?**

答案：浅层复制：只复制指向对象的指针，而不复制引用对象本身。

深层复制：复制引用对象本身。

意思就是说我有个 **A** 对象，复制一份后得到 **A_copy** 对象后，对于浅复制来说，**A** 和 **A_copy** 指向的是同一个内存资源，复制的只不过是

是一个指针，对象本身资源

还是只有一份，那如果我们对 A_copy 执行了修改操作,那么发现 A 引用的对象同样被修改，这其实违背了我们复制拷贝的一个思想。深复制就好理解了,内存中存在了

两份独立对象本身。

用网上一哥们通俗的话将就是：

浅复制好比你和你的影子，你完蛋，你的影子也完蛋

深复制好比你和你的克隆人，你完蛋，你的克隆人还活着。

**12.**类别的作用**?**继承和类别在实现中有何区别**?**

**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。

**13.** 类别和类扩展的区别。

**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**。

**14.** 什么是 **KVO** 和 **KVC?**

**kvc:**键 － 值编码是一种间接访问对象的属性使用字符串来标识属性，而不是通过调用存取方法，直接或通过实例变量访问的机制。很多情况下可以简化程序代码。**apple** 文档其实给了一个很好的例子。

**kvo:**键值观察机制，他提供了观察某一属性变化的方法，极大的简化了代码。

具体用看到嗯哼用到过的一个地方是对于按钮点击变化状态的的监控。

比如我自定义的一个 **button**

```
[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。

15. 我们说的 OC 是动态运行时语言是什么意思?

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。

16.自己写函数，实现 strlen 功能（C 语言第四讲 一维数组与字符数组）

注意：判断的时候使用“\0”

```
#include <stdio.h>
#include <string.h>
int mystrlen(char *a)
{
  int n=0;
while(*a++!='\0')
    n++;
  return n;

}

void main()
{
  int m;
  char a[100];
  printf("please input a string!\n");

  gets(a);
  printf("%d",strlen(a));
  m=mystrlen(a);
```

```
  printf("%d",m);

}
```

17.用变量 a 写出以下定义

a、一个整型数　int a;

b、一个指向整型数的指针　int *a;

c、一个指向指针的指针，它指向的指针是指向一个整型数　int **a;

d、一个有 10 个整型数的数组　int a[10];

e、一个有 10 个指针的数组，该指针是指向一个整型数的　int *a[10];

f、一个指向有 10 个整型数数组的指针　int (*a)[10];

g、一个指向函数的指针，该函数有一个整型参数，并返回一个整型数　int(*a)(int);

**18.**类变量的**@protected ,@private,@public,@package**，声明各有什么含义？（**OC** 第二讲　实例变量可见度与方法）

上面的几个声明表明的时类成员的作用域，**@private** 作用范围只能在自身类（外界既不可访问，又不能继承）；**@protected** 作用范围在自身类和子类，如果什么都不加修饰，默认是**@protected**（外界不可访问，但是可以继承）；**@public** 作用范围最大，可以在任何地方被访问（外界即可访问，又可以继承）；**@package** 作用范围在某个框架内

**19.**用 **Object-C** 写一个冒泡排序

```
NSMutableArray *array = [NSMutableArray arrayWithArray:@[@"3",@"1",@"10",@"5",@"2",@"7",@"12",@"4",@"8"]];

    NSString   *tmp;

    for (int i = 0; i < array.count; i ++) {
```

```
        for (int j = 0; j < array.count   - 1 - i; j++) {

                        if ([[array objectAtIndex:j] integerValue] > [[array
objectAtIndex:j + 1] integerValue]) {

                tmp = [array objectAtIndex:j];

                        [array replaceObjectAtIndex:j withObject:[array
objectAtIndex:j + 1]];

                [array replaceObjectAtIndex:j + 1 withObject:tmp];

            }

        }

    }
```

**20.**简述 **OC** 中内存管理机制。与 **retain** 配对使用的方法是 **dealloc** 还是 **release**，为什么？需要与 **alloc** 配对使用的方法是 **dealloc** 还是

release，为什么？readwrite，readonly，assign，retain，copy，nonatomic 、atomic、strong、weak 属性的作用？（OC 第九讲 内存管理初级）

OC 使用了一种叫做引用计数的机制来管理对象，如果对一个对象使用了 alloc、[Mutable]copy，retain，那么你必须使用相应的 realease 或者 autorelease。也可以理解为自己生成的对象，自己持有。非自己生成的对象，自己也能持有。不在需要自己持有的对象时释放。非自己持有的对象无法释放。生成并持有对象<alloc,new,copy,mutableCopy 等>，持有对象<retain>，释放对象<release>,废弃对象<dealloc>。readwrite(默认):可读可写，表示既有 getter 方法，也有 setter 方法。readonly:表示只有 getter 方法，没有 setter 方法。nonatomic:不考虑线程安全。atomic（默认）：线程操作安全。strong（默认）:ARC 下和 MRC 下 retain 一样，weak（ARC 下）:和(MRC 下)assign 类似，区别是当 weak 指向的内存释放掉后自动置为 nil，防止野指针。

unsafe_unretained 声明一个若引用，但不会自动置为 nil，可能会出现野指针。

线程安全下的 setter 和 getter 方法：

-（NSString *）value{

```
@synchronized（self）{

return [[_value retain] autorelease];

}

}

- (void)setValue:(NSString *)aValue{

@synchronized(self){

[aValue retain];

[_value release];

_value = aValue;

}

}
```

**21.** 简述应用程序按 **Home** 键进入后台时的生命周期，以及从后台回到前台时的生命周期？

应用程序的状态：

**Not running:**未运行，程序没启动

**Inactive** :未激活，程序在前台运行，不过没接受到事件，没有事件处理的状态下通常处于这个状态。

**Active** :激活 程序在前台并且接收到了事件

**Backgound** :后台 程序在后台而且能执行代码，大多数程序进入这个状态后会在在这个状态上停留一会。

**Suspended** :挂起 程序在后台不能执行代码。

**- (BOOL)application:(UIApplication *)application willFinishLaunchingWithOptions:(NSDictionary *)launchOptions**

**//**告诉代理进程启动但还没进入状态保存

**- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions**

**//** 告诉代理启动基本完成程序准备开始运行

**- (void)applicationWillResignActive:(UIApplication *)application**

// 当应用程序将要入非活动状态执行，在此期间，应用程序不接收消息或事件，比如来电话了

**- (void)applicationDidBecomeActive:(UIApplication *)application**

// 当应用程序入活动状态执行，这个刚好跟上面那个方法相反

**- (void)applicationDidEnterBackground:(UIApplication *)application**

// 当程序被推送到后台的时候调用。所以要设置后台继续运行，则在这个函数里面设置即可

**- (void)applicationWillEnterForeground:(UIApplication *)application**

//当程序从后台将要重新回到前台时候调用，这个刚好跟上面的那个方法相反。

**- (void)applicationWillTerminate:(UIApplication *)application**

//当程序将要退出是被调用，通常是用来保存数据和一些退出前的清理工作。这个需要要设置 **UIApplicationExitsOnSuspend** 的键值。

**- (void)applicationDidFinishLaunching:(UIApplication*)application**

//当程序载入后执行

**22.**描述应用程序的启动顺序。

①、程序入口 **main** 函数创建 **UIApplication** 实例和 **UIApplication** 代理实例

②、在 **UIApplication** 代理实例中重写启动方法，设置第一

ViewController

③、在第一 ViewController 中添加控件，实现对应的程序界面。

23.ViewController 的 alloc ， loadView 。 viewDidLoad,viewWillAppear,viewDidUnload,dealloc、init 分别是在什么时候调用的？在自定义 ViewController 的时候这几个函数里面应该做什么工作？（UI 第三讲 自定义视图、视图控制器）alloc 初始化当前的 ViewController

loadView:没有正在使用 nib 视图页面，子类将会创建自己的自定义视图层

viewDidLoad:试图被加载后调用

viewWillAppear：试图即将出现的时候调用

viewDidUnload：<iOS6 之后废弃>当系统内存吃紧的时候会调用该方法,释放掉当前未在 window 中显示的试图和对应的控制器

**24.** 说说响应链

当事件发生的时候，响应链首先被发送给第一个响应者(往往是事件发生的视图，也就是用户触摸屏幕的地方)。事件将沿着响应者链一直向下传递，直到被接受并作出处理。一般来说，第一响应这是个视图对象或者其子类，当其被触摸后事件就交由它处理，如果他不处理，时间就会被传递给视图控制器对象 **UIViewController**（如果存在），然后是它的父视图对象（**superview**），以此类推直到顶层视图。接下来会沿着顶层视图（**topview**）到窗口 **(UIwindow** 对象)再到程序的（**UIApplication** 对象），如果整个过程都没有响应这个事件，则该事件被丢弃，一般情况下，在响应链中只要有对象处理事件，事件就会被传递典型的响应路线图如：**First Responser -> The Window ->The Applicationn -> App Delegate**

**25.** 线程是什么？进程是什么？二者有什么区别和联系？ （**UI** 第二十二讲 多线程编程）

线程是 **CPU** 独立运行和独立调度的基本单位（可以理解为一个进程中执行的代码片段），进程是资源分配的基本单位（进程是一块包含了某些资源的内存区域）。进程是线程的容器，真正完成代码执行的是线程，而进程则作为线程的执行环境。一个程序至少包含一个进程，

一个进程至少包含一个线程，一个进程中的多个线程共享当前进程所拥有的资源。

**26.**谈谈你对多线程开发的理解？**ios** 中有几种实现多线程的方法？（**UI** 第二十二讲 多线程编程）

好处：

①、使用线程可以把程序中占据时间长的任务放到后台去处理，如图片、视频的下载

②、发挥多核处理器的优势，并发执行让系统运行的更快、更流畅，用户体验更好

缺点：

①、大量的线程降低代码的可读性，

②、更多的线程需要更多的内存空间

③、当多个线程对同一个资源出现争夺的时候要注意线程安全的问题。

**iOS** 有三种多线程编程的技术：

①、**NSThread(**两种创建方式**)**

**[NSThread detachNewThreadSelector:@selector(doSomething:) toTarget:self withObject:nil];**

**NSThread *myThread = [[NSThread alloc] initWithTarget:self selector:@selector(doSomething:) object:nil];**

**[myThread start];**

**②、NSOperationQueue**

**NSOperationQueue *oprationQueue = [[NSOperationQueue alloc] init];**

**oprationQueue addOperationWithBlock:^{**

**//这个 block 语句块在子线程中执行**

**}**

**http://alloc.sinaapp.com/wp/?p=237**

**③、Grand Central Dispatch (GCD)**

**dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORI**

**TY_DEFAULT, 0), ^{**

**// 耗时的操作**

**dispatch_async(dispatch_get_main_queue(), ^{**

```
        // 更新界面

    });

});
```

http://blog.csdn.net/totogo2010/article/details/8016129

PS:不显示的创建线程的方法：

用 NSObject 的类方法 performSelectorInBackground:withObject:

创建一个线程：

```
[Obj performSelectorInBackground:@selector(doSomething)
withObject:nil];
```

27.线程同步和异步的区别？IOS 中如何实现多线程的同步？

同步：一个线程要等待上一个线程执行完之后才能执行当前的线程，

生活中的例子（上厕所）。

异步：同时去做两件或者多件事。比如边听歌边看报。

**28. RunLoop 是什么？**

**RunLoop** 就是一个事件处理的循环，用来不停的调度工作以及处理输入事件。使用 **RunLoop** 的目的是让你的线程在有工作的时候忙于工作,而没工作的时候处于休眠状态。**runloop** 的设计是为了减少 **cpu** 无谓的空转。

使用场景：**1**、需要使用 **Port** 或者自定义 **Input Source** 与其他线程进行通讯；**2**、子线程中使用了定时器；**3**、**Cocoa** 中使用任何 **performSelector** 到了线程中运行方法；**4**、线程执行周期性任务。仅当在为你的程序创建辅助线程的时候,你才需要显式运行一个 **RunLoop**。

**29.**简述你对 **UIView**、**UIWindow** 和 **CALayer** 的理解

**UIView** 继承于 **UIResponder, UIResponder** 继承于 **NSObject,UIView** 可以响应用户事件。**CALayer** 继承于 **NSObject**，所以 **CALayer** 不能响应

事件。

**UIView** 构建界面,**UIView** 侧重于对内容的管理，**CALayer** 侧重于对内容的绘制。

**UIView** 是用来显示内容的，可以处理用户事件；**CALayer** 是用来绘制内容的，对内容进行动画处理，依赖与 **UIView** 来进行显示，不能处理用户事件。

**30.** 为什么很多内置类如 **UITableViewController** 的 **delegate** 属性都是 **assign** 而不是 **retain**？请举例说明。

防止循环引用，

```
Student * str=[];

Teacher *teacher=[[Teacher alloc] init];

Student * student=[[Student alloc] init];

teacher.delegate=student;

student.delegate= teacher;
```

在 teacher 中 dealloc 会 release 当前的 Delegate，就会触发 student 对象 release，继而也会导致 student 执行 dealloc，在 student 中也会 release 自己的 delegate，产生循环了。

**31. iOS 的动态性**

iOS 的动态性来自三个方面：动态类型、动态绑定、动态载入、SEL 类型

①、动态类型<弱类型>（id）：在代码的运行阶段判断代码的类型，使用 id 类型可以让应用在“运行时”使用任何类型来替换。动态类型让程序更加灵活，但是会使数据的统一性降低和代码的可读性。我们常用静态类型<强类型>（如 NSString）,使用静态类型编译器可以完全分析你的代码，这让代码的性能和可预知性更高。

②、动态绑定：让代码在运行时判断需要调用什么方法，而不是在编译时。 动态类型和动态绑定使得选择哪个接收者已经调用什么方法都放到运行时去完成。

③、动态载入：应用程序可以根据需要加载可执行代码以及资源，而不是在启动时就加载所有资源。

④、SEL 类型 iOS 在编译的时候会根据方法的名字（包括参数序列），生成一个用来区分这个方法的唯一的 ID，这个 ID 是 SEL 类型的，SEL 的本质就是类方法的编号[函数地址]。（类似 C 语言里面的函数指针，但是 OC 的类不能直接使用函数指针，这样只能做一个@selector 语法来取。注意：@selector 是查找当前类（含子类）的方法。）

**32. What is lazy loading?**

在使用的时候才去初始化，比如 UITableViewCell 的 imageView 属性，懒加载对象的创建是在 getter 方法里面进行创建的。

**33.** 谈谈对性能优化的看法，如何做？

从用户体验出发：1、程序 logging 不要太长、2、相同数据不做重复获取、3、昂贵资源要重用（cell、sqlite、date），4、良好的编程习惯和程序设计：选择正确的集合对象和算法来进行编程、选择适合的数据存储格式（plist、SQLite）、优化 SQLite 查询语句 5、数据资源方面

的优化（缓存和异步加载）

解决方案：

能够发现问题

利用 **log** 或工具分析问题原因

假设问题原因

改进代码和设计

**http://blog.csdn.net/yangxt/article/details/8173412**

**34. isKindOfClass、isMemberOfClass** 作用分别是什么？

**-(BOOL) isKindOfClass: classObj** 判断是否是这个类或者是这个类子类的实例

**-(BOOL) isMemberOfClass: classObj** 判断是否是这个类的实例

**35. NSAutoreleasePool 是怎么工作的?**

自动释放池以栈的形式实现:当你创建一个新的自动释放池时，它将被添加到栈顶。当一个对象收到发送 **autorelease** 消息时,它被添加到当前线程的处于栈顶的自动释放池中,当自动释放池被回收时,它们从栈中被删除，并且会给池子里面所有的对象都会做一次 **release** 操作.

**36.** 类工厂方法是什么

类工厂方法的实现是为了向客户提供方便，它们将分配和初始化合在一个步骤中， 返回被创建的对象，并
进行自动释放处理。这些方法的形式是**+ (type)classNam**e...（其中 **className** 不包括任何前缀）。

工厂方法可能不仅仅为了方便使用。它们不但可以将分配和初始化合在一起，还可以 为初始化过程提供对
象的分配信息。

类工厂方法的另一个目的是使类（比如 **NSWorkspace**）提供单件实例。

虽 然 init...方法可以确认一

个类在每次程序运行过程只存在一个实例，但它需要首先分配一个“生的”实例，然后还必须释放该实例。

工厂 方法则可以避免为可能没有用的对象盲目分配内存。

**37. block 在 ARC 中和 MRC 中的用法有什么区别,需要注意什么**

①.对于没有引用外部变量的 **Block**，无论在 **ARC** 还是非 **ARC** 下，类型都是**__NSGlobalBlock__**，这种类型的 **block** 可以理解成一种全局的 **block**，不需要考虑作用域问题。同时，对他进行 **Copy** 或者 **Retain** 操作也是无效的

②.应注意避免循环引用

**38.** 网络图片处理问题中怎么解决一个相同的网络地址重复请求的问题？

利用字典（图片地址为 **key**，下载操作为 **value**）

**39.** 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（提示：可以从两者的实现机制和适用范围来描述）。

**1> GCD** 是纯 **C** 语言的 **API**，**NSOperationQueue** 是基于 **GCD** 的 **OC** 版本封装

**2>GCD** 只支持 **FIFO** 的队列，**NSOperationQueue** 可以很方便地调整执行顺序、设置最大并发数量

**3>NSOperationQueue** 可以在轻松在 **Operation** 间设置依赖关系，而 **GCD** 需要写很多的代码才能实现

**4>NSOperationQueue** 支持 **KVO**，可以监测 **operation** 是否正在执行（**isExecuted**）、是否结束（**isFinished**），是否取消（**isCanceld**）

**5> GCD** 的执行速度比 **NSOperationQueue** 快

任务之间不太互相依赖：**GCD**

任务之间有依赖\或者要监听任务的执行情况：NSOperationQueue

40.在异步线程中下载很多图片,如果失败了,该如何处理?请结合RunLoop来谈谈解决方案.(提示:在异步线程中启动一个RunLoop重新发送网络请求,下载图片)

1> 重新下载图片

2>下载完毕，利用 RunLoop 的输入源回到主线程刷新 UIImageVIUew

41.Socket 的实现原理及 Socket 之间是如何通信的

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）；

本地的进程间通信（IPC）有很多种方式，但可以总结为下面 4 类：

消息传递（管道、FIFO、消息队列）
同步（互斥量、条件变量、读写锁、文件和写记录锁、信号量）
共享内存（匿名的和具名的）
远程过程调用（Solaris 门和 Sun RPC）

**42.**描述一下复用机制

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。

**43.**代理和协议什么区别

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.**iOS** 开发和 **Objective-c** 区别

**44.OC** 里怎么实现多继承

**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

**45.POST** 和 **GET** 有何区别

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。

46.copy 和 mutableCopy 区别

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 。与之类似 NSString ->NSMutableString, NSArray->NSMutableArray, NSDictionary->NSMutableDictionary, NSData->NSMutableData。

47.我们说的 OC 是动态运行时语言是什么意思?

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?

**48.**在项目什么时候选择使用 GCD，什么时候选择 NSOperation?

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。

**49.runloop** 和线程有什么关系？

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：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。

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。

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];
```

**50.**在 block 内如何修改 block 外部变量？

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
```

**51. GCD 的队列（dispatch_queue_t）分哪两种类型？**

串行队列 **Serial Dispatch Queue**

并行队列 **Concurrent Dispatch Queue**

**52. lldb（gdb）常用的调试命令？**

**1. breakpoint** 设置断点定位到某一个函数

**2. n** 断点指针下一步

**3. po** 打印对象

**53.** 使用 **block** 时什么情况会发生引用循环，如何解决？

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;**

**54.** 简述 **NotificationCenter**、**KVC**、**KVO**、**Delegate**？并说明它们之间的区别？

**Notification**：观察者模式，**controller** 向 **defaultNotificationCenter** 添加自己的 **notification**，其他类注册这个 **notification** 就可以收到通知，这些类可以在收到通知时做自己的操作（多观察者默认随机顺序发通知给 观察者们，而且每个观察者都要等当前的某个观察者的操作做完才能轮到他来操作，可以用 **NotificationQueue** 的方式安排观察者的反应顺序，也 可以在添加观察者中设定反映时间，取消观察需要在 **viewDidUnload** 跟 **dealloc** 中都要注销）。

**KVC** 键值编码，可以直接通过字符串的名字（**key**）来间接访问属性的机制，而不是通过调用 **getter** 和 **setter** 方法访问。

**KVO**：观测指定对象的属性，当指定对象的属性更改之后会通知相应的观察者。

**delegate**:一对一，**delegate** 遵循某个协议并实现协议声明的方法。

**55. iOS 本地数据存储都有哪几种方式？iOS 如何实现复杂对象的存储？**

**1. NSKeyedArchiver**（归档）采用归档的形式来保存数据，该数据对象需要遵守 **NSCoding** 协议，并且该对象对应的类必须提供 **encodeWithCoder:**和 **initWithCoder:**方法。**2. NSUserDefaults**：用来保存应用程序设置和属性、用户保存的数据。用户再次打开程序或开机后这些数据仍然存在。**NSUserDefaults** 可以存储的数据类型包括：**NSData、NSString、NSNumber、NSDate、NSArray、NSDictionary**。

**3. Write** 写入方式：永久保存在磁盘中。

**4. SQLite（FMDB、CoreData）**

**NSCoding + NSKeyedArchiver** 实现复杂对象的存储。

**56.为什么很多内置类如 UITableViewController 的 delegate 属性都是 assign 而不是 retain？请举例说明。**

防止循环引用，

```
Student * str=[];

Teacher *teacher=[[Teacher alloc] init];

Student * student=[[Student alloc] init];

teacher.delegate=student;

student.delegate= teacher;
```

在 teacher 中 dealloc 会 release 当前的 Delegate，就会触发 student 对象 release，继而也会导致 student 执行 dealloc，在 student 中也会 release 自己的 delegate，产生循环了。

**57.** 如何监测系统键盘的弹出

**[[NSNotificationCenter defaultCenter] addObserver:self selector:@selector( ) name:UIKeyboardWillShowNotification object:nil];**

**58. RunLoop** 是什么？

**RunLoop** 就是一个事件处理的循环，用来不停的调度工作以及处理输入事件。使用 **RunLoop** 的目的是让你的线程在有工作的时候忙于工作,而没工作的时候处于休眠状态。**runloop** 的设计是为了减少 **cpu** 无谓的空转。

使用场景：**1**、需要使用 **Port** 或者自定义 **Input Source** 与其他线程进行通讯；**2**、子线程中使用了定时器；**3**、**Cocoa** 中使用任何

performSelector 到了线程中运行方法；4、线程执行周期性任务。仅当在为你的程序创建辅助线程的时候,你才需要显式运行一个 RunLoop。

59. 说说你在项目中使用 git 的.

可视化工具(sourcetree)

服务器上创建仓库 -> clone -> 修改 -> commit -> pull -> push

git 命令行 cd 将你准备要上传的文件夹拖拽到这里，自动生成路径 -> git init -> git add .添加所有文件(只是在本地准备好，注意那个“.”表示所有文件) -> git commit -m 'version 1.2.6'引号中是说明信息 -> git remote add origin git@bitbucket.org:**********git ->git push origin master 上传步骤到此完成

**60.**使用圆角效果时为什么会出现卡顿效果,如何解决这个问题**?**

不要在滚动视图使用 **cornerRadius** 或者 **mask**，添加 **self.layer.shouldRasterize = YES; self.layer.rasterizationScale = [UIScreen mainScreen].scale;**

采取预先生成圆角图片，并缓存起来这个方法才是比较好的手段。预处理圆角图片可以在后台处理，处理完毕后缓存起来，再在主线程显示，这就避免了不必要的离屏渲染了。

**61.**简述可视化和纯代码的优缺点**.**

可视化: 优点:快,易操作.缺点:团队合作时容易出现 **xib** 代码冲突问题**.**

纯代码: 优点:团队合作时不易出问题,而且代码逻辑清楚,扩展性强.缺点:编写速度没有可视化快速**.**

**62. UITableView** 如何优化.**(**可查看下面地址中内容,然后记住**)**

**http://longxdragon.github.io/2015/05/26/UITableView%E4%BC%98%E5%8C%96%E6%8A%80%E5%B7%A7/**

提前计算并缓存好高度（布局），因为 **heightForRowAtIndexPath:**是调用最频繁的方法；

异步绘制，遇到复杂界面，遇到性能瓶颈时，可能就是突破口；

滑动时按需加载，这个在大量图片展示，网络加载的时候很管用！（**SDWebImage** 已经实现异步加载，配合这条性能杠杠的）。

关注 **iOS** 开发：**iOSDevTip**

除了上面最主要的三个方面外，还有很多几乎大伙都很熟知的优化点：

正确使用 **reuseIdentifier** 来重用 **Cells**

尽量使所有的 **view opaque**，包括 **Cell** 自身

尽量少用或不用透明图层

如果 **Cell** 内现实的内容来自 **web**，使用异步加载，缓存请求结果

减少 **subviews** 的数量

在 **heightForRowAtIndexPath:**中尽量不使用 **cellForRowAtIndexPath:**，如果你需要用到它，只用一次然后缓存结果

尽量少用 **addView** 给 **Cell** 动态添加 **View**，可以初始化时就添加，然后通过 **hide** 来控制是否显示

**63.**你做过哪些项目？项目中有何亮点技术点？

应用介绍：如做的 **XXX** 项目用户量达到 **XX** 万。

**1**、应用中的功能丰富（如图文混排、定位、支付、云等等）

**2**、逻辑复杂（使用设计模式，多线程）

3、数据库设计，预留冗余等。复杂交互（如使用了 html，JS 等，混编实现 XX 功能）

4、屏幕适配（此段不必提第三方）

64.有没有在 AppStore 上架产品？简单描述上架过程。

有。上架过程 1.登录 developer.apple.com。2.在 member center 中，首先创建 App ID。3.创建配置文件，选择(App Store 或 Ad Hoc)。4.证书和配置文件以及 bundle id 就创建好了。在本地项目中做出相应设置。5.设置 Build Settings，debug 对应的是开发证书和开发的配置文件，release 对应的是发布证书和发布配置文件，development 对应的是测试环境，production 对应的是生产环境。请大家根据实际情况选择对应的证书和配置文件。6.修改 bundle ID （与之前在 developer 上创建的 bundle ID 保持一致）和 bundle name（app 的名字）。7.打包之前，先到 https://itunesconnect.apple.com 创建应用的名字填入相应的应用信息，应用截图，基本应用的资料 以及手动发行和自动发行版本等等。8.archive 打包 ，根据需求（上传或测试）选择证书、配置文件和 scheme。9.选择 submit 后会弹出一个选择账户的提示框，

选择相应的开发者账户就好，下一步至上传完成。

以下问题答案多样自己度娘!

**65.**项目上架遇到过什么问题？被拒原因？

**66.**项目开发过程中有没有遇到什么问题？如何解决？

**67.**写过哪些技术文档？

**68.**有没有做过支付？需要注意哪些问题？

**69.**有没有做过地图开发？需要注意哪些问题？

**70.**简单描述一下推送的过程。本地推送和远程推送有什么区别？

**71.**对 **iOS9** 有了解么？

**72.**有没有做适配？怎么做的？

**73.Swift** 有几个版本？**swift2.1** 与之前有什么变化？

**74.**写一下 **swift** 代理的实现。

**75.**写一下 **swift** 的类，带有存储属性和计算属性。

**76.**讲述一次在这个 **APP** 中，用户触发了一个事件，引起了一个服务请求，然后获取服务端返回，并且更新前端界面的过程。请说的详细一点，比如数据经过了哪些类的处理，每一次传递时的格式是怎么样的？

**77.** 你参与的 **APP**，是如何处理多个服务的同步发起的？

使用 **iOS** 线程技术，创建并行队列来分别执行不同的任务。将不同的任务加入到子队列当中，任务执行后回到主线程当中刷新 **UI** 界面。

**APP** 界面一般都是根据产品需求和 **UI** 效果图来完成的，但是我们为了提高代码的复用性，会将视图的实现进行封装，然后在控制器当中进行加载调用。

**78.**数据持久化存储方案有哪些？

**1）plist** 文件**(**属性列表**)**

**2）preference(**偏好设置**)**

**3) NSKeyedArchiver(**归档**)**

**4）SQlite**

**5) CoreData**

**79.**沙盒的目录结构是怎样的？各自一般用于什么场合？

**Documents** :最常用的目录，**iTunes** 同步该应用时会同步此文件夹中的内容，适合存储重要数据。

**Library\Caches: iTunes** 不会同步此文件夹，适合存储体积大，不需要

备份的非重要数据

**Preferences:iTunes** 同步该应用时会同步此文件夹中的内容，通常保存应用的设置信息

**tmp :iTunes** 不会同步此文件夹，系统可能在应用没运行时就删除该目录下的文件，所以此目录适合保存应用中的一些临时文件，用完就删除。

**80.SQL** 语句问题：**inner join**、**left join**、**right join** 的区别是什么？

**left/ right join** 是外部链接，**inner join** 是内连接

外部链接有主表和从表，主表在 **left** 中是左侧表，**right** 中是右侧表，主表数据会全部显示，从表数据只显示关联部分匹配数据，无匹配的数据用 **null** 补全。内连接则只显示两表关联条件匹配的数据

注:所谓关联条件即是指 **on** 的条件

**81.sqlite** 的优化。

数据库在使用的时候一定要与多线程相结合的。项目中用到了数据库，假如其中一个表有 33 万条数据。对这个表进行查询，执行类似“SELECT * FROM table1 where ds_key like 'asd%%'”这样的语句，在 Touch 上用了大概 11 秒，体验很差。

调查发现，虽然已经对 table1 增加了索引，但是在实际执行的时候，如果查询条件包含 LIKE，就不会使用索引。

想知道查询是不是用了索引，可以参考 http://www.sqlite.org/eqp.html 这个链接提供的方法进行验证。

而改成“SELECT * FROM table1 where ds_key >= 'asd' and ds_key < 'ase'”这样的语句，查询时就会使用到索引，花费时间只有零点几秒，效率提高了几十倍。

当然，这只针对查询以某个字符串开头的记录的情况，而对于 like '%%asd%%'这样的就没法优化了。

82.网络通信用过哪些方式（100%的人说了 AFNetworking...）如何处

理多个网络请求并发的情况

**1)**、直接使用 **socket** 的方式

**2)**、**AFNetworking**

使用多线程中的并发队列去处理网络的并发请求。

**83.**在网络请求中如何提高性能

在网络请求中如何保证安全性

**iOS** 在数据传输过程中有数据泄露的可能，客户端在进行网络请求时一般都采用加密的方式来保证数据的安全性，在 **iOS9** 以后，苹果为了保护数据的安全性，使用了 **Https** 协议。

**84.**内存中的栈和堆的区别是什么？

管理方式：对于栈来讲，是由编译器自动管理的，无需我们手动控制，对于堆来讲，释放工作有程序猿控制，这样就容易产生 memory Leak

申请大小：栈是向低地址扩展的数据结构，是一块连续的内存区域。这句话的意思是栈顶上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 1M，总之是编译器确定的一个常数），如果申请的空间超过了栈的剩余空间时候，就 overflow。因此，能获得栈的空间较小。

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。堆的大笑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快从栈中弹出

分配方式:

堆都是动态分配的，没有静态分配的堆。栈有两种分配方式：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。动态分配是有 alloc 函数进行分配的，但是栈的动态分配和堆是不同的，他的动态分配由编译器进行释放，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，计算机会在底层堆栈提供支持，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，他的机制是很复杂的。

**85.**那些数据在栈上，哪些在堆上？

栈：在函数调用时，第一个进栈的是主函数中后的下一条指令（函数调用语句的下一条可执行语句）的地址，然后是函数的各个参数，在大多数的 C 编译器中，参数是由右往左入栈的，然后是函数中的局部变量。注意静态变量是不入栈的。当本次函数调用结束后，局部变量先出栈，然后是参数，最后栈顶指针指向最开始存的地址，也就是主

函数中的下一条指令，程序由该点继续运行。

堆：一般是在堆的头部用一个字节存放堆的大小。堆中的具体内容由程序员安排。

**86.#define** 和 **const** 定义的变量，有什么区别什么情况下会出现内存的循环引用 **block** 中的 **weak self**，是任何时候都需要加的么？

**1.** 宏：只是在预处理器里进行文本替换，没有类型，不做任何类型检查，编译器可以对相同的字符串进行优化。只保存一份到 **.rodata** 段。甚至有相同后缀的字符串也可以优化，你可以用 **GCC** 编译测试，**"Hello world"** 与 **"world"** 两个字符串，只存储前面一个。取的时候只需要给前面和中间的地址，如果是整形、浮点型会有多份拷贝，但这些数写在指令中。占的只是代码段而已，大量用宏会导致二进制文件变大.常量：共享一块内存空间，就算项目中 **N** 处用到，也不会分配 **N** 块内存空间，可以根据 **const** 修饰的位置设定能否修改，在编译阶段会执行类型检查

**2.** （**1**）计时器 **NSTimer**

一方面，NSTimer 经常会被作为某个类的成员变量，而 NSTimer 初始化时要指定 self 为 target，容易造成循环引用。 另一方面，若 timer 一直处于 validate 的状态，则其引用计数将始终大于 0.

（2）block

block 在 copy 时都会对 block 内部用到的对象进行强引用(ARC)或者 retainCount 增 1(非 ARC)。在 ARC 与非 ARC 环境下对 block 使用不当都会引起循环引用问题，一般表现为，某个类将 block 作为自己的属性变量，然后该类在 block 的方法体里面又使用了该类本身，简单说就是 self.someBlock = ^(Type var){[self dosomething]; 或者 self.otherVar = XXX;或者_otherVar = ...};block 的这种循环引用会被编译器捕捉到并及时提醒。

（3）委托 delegate

在委托问题上出现循环引用问题已经是老生常谈了，本文也不再细讲，规避该问题的杀手锏也是简单到哭，一字诀：声明 delegate 时请用 assign(MRC)或者 weak(ARC)，千万别手贱玩一下 retain 或者 strong，毕竟这基本逃不掉循环引用了！

3. 使用方将 self 或成员变量加入 block 之前要先将 self 变为__weak

**87.GCD 的 queue，main queue 中执行的代码，一定是在 main thread 么？**

**dispatch_async(dispatch_get_main_queue(), ^{ });**
**dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{ });**

**88.NSOperationQueue 有哪些使用方式？**

一种在 iOS 中执行并发操作的方法，是使用 NSOperation 和 NSOperationQueue 类。在本教程中，你将学习如何使用它们！你会先创建一款不使用多线程的 app，这样它会变得响应非常迟钝。然后改进程序，添加上并行操作－并且希望－可以提供一个交互响应更好的界面给用户！

另一种处理操作之间的依赖关系，如果操作直接有依赖关系，比如第二个操作必须等第一个操作结束后再执行。控制线程池中的线程数，具体参考 **http://blog.sina.com.cn/s/blog_45e2b66c0100ztz7.html**

**89.NSThread 中的 Runloop 的作用，如何使用？**

有些情况下，我们还是在运行一些长线任务或者复杂任务的时候需要用比较原始的 NSThread。这就需要为 NSThread 创建一个 run loop.

```
1
NSThread *thread1 = [[NSThread alloc] initWithTarget:self selector:@s
elector(playerThread: ) object:nil];

2 [thread start];

3 //如果要利用 NSOperation，原理类似。只需要加入到 queue 里面去就好了。。queue 会在合适的时机调用方法，下面代码作为参考。

4 - (void) playerThread: (void*)unused

5 {

6 audioRunLoop = CFRunLoopGetCurrent();//子线程的 runloop 引用

7 NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];//子线程的
```

**8 [self initPlayer]; CFRunLoopRun(); //运行子线程的**

**9 [pool release]; // run loop,这里就会停住了。**

**10 }**

**11 // 实现一个 timer,用于检查子线程的工作状态，并在合适的时候做任务切换。或者是合适的时候停掉自己的**

**12 -(void) initPlayer {**

**13 // 在这里你可以初始化一个工作类，比如声音或者视频播放**

**14 NSTimer *stateChange = [NSTimer scheduledTimerWithTimeInterval:0.5 target:self selector:**

**15 @selector(checkStatuserInfo:nil repeats:YES];**

**16 }**

```
17  -(void) checkState:(NSTimer*) timer

18  {

19  if(需要退出自线程了) {

20  //释放子线程里面的资源

21  CFRunLoopStop( CFRunLoopGetCurrent());//结束子线程任务

22  }

23 }
```

**90.h** 文件中的成员变量，外部可以直接访问么？（注意是变量，不是 **property**）

成员变量，有三种权限，就是大家都知道的**@private**、**@protected**、**@public** ，写在**.m** 文件中时，相当于是**@private** 权限，子类无法访问，验证了一下，做权限修改也无效。而写在**.h** 文件中，默认是

@protected 权限，子类可以访问，可以做权限修改。因为访问权限指针对.h 文件。.h 文件中成员变量，外部类对其的调用，跟 C++一样，用->来调用。

91.讲述一下 runtime 的概念，message send 如果寻找不到相应的对象，会如何进行后续处理 ？

简单来说，Objective-C runtime 是一个实现 Objective-C 语言的 C 库。对象可以用 C 语言中的结构体表示，而方法（methods）可以用 C 函数实现。事实上，他们 差不多也是这么干了，另外再加上了一些额外的特性。这些结构体和函数被 runtime 函数封装后，Objective-C 程序员可以在程序运行时创建，检 查，修改类，对象和它们的方法。

runtime 不需要知道 Foundation。runtime 会让程序定义转发函数（forwarding function），当 objc_msgSend()无法找到该 selector 的实现时，那个转发函数就会被调用。程序一启动，CoreFoundation 就将-forwardInvocation:定义成转发函数。

参考网站 http://mobile.51cto.com/hot-434685.htm

**92.TCP 和 UDP 的区别是什么？**

**TCP** 面向连接、传输可靠**(**保证数据正确性**,**保证数据顺序**)**、用于传输大量数据**(**流模式**)**、速度慢，建立连接需要开销较多**(**时间，系统资源**)**。

**UDP** 面向非连接、传输不可靠、用于传输少量数据**(**数据包模式**)**、速度快。

**93.MD5 和 Base64 的区别是什么，各自场景是什么？**

**MD5:** 全称为 **message digest algorithm 5(**信息摘要算法**),** 可以进行加密，但是不能解密，属于单向加密，通常用于文件校验

**Base64:** 把任意序列的 **8** 为字节描述为一种不易为人识别的形式，通常用于邮件、**http** 加密．登陆的用户名和密码字段通过它加密，可以进行加密和解密.

**94.**二叉搜索树的概念，时间复杂度多少？

二叉查找树（**Binary Search Tree**），也称有序二叉树（**ordered binary tree**）,排序二叉树（**sorted binary tree**），是指一棵空树或者具有下列性质的二叉树：

**1.** 若任意节点的左子树不空，则左子树上所有结点的值均小于它的根结点的值；

**2.** 若任意节点的右子树不空，则右子树上所有结点的值均大于它的根结点的值；

**3.** 任意节点的左、右子树也分别为二叉查找树。

**4.** 没有键值相等的节点（**no duplicate nodes**）。

它和二分查找一样，插入和查找的时间复杂度均为 **lgN**，但是在最坏的情况下仍然会有 **N** 的时间复杂度。原因在于插入和删除元素的时候，树没有保持平衡

**95.**哪些类不适合使用单例模式？即使他们在周期中只会出现一次。

工具类,不需要存储数据的.

**96.**如何添加一个自定义字体到工程中

添加 **ttf** 文件.

通过**[UIFont fontWithName:@"**字体名**" size:20.0];**使用

**97.**如何制作一个静态库**/**动态库，他们的区别是什么？

使用静态库的好处

**1**，模块化，分工合作

**2**，避免少量改动经常导致大量的重复编译连接

**3**，也可以重用，注意不是共享使用

动态库使用有如下好处：

1 使用动态库，可以将最终可执行文件体积缩小

2 使用动态库，多个应用程序共享内存中得同一份库文件，节省资源

3 使用动态库，可以不重新编译连接可执行程序的前提下，更新动态库文件达到更新应用程序的目的。

98.Configuration 中，debug 和 release 的区别是什么？

Release 是发行版本,比 Debug 版本有一些优化，文件比 Debug 文件小 Debug 是调试版本，Debug 和 Release 调用两个不同的底层库。

一、"Debug 是调试版本，包括的程序信息更多"

二、只有 DEBUG 版的程序才能设置断点、单步执行、使用 TRACE/ASSERT 等调试输出语句。

三、REALEASE 不包含任何调试信息，所以体积小、运行速度快。

**99.**介绍一下响应者链以及它的工作流程。

**OS** 系统检测到手指触摸**(Touch)**操作时会将其打包成一个 **UIEvent** 对象，并放入当前活动 **Application** 的事件队列，单例的 **UIApplication** 会从事件队列中取出触摸事件并传递给单例的 **UIWindow** 来处理，**UIWindow** 对象首先会使用 **hitTest:withEvent:**方法寻找此次 **Touch** 操作初始点所在的视图**(View)**，即需要将触摸事件传递给其处理的视图，这个过程称之为 **hit-test view**。

响应者对象（**Responder Object**）： 指的是 有响应和处理事件能力的对象。 响应者链就是由一系列的响应者对象 构成的一个层次结构。

**UIAppliction --> UIWiondw -->**递归找到最适合处理事件的控件**-->**控件调用 **touches** 方法**-->**判断是否实现 **touches** 方法**-->**没有实现默认会将事件传递给上一个响应者**-->**找到上一个响应者

**UIResponder** 是所有响应对象的基类，在 **UIResponder** 类中定义了处理上述各种事件的接口。我们熟悉的 **UIApplication** 、

UIViewController、 UIWindow 和所有继承自 UIView 的 UIKit 类都直接或间接的继承自 UIResponder，所以它们的实例都是可以构成响应者链的响应者对象

100.发送 10 个网络请求，然后再接收到所有回应之后执行后续操作，如何实现？

开辟一个子线程,然后创建 10 个并行队列

实现一个第三方控件，可以在任何时候出现在 APP 界面最上层

实现一个最简单的点击拖拽功能。上面那个拖拽之外，如果在手放开时，需要根据速度往前滑动呢？

101.如何减小一个应用程序占用存储空间？

检查程序 去掉多余的 xib。iOS App Store 相关因素作为提交到 App Store 中 app 里的可执行文件是被加过密的。加密的副作用是可执行文件的压缩效果没有之前的好了。Build Settings 编译选项，将 build setting 中的 Optimization Level 设置为 Fastest, Smallest [-Os]; 将 build setting 中的 Strip Debug Symbols During Copy 设置为 YES(COPY_PHASE_STRIP = YES)，这样可以减小编译出二进制文件的尺

寸。Target 针对较少的 CPUs 对程序指定的特定 CPU 类型做优化处理，以生成相对于的可执行文件。不同的硬件，将运行不同的可执行代码。虽然这样优化后的程序，只能针对某些设备运行，但是这大大减小可执行程序的大小。要想只设定特定类型的 CPUs，可以修改 build setting 中的 Architectures，将其从 Standard $(ARCHS_STANDARD)修改为你希望支持的列表中对应的特定类型 CPU。有效的 CPU 名称列在 Valid Architectures (VALID_ARCHS) build setting 中。请不要修改 Valid Architectures 设置项，最好由 Xcode 管理。尽量使用 8-bit 图片。使用 8-bit 的 PNG 图片，比 32-bit 的图片能减少 4 倍的压缩率。由于 8-bit 的图片支持最多 256 种不同的颜色，所以 8-bit 的图片一般只应该用于一小部分的颜色图片。例如灰度图片最好使用 8-bit。

**102.**如何提高一个应用程序的性能？

1、使用 ARC 减少内存失误，dealloc 需要重写并对属性置 nil。2、重用。3、尽量少使用透明或半透明。会产生额外的运算。4、少用运算获得圆角，不论 view.maskToBounds 还是 layer.clipToBounds 都会有很大资源开销，必须要用圆角的话不如图片本身就做成圆角。5、不要阻塞主线程。6、使用正确的容器类型。7、图片与 imageView 相同大小避免多余运算。8、使用懒加载。9、使用绘制。

**103.**不同版本的 **APP**，数据库结构变化了，如何处理**?**（数据库迁移）

数据库迁移问题，一般项目中使用数据框架有两种 **Sqlite** 和 **CoreData**。

在 **Sqlite** 中有 **Alter** 命令修改数据库的结构和库名。

**//**对于老用户，数据库已经存在，需要修改

**1.**查询旧表 **NSString *searchSql = [NSString stringWithFormat:@"select sql from sqlite_master where tbl_name='**表名**' and type='table'"];**

**2.**若找到旧表，判断是否存储记录，添加新字段或改名

**NSString *sql_add = "ALTER TABLE** 表名 **ADD** 字段名 字段类型”；

或者修改表名

**ALTER TABLE** 原表名 **RENAME TO** 新表名**;**

**3.** 执 行 改 **sqlite3_exec(** 数 据 库 **, sql_add, NULL, NULL, NULL)!=SQLITE_OK**。

**4.**释放伴随指针。

**104.**同一个工程中 **ARC** 和非 **ARC** 的如何混合使用？

点击项目**->TARGETS->Build Phases->Compile Sources** 中选择要改的**.m**，双击，在标签中添加：

**1.**如果是 **ARC** 项目，要加入非 **ARC** 的代码文件：**-fobjc-arc**

**2.**如果是非 **ARC**，要加入 **ARC** 的代码： **-fno-objc-arc**

回车