Table of Contents

[网络上的面试题： 2](#_Toc34821356)

[UIView和CAlayer的区别 2](#_Toc34821357)

[blcok如何避免内存泄漏 4](#_Toc34821358)

[strong和weak和assign的区别 5](#_Toc34821359)

[可以给Foundation框架中的类增加属性么？ 5](#_Toc34821360)

[！下面分类中的属性delegate，在.m中大致要干些什么才能实现？ 5](#_Toc34821361)

[谈谈锁机制 5](#_Toc34821362)

[什么情况下会出现死锁，如何解决呢？ 5](#_Toc34821363)

[浮点数在内存中如何存储？ 6](#_Toc34821364)

[iOS timer的内存泄露 6](#_Toc34821365)

[java GC root 7](#_Toc34821366)

[static类的回收 7](#_Toc34821367)

# 网络上的面试题：

### UIView和CAlayer的区别

UIView表示屏幕上的一块矩形区域，它是基本上是iOS中所有可视化控件的父类，UIView可以管理矩形区域里的内容，处理矩形区域的事件，包括对子视图的管理以及动画的实现。

UIView继承自UIResponder，所以UIView可以做事件响应，它也是iOS中所有视图(控件)直接或者间接的父类。

CALayer我们称之为层，CALayer类的概念与UIView非常类似，可以包含图片、文本、背景色等。CALayer直接继承自NSObject，没有事件响应的功能，这也是它与UIView最大的区别。CALayer中包含API，可判断某点是否在图层范围内，但是没有响应链的存在。

关系：在每一个UIView实例当中，都有一个默认的支持图层layer，UIView负责创建并且管理这个图层。实际上 UIView之所以能够显示,就是因为它里面有这个一个层,才具有显示的功能 ，UIView仅仅是对它的一层封装，实现了CALayer的delegate，提供了处理事件交互的具体功能，还有动画底层方法的高级API。可以说CALayer是UIView的内部实现细节。

区别

首先UIView可以响应事件，CALayer不可以响应事件

一个 Layer 的 frame 是由它的 anchorPoint,position,bounds,和 transform 共同决定的，而一个 View 的 frame 只是简单的返回 Layer的 frame

UIView主要是对显示内容的管理而 CALayer 主要侧重显示内容的绘制

在做 iOS 动画的时候，修改非 RootLayer的属性（譬如位置、背景色等）会默认产生隐式动画，而修改UIView则不会。

谈谈内存管理

ARC中会有内存泄漏么？

**循环引用（Retain Cycle）**

**什么是引用循环（retain cycle）**  
​假设我们有两个实例A和B，B是A的一个strong型的property，则B的引用计数是1，当A的需要释放的时候，A则会调用[B release]来释放B，B的引用计数则减为0，释放。

​可如果这时候将B的一个strong型property指向A，则A与B互相为强引用，问题就来了。因为B强引用A，A的引用计数永远不会减为0，当A原本的强引用对象被释放以后，A和B成为了一个相互引用的孤岛，永远不会被释放了，这就会引起内存泄漏。

​在上面的例子中，就是一种非常普遍的引用循环情况，加入如上代码的VC在dismiss或者pop以后，并不会执行dealloc方法，证明内存泄漏了。而引起泄漏的原因就是在作为self的property的block中，使用self指针导致self被block强引用，形成引用循环。

**1、Delegate**  
我们在使用代理设计模式的时候，一定要注意将 delegate 变量声明为 weak 类型，像这样  
@property (nonatomic, weak) id<xxxx> delegate;  
如使用strong或别的类型修饰的话将会导致循环引用，导致dealloc()不会被调用。从而触发一些意想不到的后果。

**2、Block**  
目前在项目中出现的内存泄漏大部分是因为block的问题。  
在 ARC 下，当 block 获取到外部变量时，由于编译器无法预测获取到的变量何时会被突然释放，为了保证程序能够正确运行，让 block 持有获取到的变量，向系统声明：我要用它，你们千万别把它回收了！然而，也正因 block 持有了变量，容易导致变量和 block 的循环引用，造成内存泄露！

[\_sortButton setButtonSpreadPreAction:^BOOL{

if (\_resultItems.count == 0) {

[progressHUD showText:@"xxxx"];

return NO;

}

return YES;

}];

这个例子的问题就在于在使用 block 的过程中形成了循环引用：self 持有 sortButton；sortButton 持有 block；block 持有 self。三者形成循环引用，内存泄露。

GCD已经一些系统级的API并不会提示循环引用的警告，但通过测试发现，大部分系统提供block也是需要弱引用的\_\_weak typeof(self) weakSelf = self;  
项目中除了AFN的第三方组件在调用block时都是需要弱引用的，如MJRefresh。

**3、NSTimer**  
​NSTimer在VC释放前，一定要调用[timer invalidate]，不调用的后果就是NSTimer无法释放其target，如果target正好是self，则会导致引用循环。

​这里要补充一点，引用循环不是只能有两个对象，三个四个更多都是可以的，甚至环数也不一定只有一个，所以要养成良好的代码习惯，在NSTimer停用前调用invalidate方法。

关于performSelector:afterDelay的问题

- (void)performSelector:(SEL)aSelector withObject:(id)anArgument afterDelay:(NSTimeInterval)delay

我们还是看看官方文档怎么说的。  
This method sets up a timer to perform the aSelector message on the current thread’s run loop. The timer is configured to run in the default mode (NSDefaultRunLoopMode). When the timer fires, the thread attempts to dequeue the message from the run loop and perform the selector. It succeeds if the run loop is running and in the default mode; otherwise, the timer waits until the run loop is in the default mode.

大概意思是系统依靠一个timer来保证延时触发，但是只有在runloop在default mode的时候才会执行成功，否则selector会一直等待run loop切换到default mode。根据我们之前关于timer  
的说法，在这里其实调用performSelector:afterDelay:同样会造成系统对target强引用，也即retain住。这样子，如果selector一直无法执行的话（比如runloop不是运行在default model下）,这样子同样会造成target一直无法被释放掉，发生内存泄露。怎么解决这个问题呢？其实很简单，我们在适当的时候取消掉该调用就行了，系统提供了接口:

+ (void)cancelPreviousPerformRequestsWithTarget:(id)aTarget

### blcok如何避免内存泄漏

<https://www.jianshu.com/p/1981365a45ce>

block中调用了self，将self的引用计数+1，这样当block释放时，self仍不得释放

#### 如何解决：

所以在使用block技术的时候，需要格外小心。有几个解决方法

approach 1: 让block里面的self的引用计数不要+1，这个时候做法是将" \_\_block B \*b1 = self;"这一行改为，" \_\_weak \_\_block B \*b1 = self;"，表示说“我block里面虽然会用到self，但是别担心，我不会讲引用计数+1的”

approach 2：在doAction函数内存的最后一行添加 self.completionBlock=nil； 因为block内部将self计数+1了，但是如果这个block自己先消亡，那么与之相关的一切都讲消亡（当然对于引用计数大于1的对象，不会消亡，只会计数减1）。

### strong和weak和assign的区别

<https://www.jianshu.com/p/585e8ed3ba35>

### 可以给Foundation框架中的类增加属性么？

<https://www.jianshu.com/p/23a7f0c99057>

void objc\_setAssociatedObject(id object, void \*key, id value, objc\_AssociationPolicy policy)

### ！下面分类中的属性delegate，在.m中大致要干些什么才能实现？

|  |  |
| --- | --- |
| 1  2  3 | @interface NSObject (Category)  @property (weak, nonatomic) id delegate;  @end |

谈谈objc\_send么？

如果消息传递失败了会怎么样？

### 谈谈锁机制

互斥锁、条件锁、自旋锁、读写锁<https://blog.csdn.net/bian_qing_quan11/article/details/73734157>

递归锁<https://blog.csdn.net/zouxinfox/article/details/5838861>

区域锁<https://blog.csdn.net/zouxinfox/article/details/5848519>

### 什么情况下会出现死锁，如何解决呢？

产生条件：

1）互斥条件：指进程对所分配到的资源进行排它性使用，即在一段时间内某资源只由一个进程占用。如果此时还有其它进程请求资源，则请求者只能等待，直至占有资源的进程用毕释放。打破互斥条件：改造独占性资源为虚拟资源，大部分资源已无法改造。

2）请求和保持条件：指进程已经保持至少一个资源，但又提出了新的资源请求，而该资源已被其它进程占有，此时请求进程阻塞，但又对自己已获得的其它资源保持不放。

打破请求和保持条件：当一进程占有一独占性资源后又申请一独占性资源而无法满足，则退出原占有的资源。

3）不剥夺条件：指进程已获得的资源，在未使用完之前，不能被剥夺，只能在使用完时由自己释放。

打破占有且申请条件：采用资源预先分配策略，即进程运行前申请全部资源，满足则运行，不然就等待，这样就不会占有且申请。

4）环路等待条件：指在发生死锁时，必然存在一个进程——资源的环形链，即进程集合{P0，P1，P2，···，Pn}中的P0正在等待一个P1占用的资源；P1正在等待P2占用的资源，……，Pn正在等待已被P0占用的资源。

打破循环等待条件：实现资源有序分配策略，对所有设备实现分类编号，所有进程只能采用按序号递增的形式申请资源。

浮点数在内存中如何存储？

<https://blog.csdn.net/boatalways/article/details/17121205>



在网络通信中如何保障浮点数的精确位数？

# 新氧iOS面试

### iOS timer的内存泄露

<https://www.jianshu.com/p/3614d92d41f1>

原因是timer强引用了target中的self，timer被加入到runloop之后又被当前的self强引用

解决方法：自己写一个target

### java GC root

为了使您的应用程序代码到达一个对象，应该有一个根对象，该根对象已连接到您的对象，并且能够从堆外部进行访问。从堆外部可以访问的此类根对象称为垃圾收集（GC）根。GC root有几种类型，例如局部变量，Active Java线程，静态变量，JNI引用等。只要对象被这些GC root直接或间接引用，并且GC根仍然存活，对象就可以被视为可到达的对象。一旦我们的对象失去被GC根的引用，它就变得不可访问，才有资格进行垃圾回收。

# iHandy iOS面试

### static类的回收

1、栈上的变量一旦声明，出作用域即会被回收。

2、堆里的对象，没有任何变量(栈上变量或静态区域的变量)指向这个对象的时候就会被回收，这个对象被标记为“垃圾对象”等待回收GC

3、GC是只回收堆空间，检查定时回收（频率由GLR决定）。

4、调用GC.Collect()；显式回收对象（必须等GC处理完目前的任务，才来处理该任务）。

静态成员垃圾回收：

静态成员一般也可以分为静态基本类型和静态引用类型。

静态基本类型存储在在静态变量区域；静态引用类型的引用存储在静态变量区域，而实例（具体内容）存储在堆上。静态成员只存在一份，静态成员加载时机：类加载的时候（第一次访问），这个类中所有静态成员就会被加载在静态存储区，**同时存储在静态变量区域的成员一旦创建，直到程序退出才会被回收**。（注：如果是引用类型，如static student myst=new student()，myst=null这时候，在静态存储区里面存的是一个地址（myst），这个地址指向在堆里面创建的student实例对象，当myst=null的时候，在静态存储区里面的变量会一直存在，但是在堆里面的student实例对象因为没有变量指向它，所以会被回收）。因此如果不用的静态引用类型可以通过设置=null方式让GC可以回收其堆上的空间。

# Bda数据挖掘面试

### TCP的自动关闭

<http://man7.org/linux/man-pages/man7/tcp.7.html>

**TCP\_USER\_TIMEOUT**关键字：

此选项将unsigned int作为参数。当该值大于0时，它指定在TCP强制关闭相应的连接并将ETIMEDOUT返回给应用程序之前，传输的数据可能仍未被确认的最长时间（以毫秒为单位）。如果选项值指定为0，则TCP将使用系统默认值。

 用户超时的增加允许TCP连接在没有端到端连接的情况下可以存活更长的时间。如果需要，减少用户超时可以使应用程序“快速失败”。否则，在正常的WAN环境中，当前的系统默认设置可能最多需要20分钟才能完成故障。

可以在TCP连接的任何状态下设置此选项，但是仅在连接的同步状态（ESTABLISHED，FIN-WAIT-1，FIN-WAIT-2，CLOSE-WAIT，CLOSING和LAST-ACK）下有效。 。此外，与TCP keepalive（SO\_KEEPALIVE）选项一起使用时，TCP\_USER\_TIMEOUT将覆盖keepalive以确定何时由于keepalive失败而关闭连接。

在TCP重新传输数据包或发送keepalive探针时，该选项无效。

**tcp\_keepalive\_time**关键字：

TCP开始发出keep-alive状态探测之前，连接需要空闲的秒数。 仅当启用SO\_KEEPALIVE套接字选项时，才会发送keep-alive。 默认值为7200秒（2小时）。 如果启用了keep-alive，则大约在另外11分钟（9个探测间隔为75秒）之后终止了一个空闲连接。

请注意，底层的连接跟踪机制和应用程序超时可能要短得多。