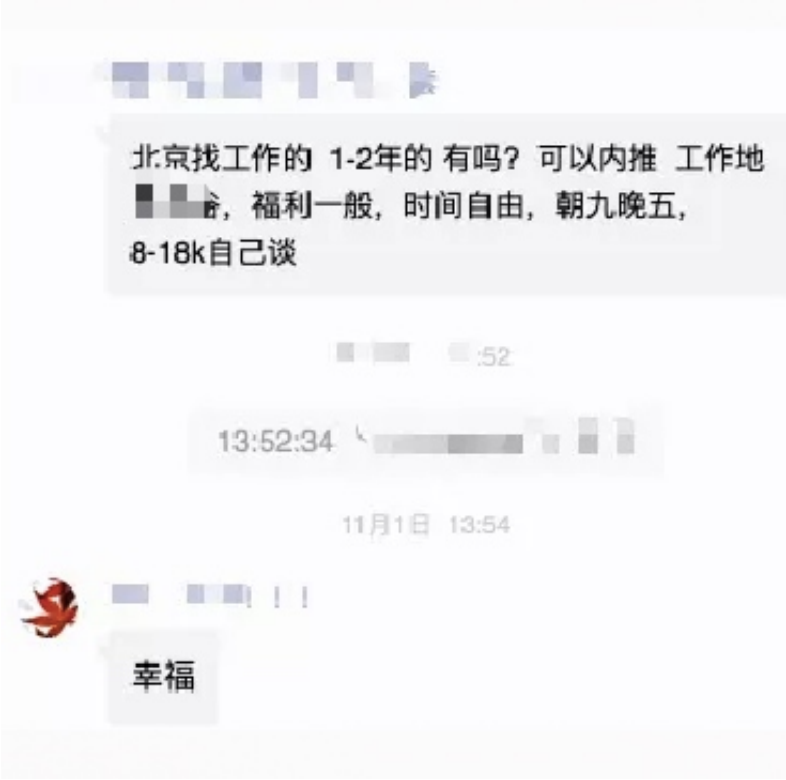
作者有一个iOS内推QQ群，里面有同iOS开发者提供内推岗位，有交流iOS面试题，iOS资源分享，技术交流；QQ群号：637919808



iOS面试整理

本文档有MWeb编写，如读者发现有语法问题请联系作者说明，分享目的是为了寻找志同道合的朋友一起创建一个我们自己的程序员圈子，有意者请联系作者，让我们一起搭建一个属于我们iOS开发者的私有圈子，内推、共享资源。

目录

底层：

2、你是否接触过OC中的反射机制？简单聊一下概念和使用

3、objc中的类方法和实例方法有什么本质区别和联系?

4、runloop的mode作用是什么?

5、1runloop是什么 / runloop的概念?

6、id与instanncetype

7、什么是GDB和LLDB?

8、模拟栈操作

9、你是否接触过OC中的反射机制？简单聊一下概念和使用

10、ldb（gdb）常用的调试命令?

11、block一般用那个关键字修饰，为什么？ 哪些情况会引起block循环引用，怎么解决

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

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

14、使用block时什么情况会发生引用循环，如何解决?

15、IB中User Defined Runtime Attributes如何使用?

基础：

16、weak和unowned

17、什么是GDB和LLDB?

18、static 关键字的作用：

19、Objective-C如何对内存管理的,说说你的看法和解决方法?

20、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?

21、NSOperation queue?

22、什么是延迟加载?

23、BAD_ACCESS在什么情况下出现?

24、0x8badf00d表示是什么?

25、GCD与NSOperation这两者有什么区别?

26、单例的优弊是什么?

27、RunLoop和线程的关系: 怎么让子线程执行完成任务后不销毁

28、nonatomic和atomic对比 说说你对他们的理解。

29、两个对象之间相互通信 有哪些通信方式

30、os的数据存储有哪些方式，sqlite数据库是否支持多个线程写入数据，如果不支持怎么解决?

31、一个函数执行10次，有二次结果不正确 八次正确，你应 该怎么检查该bug。

32、1-1000 个数字，顺序打乱，又丢失了三个数字，找出丢失的三个数字。

33、Object-C有私有方法吗？私有变量呢?

34、堆和栈的区别?

35、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?

36、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）

37、dispatchbarrierasync的作用是什么?

框架:

38、第三方内存泄漏检测工具：MLeaksFinder原理分析

39、逃逸闭包

40、cocoa touch框架

网络:

41、HTTP协议详解

42、谈谈你对HTTP 、TCP、 IP、socket 协议的理解

测试：

43、单元测试那些事!

44、为什么使用这个单元测试呢？它给我们带来了什么好处呢?

45、单元测试的重要性:

swift：

46、swift闭包

47、swift中高阶函数map用法

底层

1、runloop和线程有什么关系?

答案：

总的说来，Runloop，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 runloop 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, NSStringFromC
lass([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];
```

2、你是否接触过OC中的反射机制？简单聊一下概念和使用

答案：1). class反射
通过类名的字符串形式实例化对象。

```
        Class class = NSClassFromString(@"student");
        Student *stu = [[class alloc] init];
    将类名变为字符串。
        Class class =[Student class];
        NSString *className = NSStringFromClass(class);
```

2). SEL的反射
通过方法的字符串形式实例化方法。

```
SEL selector = NSSelectorFromString(@"setName");
        [stu performSelector:selector withObject:@"Mike"];
将方法变成字符串。
        NSStringFromSelector(@selector*(setName:));
```

3、objc中的类方法和实例方法有什么本质区别和联系?

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
类方法中不能直接调用对象方法

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)

4、runloop的mode作用是什么?

答案：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)
```

5、1runloop是什么 / runloop的概念?

Run loops是线程相关的的基础框架的一部分。一个run loop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。其实内部就是do－while循环，这个循环内部不断地处理各种任务（比 如Source，Timer，Observer）。使用run loop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。

6、id与instanncetype

id 与 instancetype 区别
1.id 可以当返回值类并且可以声明对象
2.instancetype 只可以当返回值类型
3.instancetype 返回和方法所在类相同类型的对象,id返回未知类型的对象(instancetype 会对返回值类型做一个检查,检查你这个返回值是不是当前类类型)
4.自定义初始化方法,返回值类型如果写成id,编译器会自动转换成instancetype

7、什么是GDB和LLDB?

我们在开发iOS程序的时候常常会用到调试跟踪，如何正确的使用调试器来debug十分重要。xcode里有内置的Debugger，老版使用的是GDB，xcode自4.3之后默认使用的就是LLDB了。
GDB:
UNIX及UNIX-like下的调试工具。
LLDB:
LLDB是个开源的内置于XCode的具有REPL(read-eval-print-loop)特征的Debugger，其可以安装C++或者Python插件。

8、模拟栈操作

```
/**
*  栈是一种数据结构，特点：先进后出
*  练习：使用全局变量模拟栈的操作
*/
#include <stdio.>
#include <stdbool.h>
#include <assert.h>

//保护全局变量：在全局变量前加static后，这个全局变量就只能在本文件中使用
static int data[1024];//栈最多能保存1024个数据
static int count = 0;//目前已经放了多少个数(相当于栈顶位置)

//数据入栈 push
```

```
void push(int x){
    assert(!full());//防止数组越界
    data[count++] = x;
}

//数据出栈 pop

int pop(){
    assert(!empty());
    return data[--count];
}
//查看栈顶元素 top
int top(){
    assert(!empty());
    return data[count-1];
}
//查询栈满 full
bool full() {
    if(count >= 1024) {
        return 1;
    }
     return 0;
}
//查询栈空 empty
bool empty() {
    if(count <= 0) {
        return 1;
    }
    return 0;
}

int main(){
     //入栈
    for (int i = 1; i <= 10; i++) {
        push(i);
```

```
    }
     //出栈
   while(!empty()){
        printf("%d ", top()); //栈顶元素
        pop(); //出栈
    }
    printf("\n");

    return 0;
}
```

9、你是否接触过OC中的反射机制？简单聊一下概念和使用

1). class反射

通过类名的字符串形式实例化对象。

```
Class class = NSClassFromString(@"student");
Student *stu = [[class alloc] init];
```

将类名变为字符串。

```
Class class =[Student class];
NSString *className = NSStringFromClass(class);
```

2). SEL的反射

通过方法的字符串形式实例化方法。

```
SEL selector = NSSelectorFromString(@"setName");
[stu performSelector:selector withObject:@"Mike"];
```

将方法变成字符串。

```
NSStringFromSelector(@selector*(setName:));
```

10、lldb（gdb）常用的调试命令?

- 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 更多 lldb（gdb） 调试命令可查看:http://lldb.llvm.org/lldb-gdb.html

11、block一般用那个关键字修饰，为什么？ 哪些情况会引起block循环引用，怎么解决

答案：copy 因为创建block 在栈内存里面，需要通过copy放入堆内存 进行管理 。
一个对象A引用另一个对象B ， B引用block ，bloc引用A，形成循环引用， block引用A使用 weak解决

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

循环引用

所有的引用计数系统，都存在循环应用的问题。

例如下面的引用关系：

· 对象a创建并引用到了对象b.

· 对象b创建并引用到了对象c.

· 对象c创建并引用到了对象b.

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

这种情况，必须打断循环引用，通过其他规则来维护引用关系。比如，我们常见的delegate往往是assign方式的属性而不是retain方式的属性，赋值不会增加引用计数，就是为了防止delegation两端产生不必要的循环引用。如果一个UITableViewController 对象a通过retain获取了UITableView对象b的所有权，这个UITableView对象b的delegate又是a，

如果这个delegate是retain方式的，那基本上就没有机会释放这两个对象了。自己在设计使用delegate模式时，也要注意这点。

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

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

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

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

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

你可以重载这个函数来释放controller中使用的其他内存。但要记得调用这个函数的super实现来允许父类（一般是UIVIewController）释放view。

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

14、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中又强引用了该对象，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。

```
    1.  id weak weakSelf = self; 或者 weak __typeof(&*self)weakSelf = self该方法可以设置宏
    2.  id __block weakSelf = self;
```

或者将其中一方强制制空 xxx = nil。
检测代码中是否存在循环引用问题，可使用 Facebook 开源的一个检测工具 FBRetainCycleDetector 。

15、IB中User Defined Runtime Attributes如何使用?

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

基础

16、weak 和 unowned

unowned 有点像oc里面的unsafe_unretained，而weak就是以前的weak。对于这两者的使用，不能说用哪一个要好一点，要视情况而定。用unowned的话，即使它原来的引用的内容被释放了，它仍然会保持对被已经释放了的对象的一个引用，它不能是Optional也不能是nil值，这个时候就会出现一个问题，如果你调用这个引用方法或者访问成员属性的话，就会出现崩溃（野指针）。而weak要稍微友善一点，在引用的内容被释放之后，会自动将weak的成员标记为nil。有人要说，既然这样，那我全部使用weak。但是在可能的情况下，我们还是应该倾向于尽量减少出现Optional的可能性，这样有助于代码的简化。Apple给我们的建议是如果能够确定访问时不会被释放的话，尽量用unowned，如果存在被释放的可能性的话，就用weak

17、什么是GDB和LLDB?

我们在开发iOS程序的时候常常会用到调试跟踪，如何正确的使用调试器来debug十分重要。xcode里有内置的Debugger，老版使用的是GDB，xcode自4.3之后默认使用的就是LLDB了。
GDB:
UNIX及UNIX-like下的调试工具。
LLDB:
LLDB是个开源的内置于XCode的具有REPL(read-eval-print-loop)特征的Debugger，其可以安装C++或者Python插件。

18、static 关键字的作用：

（1）函数体内 static 变量的作用范围为该函数体，不同于 auto 变量，该变量的内存只被分配一次，
因此其值在下次调用时仍维持上次的值；
（2）在模块内的 static 全局变量可以被模块内所用函数访问，但不能被模块外其它函数访问；
（3）在模块内的 static 函数只可被这一模块内的其它函数调用，这个函

数的使用范围被限制在声明
它的模块内；
（4）在类中的 static 成员变量属于整个类所拥有，对类的所有对象只有一份拷贝；
（5）在类中的 static 成员函数属于整个类所拥有，这个函数不接收 this 指针，因而只能访问类的static 成员变量。

19、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 – 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.

20、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,

21、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。

22、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。

23、BAD_ACCESS在什么情况下出现?

答：这种问题在开发时经常遇到。原因是访问了野指针，比如访问已经释放对象的成员变量或者发消息、死循环等。

24、0x8badf00d表示是什么?

1、0x8badf00d：读作“ate bad food”！（把数字换成字母，是不是很像：p）该编码表示应用是因为发生watchdog超时二被iOS终止的。通常是应用话费太多时间而无法启动、终止或响应应用系统事件。
0xbad22222:该编码表示Volp应用因为过于频繁启动而被终止
3、0xdead10cc：读作“dead lock”！该代码表明应用，因为在后台运行时占用系统资源，如通讯录数据库不释放而被终止。
4、0xdeadfa11:读作“dead fall”！该代码表示应用是被用户强制退出的。根据苹果文档，强制退出发生在用户长按开关按钮直到出现“滑动来关机，然后长按Home按钮。强制退出将产生 包含0xdeadfa11 异常编码的奔溃日志，因为大多数强制退出是因为应用阻塞了界面。

25、GCD与NSOperation这两者有什么区别?

GCD:
1、GCD是底层的语言构成的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没发停止已经加入queue的blok（其实有的，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Opaeration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是否完成或者取消，这样子能比GCD更加有效地掌握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进行继续，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。总的来说，Operation queue提供了更多你在编写多线程程序时需要的功能，并隐藏了多线程调度，线程取消与线程优先级的复杂代码，为我们提供简单的API入口。从编程原则来说，一般我们需要尽可能的使用高等级、封装完美的API，在必须时才使用底层API。但是我认为当我们的需求能够以更简单的底层代码完成的时候，简洁的GCD或许是个更好的选择，而Operation queue为我们提供能更多的选择。

26、单例的优弊是什么?

优点:
1、一个类只被实例化一次，提供了对唯一实例对受控访问。
2、节省系统资源
3、允许可变数目的实例
缺点:
1、一个类只有一个对象，可能造成责任过重，在一定程度上违背了“单一职责原则”
2、由于单例模式中没有抽象层，因此单例类的扩展有很大的困难。
3、滥用单例将带来一些负面问题，如为了节省资源将数据库连接池对象设计为的单例类，可能会导致共享连接池对象的程序过多而出现连接池溢

出；如果实例化的对象长时间不被利用，系统会认为是垃圾而被回收，这将导致对象状态的丢失。

27、RunLoop和线程的关系: 怎么让子线程执行完成任务后不销毁

答案：
1. runloop与线程是一一对应的，一个runloop对应一个核心的线程，为什么说是核心的，是因为runloop是可以嵌套的，但是核心的只能有一个，他们的关系保存在一个全局的字典里。
2. runloop是来管理线程的，当线程的runloop被开启后，线程会在执行完任务后进入休眠状态，有了任务就会被唤醒去执行任务。
3. runloop在第一次获取时被创建，在线程结束时被销毁。
4. 对于主线程来说，runloop在程序一启动就默认创建好了。
5. 对于子线程来说，runloop是懒加载的，只有当我们使用的时候才会创建，所以在子线程用定时器要注意：确保子线程的runloop被创建，不然定时器不会回调。
[RunLoop run]来让该线程长时间存活而不被销毁。

28、nonatomic和atomic对比 说说你对他们的理解。

答案：
atomic：原子属性，为setter方法加锁（默认就是atomic）
nonatomic：非原子属性，不会为setter方法加锁

atomic加锁原理
原子和非原子属性的选择
nonatomic和atomic对比
atomic：线程安全，需要消耗大量的资源
nonatomic：非线程安全，适合内存小的移动设备

29、两个对象之间相互通信 有哪些通信方式

答案： 委托代理 通知 使用block 单例 使用对象指针

30、os的数据存储有哪些方式，sqlite数据库是否支持

多个线程写入数据，如果不支持怎么解决?

答案:

Plist nscoding NSUserDefault sqliste

不支持 利用线程锁 锁住写数据库的地方

31、一个函数执行10次，有二次结果不正确 八次正确，你应 该怎么检查该bug。

答案： 检查该函数各个分支，使其每个分支业务都正确 查看传入的参数是否正确 。根据参数看执行的地方。检查该函数里面的全局变量是否有问题。

32、1-1000 个数字，顺序打乱，又丢失了三个数字，找出丢失的三个数字。

答案:

建立一个新的数组 10001长度， 全部置0， 循环丢失三个数的数组， 每个数据的数写到新数组对应的位置， 再循环新的数组，为0的地方就是丢失的数据（除了第0位）。

33、Object-C有私有方法吗？私有变量呢?

objective-c – 类里面的方法只有两种, 静态方法和实例方法. 这似乎就不是完整的面向对象了,按照OO的原则就是一个对象只暴露有用的东西. 如果没有了私有方法的话, 对于一些小范围的代码重用就不那么顺手了. 在类里面声名一个私有方法

```
@interface Controller : NSObject { NSString *something; }
+ (void)thisIsAStaticMethod;
- (void)thisIsAnInstanceMethod;
@end
@interface Controller (private) -
(void)thisIsAPrivateMethod;
@end
```

@private可以用来修饰私有变量

在Objective-C中，所有实例变量默认都是私有的，所有实例方法默认都是

公有的。

34、堆和栈的区别?

管理方式：对于栈来讲，是由编译器自动管理，无需我们手工控制；对于堆来说，释放工作由程序员控制，容易产生memory leak。

申请大小：

栈：在Windows下,栈是向低地址扩展的数据结构，是一块连续的内存的区域。这句话的意思是栈顶的地址和栈的最大容量是系统预先规定好的，在WINDOWS下，栈的大小是2M（也有的说是1M，总之是一个编译时就确定的常数），如果申请的空间超过栈的剩余空间时，将提示overflow。因此，能从栈获得的空间较小。

堆：堆是向高地址扩展的数据结构，是不连续的内存区域。这是由于系统是用链表来存储的空闲内存地址的，自然是不连续的，而链表的遍历方向是由低地址向高地址。堆的大小受限于计算机系统中有效的虚拟内存。由此可见，堆获得的空间比较灵活，也比较大。

碎片问题：对于堆来讲，频繁的new/delete势必会造成内存空间的不连续，从而造成大量的碎片，使程序效率降低。对于栈来讲，则不会存在这个问题，因为栈是先进后出的队列，他们是如此的一一对应，以至于永远都不可能有一个内存块从栈中间弹出

分配方式：堆都是动态分配的，没有静态分配的堆。栈有2种分配方式：静态分配和动态分配。静态分配是编译器完成的，比如局部变量的分配。动态分配由alloca函数进行分配，但是栈的动态分配和堆是不同的，他的动态分配是由编译器进行释放，无需我们手工实现。

分配效率：栈是机器系统提供的数据结构，计算机会在底层对栈提供支持：分配专门的寄存器存放栈的地址，压栈出栈都有专门的指令执行，这就决定了栈的效率比较高。堆则是C/C++函数库提供的，它的机制是很复杂的。

35、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?

- 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

36、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）

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_QU
EUE_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
});
```

37、`dispatch_barrier_async` 的作用是什么？

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。 dispatchbarrierasync 函数追加的内容就如同 “上完厕所就上高速”这个动作。
（注意：使用 `dispatch_barrier_async` ，该函数只能搭配自定义并行队列 `dispatch_queue_t` 使用。不能使用： `dispatch_get_global_queue` ，否则 `dispatch_barrier_async` 的作用会和 `dispatch_async` 的作用一模一样。）

框架：

38、第三方内存泄漏检测工具：MLeaksFinder原理分析

MLeakFinder的UIViewController扩展在load方法中，viewDidDisappear:的实现配合 UINavigationController来在UIViewController被pop出去的时候调用willDealloc。
willDealloc 的实现首先调用了 NSObject+MemoryLeak 扩展的willDealloc，
willDealloc 然后调用 willReleaseChildren 构建了parentPtrs用来判断父对象是否被释放，还有viewStack来展示层级结构。
willDealloc的NSObject默认实现在两秒后，如果没有被释放，就调用assertNotDealloc。(父对象会比子对象的生命周期先结束)
assertNotDealloc 先判断是否有父对象未释放
a. 如果没有就使用当前对象
b. 创建MLeakedObjectProxy关联到当前对象
c. 将当前对象指针加入到leakedObjectPtrs
d. 当前对象被释放的时候对应的MLeakedObjectProxy会被释放
e. MLeakedObjectProxy被释放的时候，对应的对象被从leakedObjectPtrs移除

39、逃逸闭包

//当一个闭包作为参数传到一个函数中，但是这个闭包在函数返回之后才被执行，我们称该闭包从函数中逃逸。当你定义接受闭包作为参数的函数时，你可以在参数名之前标注 @escaping，用来指明这个闭包是允许“逃逸”出这个函数的。

//一种能使闭包“逃逸”出函数的方法是，将这个闭包保存在一个函数外部定义的变量中。举个例子，很多启动异步操作的函数接受一个闭包参数作为 completion handler。这类函数会在异步操作开始之后立刻返回，但是闭包直到异步操作结束后才会被调用。在这种情况下，闭包需要“逃逸”出函数，因为闭包需要在函数返回之后被调用

40、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

网络

41、HTTP协议详解

HTTP是一个属于应用层的面向对象的协议，由于其简捷、快速的方式，适用于分布式超媒体信息系统。目前在WWW中使用的是HTTP/1.0的第六版，HTTP/1.1的规范化工作正在进行之中。
http（超文本传输协议）是一个基于请求与响应模式的、无状态的、应用层的协议，常基于TCP的连接方式，HTTP1.1版本中给出一种持续连接的机制，绝大多数的Web开发，都是构建在HTTP协议之上的Web应用。
HTTP协议的主要特点可概括如下：1.支持客户/服务器模式。2.简单快速：客户向服务器请求服务时，只需传送请求方法和路径。请求方法常用的有GET、HEAD、POST。每种方法规定了客户与服务器联系的类型不同。由于HTTP协议简单，使得HTTP服务器的程序规模小，因而通信速度很快。3.灵活：HTTP允许传输任意类型的数据对象。正在传输的类型由Content-Type加以标记。4.无连接：无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间。5.无状态：HTTP协议是无状态协议。无状态是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快。

42、谈谈你对HTTP 、TCP、 IP、socket 协议的理解。

HTTP

超文本传输协议 （HTTP，HyperText Transfer Protocol)

HTTP 协议对应应用层，HTTP 协议是基于 TCP 连接的。HTTP 链接就是所谓的短连接，即客户端向服务器发送一次请求，服务器端响应后连接即

会断掉。

HTTP 1.0中，客户端的每次请求都要求建立一次单独的连接，在处理完本次请求后，就自动释放连接。

HTTP 1.1则可以在一次连接中处理多个请求，并且多个请求可以重叠进行，不需要等待一个请求结束后再发送下一个请求。

因为 HTTP 是“短连接”，所以要保持客户端程序的在线状态，需要不断的向服务器发起连接请求。一般的做法就是不需要获取任何数据，客户端也保持每隔一段固定的时间向服务器发送一次“保持连接”的请求，服务器在收到该请求后对客户端进行回复，表示知道客户端“在线”。若服务器长时间无法收到客户端的请求，则认为客户端“下线”，若客户端长时间无法收到服务器的回复，则认为网络已经断开。

TPC/IP

TPC/IP 协议是传输层协议，主要解决数据如何在网络中传输。“IP”代表网际协议，TCP和UDP使用该协议从一个网络传送数据包到另一个网络。把IP想像成一种高速公路，它允许其它协议在上面行驶并找到到其它电脑的出口。TCP和UDP是高速公路上的“卡车”，它们携带的货物就是像HTTP，文件传输协议FTP这样的协议等。

你应该能理解，TCP和UDP是FTP，HTTP和SMTP之类使用的传输层协议。虽然TCP和UDP都是用来传输其他协议的，它们却有一个显著的不同：TCP提供有保证的数据传输，而UDP不提供。这意味着TCP有一个特殊的机制来确保数据安全的不出错的从一个端点传到另一个端点，而UDP不提供任何这样的保证。

下面的图表试显示不同的TCP/IP和其他的协议在最初OSI模型中的位置：

TCP 标志位，有以下6种标示

1、SYN(synchronous建立联机)

2、ACK(acknowledgement 确认)

3、PSH(push传送)

4、FIN(finish结束)

5、RST(reset重置)

6、URG(urgent紧急)

Sequence number(顺序号码)

Acknowledge number(确认号码)

客户端 TCP 状态迁移：

`CLOSED->SYN_SENT->ESTABLISHED->FIN_WAIT_1->FIN_WAIT_2->TIME_WAIT->CLOSED`

服务器TCP状态迁移：

`CLOSED->LISTEN->SYN收到->ESTABLISHED->CLOSE_WAIT->LAST_ACK->CLOSED`

各个状态的意义如下：

LISTEN - 侦听来自远方TCP端口的连接请求；

SYN-SENT -在发送连接请求后等待匹配的连接请求；

SYN-RECEIVED - 在收到和发送一个连接请求后等待对连接请求的确认；

ESTABLISHED- 代表一个打开的连接，数据可以传送给用户；

FIN-WAIT-1 - 等待远程TCP的连接中断请求，或先前的连接中断请求的确认；

FIN-WAIT-2 - 从远程TCP等待连接中断请求；

CLOSE-WAIT - 等待从本地用户发来的连接中断请求；

CLOSING -等待远程TCP对连接中断的确认；

LAST-ACK - 等待原来发向远程TCP的连接中断请求的确认；

TIME-WAIT -等待足够的时间以确保远程TCP接收到连接中断请求的确认；

CLOSED - 没有任何连接状态；

TCP/IP 三次握手

第一次握手：建立连接时，客户端A发送SYN包（SYN=j）到服务器B，并进入SYN_SEND状态，等待服务器B确认。

第二次握手：服务器B收到SYN包，必须确认客户A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B进

入SYN_RECV状态。

第三次握手：客户端A收到服务器B的SYN＋ACK包，向服务器B发送确认包ACK（ACK=k+1），此包发送完毕，客户端A和服务器B进入ESTABLISHED状态，完成三次握手。 完成后，客户端和服务器开始传送数据。

由于TCP连接是全双工的，因此每个方向都必须单独进行关闭。这个原则是当一方完成它的数据发送任务后就能发送一个FIN来终止这个方向的连接。收到一个 FIN只意味着这一方向上没有数据流动，一个TCP连接在收到一个FIN后仍能发送数据。首先进行关闭的一方将执行主动关闭，而另一方执行被动关闭。

CP的连接的拆除需要发送四个包，因此称为四次挥手(four-way handshake)。客户端或服务器均可主动发起挥手动作，在socket编程中，任何一方执行close()操作即可产生挥手操作。

客户端A发送一个FIN，用来关闭客户A到服务器B的数据传送。

服务器B收到这个FIN，它发回一个ACK，确认序号为收到的序号加1。和SYN一样，一个FIN将占用一个序号。

服务器B关闭与客户端A的连接，发送一个FIN给客户端A。

客户端A发回ACK报文确认，并将确认序号设置为收到序号加1。

深入理解TCP连接的释放：

由于TCP连接是全双工的，因此每个方向都必须单独进行关闭。这原则是当一方完成它的数据发送任务后就能发送一个FIN来终止这个方向的连接。收到一个 FIN只意味着这一方向上没有数据流动，一个TCP连接在收到一个FIN后仍能发送数据。首先进行关闭的一方将执行主动关闭，而另一方执行被动关闭。

TCP协议的连接是全双工连接，一个TCP连接存在双向的读写通道。

简单说来是 “先关读，后关写”，一共需要四个阶段。以客户机发起关闭连接为例：

1.服务器读通道关闭 2.客户机写通道关闭 3.客户机读通道关闭 4.服务器写通道关闭

关闭行为是在发起方数据发送完毕之后，给对方发出一个FIN（finish）数

据段。直到接收到对方发送的FIN，且对方收到了接收确认ACK之后，双方的数据通信完全结束，过程中每次接收都需要返回确认数据段ACK。

详细过程：

第一阶段：客户机发送完数据之后，向服务器发送一个FIN数据段，序列号为i；

1.服务器收到FIN(i)后，返回确认段ACK，序列号为i+1，关闭服务器读通道；

2.客户机收到ACK(i+1)后，关闭客户机写通道；

（此时，客户机仍能通过读通道读取服务器的数据，服务器仍能通过写通道写数据）

第二阶段：服务器发送完数据之后，向客户机发送一个FIN数据段，序列号为j；

3.客户机收到FIN(j)后，返回确认段ACK，序列号为j+1，关闭客户机读通道；

4.服务器收到ACK(j+1)后，关闭服务器写通道。

这是标准的TCP关闭两个阶段，服务器和客户机都可以发起关闭，完全对称。

FIN标识是通过发送最后一块数据时设置的，标准的例子中，服务器还在发送数据，所以要等到发送完的时候，设置FIN（此时可称为TCP连接处于半关闭状态，因为数据仍可从被动关闭一方向主动关闭方传送）。如果在服务器收到FIN(i)时，已经没有数据需要发送，可以在返回ACK(i+1)的时候就设置FIN(j)标识，这样就相当于可以合并第二步和第三步。

TCP的 `TIME_WAIT` 和 `CLOSE_WAIT` 状态

`CLOSE_WAIT:`

发起TCP连接关闭的一方称为client，被动关闭的一方称为server。被动关闭端未发出FIN的TCP状态是 `CLOASE_WAIT` 。出现这种状况一般都是由于server端代码的问题，如果你的服务器上出现大量 `CLOSE_WAIT` ，应该要考虑检查代码。

TIME_WAIT：

根据TCP协议定义的3次握手断开连接规定,发起socket主动关闭的一方socket将进入 `TIME_WAIT` 状态。 `TIME_WAIT` 状态将持续2个MSL(Max Segment Lifetime),在Windows下默认为4分钟，即240秒。TIME_WAIT状态下的socket不能被回收使用. 具体现象是对于一个处理大量短连接的服务器,如果是由服务器主动关闭客户端的连接，将导致服务器端存在大量的处于 `TIME_WAIT` 状态的socket， 甚至比处于Established状态下的socket多的多,严重影响服务器的处理能力，甚至耗尽可用的socket，停止服务。
socket

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。

套接字(socket)是通信的基石，是支持TCP/IP协议的网络通信的基本操作单元。它是网络通信过程中端点的抽象表示，包含进行网络通信必须的五种信息：连接使用的协议，本地主机的IP地址，本地进程的协议端口，远地主机的IP地址，远地进程的协议端口。

我们平时说的最多的socket是什么呢，实际上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。 实际上，Socket跟TCP/IP协议没有必然的联系。Socket编程接口在设计的时候，就希望也能适应其他的网络协议。所以说，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，比如create、 listen、connect、accept、send、read和write等等。网络有一段关于socket和TCP/IP协议关系的说法比较容易理解：

“TCP/IP只是一个协议栈，就像操作系统的运行机制一样，必须要具体实现，同时还要提供对外的操作接口。这个就像操作系统会提供标准的编程接口，比如win32编程接口一样，TCP/IP也要提供可供程序员做网络开发所用的接口，这就是Socket编程接口。”

实际上，传输层的TCP是基于网络层的IP协议的，而应用层的HTTP协议又是基于传输层的TCP协议的，而Socket本身不算是协议，就像上面所说，它只是提供了一个针对TCP或者UDP编程的接口。socket是对端口通信开

发的工具,它要更底层一些.

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)。

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。

测试

43、单元测试那些事!

OCUnit（即用XCTest进行测试）其实就是苹果自带的测试框架，我们主要讲的就是这个。GHUnit是一个可视化的测试框架。（有了它，你可以点击APP来决定测试哪个方法，并且可以点击查看测试结果等。）OCMock就是模拟某个方法或者属性的返回值，你可能会疑惑为什么要这样做?使用用模型生成的模型对象，再传进去不就可以了？答案是可以的，但是有特殊的情况。比如你测试的是方法A，方法A里面调用到了方法B，而且方法B是有参数传入，但又不是方法A所提供。这时候，你可以使用OCMock来模拟方法B返回的值。（在不影响测试的情况下，就可以这样去模拟。）除了这些，在没有网络的情况下，也可以通过OCMock模拟返回的数据。UITests就是通过代码化来实现自动点击界面，输入文字等功能。靠人工操作的方式来覆盖所有测试用例是非常困难的，尤其是加入新功能以后，旧的功能也要重新测试一遍，这导致了测试需要花非常多的时间来进行回归测试，这里产生了大量重复的工作，而这些重复的工作有些是可以自动完成的，这时候UITests就可以帮助解决这个问题了

44、那为什么使用这个单元测试呢？它给我们带来了什么好处呢?

答：通常一般我们为了省事 通常会直接在工程中写一些测试代码 如NSLog一些打印测试 断言 等等，但是 如果测试代码过多会是工程变得看

起来很乱，而且工程体积也会变的大一些，当我们测试完成以后还要将以前写的测试代码删掉，难道不觉得我们辛辛苦苦写的测试代码 再我们不需要时候再删掉 或者说如果我们再次需要这个测试的时候 再一次来写 这种反复操作很烦嘛？ 当然单元测试很好的解决了这一切。 因为测试单元中的代码 不会打包到我们的工程里面，而且它也不会build整个工程只会run你所写的那个方法 省时 省力！空说 可能无法理解它的好处，用起来就知道了

45、单元测试的重要性:

现在很多公司都在抓质量，质量！质量！质量！为什么都在抓质量，在IT行业多元化复杂化的今天，也就意味着竞争会异常的激烈，那么作为互联网软件公司，怎么提升我们的竞争力？我们不是某些国企，不需要某些套路，我们的企业是否能生存，取决于我们的用户，抓住我们的用户，我们就能生存，怎么抓住我们的用户？那必须要获得用户的信任，让他们能放心的投资在我们身上，购买我们的产品。

抓住我们的用户，那么我们就需要提高我们的产品质量，这也是取得用户信任的关键步骤，那么我们是做软件的，就必须要提高我们的软件质量。针对软件质量可以从很多方面去提升，开发，测试，运维等等。开发里面又要分静态检查，白盒测试；测试里面又要分功能测试，性能测试等等，每个环节都有很多提高我们质量的方法，今天呢我只针对开发来讲怎么提高我们的代码质量，而且只针对白盒测试，也就是单元测试。

我曾经待过的公司，几乎上到公司CTO，下到开发，几乎人人鼓吹白盒测试有多好，对质量的杀伤力有多大，可是，能真正做起来单元测试的公司少之又少，小公司基本不愿意投入做单元测试的人力成本，大公司呢，说起来容易，个个开发几乎都是自己写完代码自己测试两次了事~那么我们到底有没有必要来做单元测试呢？我来说说我的观点，分别从时间和工作量来分析。

时间

即使你没有多少开发经验，你也应该能够想象，单元测试最大的问题，就是它需要花时间花精力去写，那么这个花费是否值得呢？这还是由你架构的目标决定的，或者你的需求决定的。如果系统是一次成型交付使用，此后几乎不会更改的，那么一次性的手工测试就够了；但如果你的系统是会被“千锤百炼”的不断折腾修改的，那么这个测试就是很有必要的。最简单的考虑：每一次更改，我都要手工测试一次；那还不会如我多花点时间，弄个“自动化”的东西出来。单元测试，其实就可以理解为一种自动

化的测试工具。

但是“自动化”的理由还远远不够。因为你马上想到的，每一次需求变更代码调整，测试代码也得相应的改呀？没有测试代码，我就只需要改开发代码；现在有了单元测试，我还得再改测试代码。本来我只维护一套代码，现在我凭空增加了一套代码也需要维护，这不是增加了维护成本，不是和你“可维护性”的架构目标背道而驰了么？是一套代码好维护呢，还是两套代码更好维护?

这是一个非常好的问题，适用于很多情景（比如分层架构，你说分层解耦，实际上还不是一改就得从UI层改到数据库，每一层都得改？）。我能给出的回答大概有：

一、无论有无单元测试，开发代码进行修改之后，是不是都要进行测试？没有单元测试，并不代表你的代码就不需要测试了，只不过是你手工的去测试了一遍而已。切记：程序员的工作并不只是把代码写出来而已！　　二、进行手工测试，和更改单元测试，两者的耗费比，会根据测试重用的次数而变化。一次手工测试可能需要5分钟跑完，更改单元测试代码可能需要20分钟，但如果这测试会跑100遍，单元测试完胜手工测试。你说，哪里哟？什么功能会改100遍？我没说你的功能会改100遍，我说的是测试会跑100遍。有区别么?

工作量

所以其实当对象间的关系变得越来越错综复杂，像一张密密麻麻的网一样之后，一个局部的改动就很有可能会触发极其复杂的连锁反应。所以为了保险起见，所有可能相关的组件都应该进行测试（所谓的“回归测试”）。这时候如果只有纯粹的手工测试，会面临两个问题：

* 难以确定测试的边界（那些部分可能会被影响），凭空想？手动代码走读?

* 极大的测试耗费。而且这种耗费是相当的无聊繁琐伤人心的——没人愿意做这种事。

测试只能告诉你出了bug，不能告诉你根源啊。没有单元测试，单步调试需要花更多的时间。这是系统本身的复杂性，或者代码组织的不合理造成的，不能归咎于单元测试。不还是有这么多开源代码都有详尽的单元测试么？他们是怎么做到的呢？在单元测试上的付出，最终一定会获得超值回报！想想没有单元测试的公司，那超级庞大的测试团队，或者四处冒烟的系统，你愿意走这么一条路么?

这样一种说法：可测试的代码不一定是好代码，但坏代码几乎是不可能被测试的。深度耦合的代码，写他们的单元测试，难于上青天；但反过

来，我们可以以可测试为标准，不断的完善重构开发代码，只要这样坚持下来，最终代码的质量怎么都不会差到哪里去。

总结：所以，个人观点，对于某些关键功能和通用底层库，必须要做单元测试，如果真的要把产品所有的代码都做单元测试，其实是没有必要的，也不必要花这么多的财力人力。当然，通过做单元测试，不仅能提升公司产品的竞争力和稳定性，也可以提升开发人员的开发能力。

swift

46、swift闭包

闭包(Closures)是自包含的功能代码块，可以在代码中使用或者用来作为参数传值。
Swift 中的闭包与 C 和 Objective-C 中的代码块（blocks）以及其他一些编程语言中的 匿名函数比较相似。
全局函数和嵌套函数其实就是特殊的闭包。

闭包的形式有：
全局函数：有名字但不能捕获任何值。
嵌套函数：有名字，也能捕获封闭函数内的值。
闭包表达式：无名闭包，使用轻量级语法，可以根据上下文环境捕获值。

Swift中的闭包有很多优化的地方:
1.根据上下文推断参数和返回值类型
2.从单行表达式闭包中隐式返回（也就是闭包体只有一行代码，可以省略return）
3.可以使用简化参数名，如 `$0, $1` (从0开始，表示第i个参数...)
4.提供了尾随闭包语法(Trailing closure syntax)

47、swift中高阶函数map用法

Swift相比于Objective-C一个重要的优点，它对函数式编程提供了很好的支持，Swift提供了一些高阶函数作为对容器的支持

```
let strArr = ["Objective-C", "Swift", "Java", "C", "C++"]

func count(string: String) -> Int {
    return string.characters.count
```

```swift
}
let strCountArr1 = strArr.map(count)
print(strCountArr1)


let strCountArr2 = strArr.map({string -> Int in
    return string.characters.count
})
print(strCountArr2)

// $0代表数组中的每一个元素
let strCountArr3 = strArr.map{
    return $0.characters.count
}
print(strCountArr3)
```