招聘一个靠谱的 iOS

近一年内陆续面试了不少人了，从面试者到面试官的转变让我对 iOS 招聘有了更多的感受。经过了前段时间的一大波面试，我们终于找到了志同道合的小伙伴，面试也暂时告一段落了。总结下面试人过程中的感受，你也可以读到我们对简历、算法、性格、iOS 基础、底层知识的看法和一些常问的面试题。

一个靠谱的简历

简历非常能反映一个人的性格和水平，相比于你在学校获得多少奖项，工作经历、项目经历、熟悉的技术等更加关键，如果还有博客和一些 Github 上的项目，好感度++，但记得在去面试前收拾下，我们真的会挨个文件 review 你的开源代码的。我们还喜欢关注一些细节，比如简历里关键字的拼写，看似无关紧要但很能反映出对自己的要求，经常见一个简历中 **iOS** 这三个字母的拼写就出现 IOS、iOS、ios 三种的，非常不能忍，再列举几个常见问题：

iPhone -> IPHONE IPhone
Xcode -> XCode xcode
Objective-C -> Object-C
JSON -> Json
HTTP -> Http

还有，注意中英文间用一个半角空格隔开，排版会漂亮很多，简历承载的不仅是内容，还有细节和态度，上面这些点往往都反映着面试者的代码风格、做事的认真程度。当然，简历写的很漂亮但面聊之后发现啥都不会的也有，甚至见过来面试上来就跟我说简历是假的，就想求个面试机会这种 - -

面试

别迟到，别迟到，别迟到，重要的事说三遍。有变动提前通知 HR，碰到过临时有事没来，和谁都不说一声，打电话过去还要求改个时间的，这种直接拜拜。
面试时最好准备纸、笔、简历，可能用不上，但很能体现认真程度。有条件的话带着 Mac 和源码，手机中装好所有在简历中出现的 App。

关于算法

我们是实用主义，iOS 开发中很少需要自己写复杂的算法，所以不在面试考核标准中。

代码规范

这是一个重点考察项，曾经在微博上发过一个风格纠错题：

```
typedef enum{
    UserSex_Man,
    UserSex_Woman
}UserSex;

@interface UserModel :NSObject


@property(nonatomic, strong) NSString *name;
@property (assign, nonatomic) int age;
@property (nonatomic, assign) UserSex sex;

-(id)initUserModelWithUserName: (NSString*)name withAge:(int)age;


-(void)doLogIn;

@end
```


也曾在面试时让人当场改过，槽点不少，能够有 10 处以上修改的就基本达到标准了（处女座的人在这方面表现都很优秀

一个区分度很大的面试题

考察一个面试者基础咋样，基本上问一个 @property 就够了：

- @property 后面可以有哪些修饰符？
- 什么情况使用 weak 关键字，相比 assign 有什么不同？
- 怎么用 copy 关键字？
- 这个写法会出什么问题：`@property (copy) NSMutableArray *array;`
- 如何让自己的类用 copy 修饰符？如何重写带 copy 关键字的 setter？

这一套问题区分度比较大，如果上面的问题都能回答正确，可以延伸问更深入点的：

- @property 的本质是什么？ivar、getter、setter 是如何生成并添加到这个类中的
- @protocol 和 category 中如何使用 @property
- runtime 如何实现 weak 属性

每个人擅长的领域不一样，我们一般会从简历上找自己写擅长的技术聊，假如自己并不是很熟，最好别写出来或扯出来，万一面试官刚好非常精通这里就露馅了。

Checklist

总结过些面试题，没坚持下去，后来把这些当 checklist，面试的时候实在没话聊的时候做个提醒，语言、框架、运行机制性质的：

[※]@property 中有哪些属性关键字？
[※]weak 属性需要在 dealloc 中置 nil 么？
[※※]@synthesize 和@dynamic 分别有什么作用？
[※※※]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？
[※※※]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？
[※※※]@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？
[※※※※※]在有了自动合成属性实例变量之后，@synthesize 还有哪些使用场景？

[※※]objc 中向一个 nil 对象发送消息将会发生什么？
[※※※]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？
[※※※]什么时候会报 `unrecognized selector` 的异常？
[※※※※]一个 objc 对象如何进行内存布局？（考虑有父类的情况）
[※※※※]一个 objc 对象的 `isa` 的指针指向什么？有什么作用？
[※※※※]下面的代码输出什么？

```
@implementation Son : Father
- (id)init
{
    self = [super init];
    if (self) {
        NSLog(@"%@", NSStringFromClass([self class]));
        NSLog(@"%@", NSStringFromClass([super class]));
    }
    return self;
}
@end
```

[※※※※]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）
[※※※※]使用 `runtime Associate` 方法关联的对象，需要在主对象 dealloc 的时候释放么？
[※※※※※]objc 中的类方法和实例方法有什么本质区别和联系？
[※※※※※]`_objc_msgForward` 函数是做什么的，直接调用它将会发生什么？
[※※※※※]runtime 如何实现 weak 变量的自动置 nil？
[※※※※※]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？

[※※※]runloop 和线程有什么关系？
[※※※]runloop 的 mode 作用是什么？
[※※※※]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？
[※※※※※]猜想 runloop 内部是如何实现的？

[※]objc 使用什么机制管理对象内存？
[※※※※]ARC 通过什么方式帮助开发者管理内存？
[※※※※]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）
[※※※※]BAD_ACCESS 在什么情况下出现？
[※※※※※]苹果是如何实现 autoreleasepool 的？

[※※]使用 block 时什么情况会发生引用循环，如何解决？
[※※]在 block 内如何修改 block 外部变量？
[※※※]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？

[※※]GCD 的队列（dispatch_queue_t）分哪两种类型？
[※※※※]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）
[※※※※]dispatch_barrier_async 的作用是什么？
[※※※※※]苹果为什么要废弃 dispatch_get_current_queue？
[※※※※※]以下代码运行结果如何？

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    NSLog(@"1");
    dispatch_sync(dispatch_get_main_queue(), ^{
        NSLog(@"2");
    });
```

```
    NSLog(@"3");
}
```

[⁂]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？
[⁂]如何手动触发一个 value 的 KVO
[⁂]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？
[⁂]KVC 的 keyPath 中的集合运算符如何使用？
[⁂]KVC 和 KVO 的 keyPath 一定是属性么？
[⁂]如何关闭默认的 KVO 的默认实现，并进入自定义的 KVO 实现？
[⁂]apple 用什么方式实现对一个对象的 KVO？

[⁂]IBOutlet 连出来的视图属性为什么可以被设置成 weak？
[⁂]IB 中 User Defined Runtime Attributes 如何使用？

[⁂]如何调试 BAD_ACCESS 错误
[⁂]lldb（gdb）常用的调试命令？

这些小题可以做为讨论的入口，根据面试者的回答再继续聊下去。其中一些题比较底层，是留给屌屌的面试者或者试探评级用的，一般情况并不是重点的考察内容。

业务能力

毕竟平常的工作内容不是 runtime、runloop，不怎么会用到底层的黑魔法，80% 的时间都是和搭建页面、写业务逻辑、网络请求打交道。
要求面试者能够熟练构建 UI，我会找一个面试者做过的页面让他分析下页面结构、约束或者 frame 布局的连法和计算方法；有时也会让面试者说说 UITableView 常用的几个 delegate 和 data source 代理方法，动态 Cell 高度计算什么的；接下来，在手机里随便找一个 App 的页面，让面试者当场说说如果是他写应该用哪些 UI 组件和布局方式等。问几个问题后就能大概了解业务能力了，我们这边重度使用 IB 和 AutoLayout，假如面试者依然使用代码码 UI 也到没关系，有“从良”意愿就很好~

程序架构和一些设计模式如果面试者自己觉得还不错的话也会聊聊，但跪求别说 Singleton 了，用的越多对水平就越表示怀疑。对设计模式自信的我一般问一个问题，抽象工厂模式在 Cocoa SDK 中哪些类中体现？
架构上 MVC 还是 MVVM 还是 MVP 神马的到是可以聊聊各自的见解，反正也没有正确答案，只要别搞的太离谱就行，比如有的人说网络请求和数据库的操作最好放到 UIView 的子类里面干。

网络请求、数据库等各家都有成熟的封装，基本知道咋用就行。除此之外，我们还会顺带的问下除了 iOS 开发外，还会什么其他编程语言、或者熟悉哪种

脚本语言和 Terminal 操作等，甚至还问问是如何翻墙- -，相信这些技能都是很重要的。

性格

大家都是写程序的，没啥必要用奇怪的、很难的问题难为对方，更关键的还是性格，和 Team 的风格是不是和的来。一个心态良好的面试者需要有个平常心，不傲娇也不跪舔，表达要正常，经常遇到问一个问题后一两分钟一直处于沉思状态，一句话不说，交流像挤牙膏一样，很是憋屈；还有非常屌屌的，明明不懂仍然强行据理力争，镇得住面试官也罢，撞枪口上就别怪不客气了- - 。决定要不要一个人基本上聊 5 分钟就可以确定了，喜欢水到渠成的感觉，看对眼了挡都挡不住。

招聘告一段落，后面将会有更精彩的事情发生。最后，再次感谢大家的支持和对我的信任。

超全！iOS 面试题汇总

作者：Job_Yang

之前看了很多面试题，感觉要不是不够就是过于冗余，于是我将网上的一些面试题进行了删减和重排，现在分享给大家。（题目来源于网络，侵删）

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 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：

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

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 – 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?

```
1    NSMutableArray*?ary?=?[[NSMutableArray?array]?retain];
2    NSString?*str?=?[NSString?stringWithFormat:@"test"];
```

```
3     [str?retain];
4     [aryaddObject:str];
5     NSLog(@"%@%d",str,[str?retainCount]);
6     [str?retain];
7     [str?release];
8     [str?release];
9     NSLog(@"%@%d",str,[str?retainCount]);
10    [aryremoveAllObjects];
11    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，

12.如何对 iOS 设备进行性能测试?

答： Profile-> Instruments ->Time Profiler

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：

```
1  public?class?Factory{
2  public?static?Sample?creator(int?which){
3  if?(which==1)
4  return?new?SampleA();
5  else?if?(which==2)
6  return?new?SampleB();
```

```
7    }
8    }
```

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。

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 – 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:
{
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。

```
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]?ins
completion:NULL];
```

51. 写出上面代码的 Block 的定义。

答：

```
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

```
+?(void)transitionWithView:(UIView?*)view?duration:(NSTimeInterval)duration?opti
```

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: 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    typedef?void(^didFinishBlock)?(NSObject?*ob);
2    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—-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

```
1  void?dispatch_async(
2  dispatch_queue_t?queue,
3  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 – 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

```
1  @interface?Controller?:?NSObject?{?NSString?*something;?}
2  +?(void)thisIsAStaticMethod;
3  -?(void)thisIsAnInstanceMethod;
4  @end
5  @interface?Controller?(private)?-
6  (void)thisIsAPrivateMethod;
7  @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？

```
1  const?int?a;
2  int?const?a;
3  const?int?*a;
4  int?*?const?a;
5  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.

iOS,面试必看， 最全梳理

作者 Jack_lin 关注

2016.03.20 22:03* 字数 17243 阅读 43046 评论 115 喜欢 1256 赞赏 3

来自网络

序言

目前形势，参加到 iOS 队伍的人是越来越多，甚至已经到供过于求了。今年，找过工作人可能会更深刻地体会到今年的就业形势不容乐观，加之，培训机构一火车地向用人单位输送 iOS 开发人员，打破了生态圈的动态平衡。矫情一下，言归正传，我奉献一下，为 iOS 应聘者梳理一下面试题，希望能助一臂之力!

OC 的理解与特性

- OC 作为一门面向对象的语言，自然具有面向对象的语言特性：封装、继承、多态。它既具有静态语言的特性（如 C++），又有动态语言的效率（动态绑定、动态加载等）。总体来讲，OC 确实是一门不错的编程语言，
- Objective-C 具有相当多的动态特性，表现为三方面：动态类型（Dynamic typing）、动态绑定（Dynamic binding）和动态加载（Dynamic loading）。动态——必须到运行时（run time）才会做的一些事情。
- 动态类型：即运行时再决定对象的类型，这种动态特性在日常的应用中非常常见，简单来说就是 id 类型。事实上，由于静态类型的固定性和可预知性，

从而使用的更加广泛。静态类型是强类型，而动态类型属于弱类型，运行时决定接受者。

- `动态绑定`：基于动态类型，在某个实例对象被确定后，其类型便被确定了，该对象对应的属性和响应消息也被完全确定。
- `动态加载`：根据需求加载所需要的资源，最基本就是不同机型的适配，例如，在 Retina 设备上加载@2x 的图片，而在老一些的普通苹设备上加载原图，让程序在运行时添加代码模块以及其他资源，用户可根据需要加载一些可执行代码和资源，而不是在启动时就加载所有组件，可执行代码可以含有和程序运行时整合的新类。

简述内存管理基本原则

- 之前：OC 内存管理遵循“谁创建，谁释放，谁引用，谁管理”的机制，当创建或引用一个对象的时候，需要向她发送 alloc、copy、retain 消息，当释放该对象时需要发送 release 消息，当对象引用计数为 0 时，系统将释放该对象，这是 OC 的手动管理机制（MRC）。
- 目前：iOS 5.0 之后引用自动管理机制——自动引用计数（ARC），管理机制与手动机制一样，只是不再需要调用 retain、release、autorelease；它编译时的特性，当你使用 ARC 时，在适当位置插入 release 和 autorelease；它引用 strong 和 weak 关键字，strong 修饰的指针变量指向对象时，当指针指向新值或者指针不复存在，相关联的对象就会自动释放，而 weak 修饰的指针变量指向对象，当对象的拥有者指向新值或者不存在时 weak 修饰的指针会自动置为 nil。
- 如果使用 `alloc`、`copy(mutableCopy)`或者 `retian` 一个对象时,你就有义务,向它发送一条 `release` 或者 `autorelease` 消息。其他方法创建的对象,不需要由你来管理内存。
- 向一个对象发送一条 `autorelease` 消息,这个对象并不会立即销毁，而是将这个对象放入了自动释放池,待池子释放时,它会向池中每一个对象发送 一条 `release` 消息,以此来释放对象.

- 向一个对象发送 `release` 消息,并不意味着这个对象被销毁了,而是当这个对象的引用计数为 0 时,系统才会调用 `dealloc` 方法,释放该对象和对象本身它所拥有的实例。

其他注意事项

- 如果一个对象有一个_strong 类型的指针指向着，找个对象就不会被释放。如果一个指针指向超出了它的作用域，就会被指向 nil。如果一个指针被指向 nil，那么它原来指向的对象就被释放了。当一个视图控制器被释放时，它内部的全局指针会被指向 nil。用法“：不管全局变量还是局部变量用 _strong 描述就行。
- 局部变量：出了作用域，指针会被置为 nil。
- 方法内部创建对象，外部使用需要添加_autorelease;
- 连线的时候，用_weak 描述。
- 代理使用 unsafe_unretained 就相当于 assign；
- block 中为了避免循环引用问题，使用_weak 描述；
- 声明属性时，不要以 `new` 开头。如果非要以 `new` 开头命名属性的名字，需要自己定制 get 方法名，如

```
@property(getter=theString) NSString * newString;
```

- 如果要使用自动释放池，用`@autoreleasepool{}`
- ARC 只能管理 Foundation 框架的变量，如果程序中把 Foundation 中的变量强制换成 COre Foundation 中的变量需要交换管理权；
- 在非 ARC 工程中采用 ARC 去编译某些类：-fobjc-arc。
- 在 ARC 下的工程采用非 ARC 去编译某些类：-fno-fobjc-arc。

如何理解 MVC 设计模式

MVC 是一种架构模式，M 表示 MOdel，V 表示视图 View，C 表示控制器 Controller：

- Model 负责存储、定义、操作数据；
- View 用来展示书给用户，和用户进行操作交互；
- Controller 是 Model 和 View 的协调者，Controller 把 Model 中的数据拿过来给 View 用。Controller 可以直接与 Model 和 View 进行通信，而 View 不能和 Controller 直接通信。View 与 Controller 通信需要利用代理协议的方式，当有数据更新时，MOdel 也要与 Controller 进行通信，这个时候就要用 Notification 和 KVO，这个方式就像一个广播一样，MOdel 发信号，Controller 设置监听接受信号，当有数据更新时就发信号给 Controller，Model 和 View 不能直接进行通信，这样会违背 MVC 设计模式。

如何理解 MVVM 设计模式

- `ViewModel` 层，就是 View 和 Model 层的粘合剂，他是一个放置用户输入验证逻辑，视图显示逻辑，发起网络请求和其他各种各样的代码的极好的地方。说白了，就是把原来 `ViewController` 层的业务逻辑和页面逻辑等剥离出来放到 ViewModel 层。
- View 层，就是 ViewController 层，他的任务就是从 ViewModel 层获取数据，然后显示。
- 如需了解更多，请查看这篇文章

Objective-C 中是否支持垃圾回收机制？

- OC 是支持垃圾回收机制的(`Garbage collection` 简称 `GC`),但是 apple 的移动终端中,是不支持 GC 的,Mac 桌面系统开发中是支持的.
- 移动终端开发是支持 ARC（`Automatic Reference Counting` 的简称）,ARC 是在 IOS5 之后推出的新技术,它与 GC 的机制是不同的。我们在编写代码时,不需要向对象发送 `release` 或者 `autorelease` 方法,也不可以调用 `delloc` 方法,

编译器会在合适的位置自动给用户生成 `release` 消息(`autorelease`),ARC 的特点是自动引用技术简化了内存管理的难度.

协议的基本概念和协议中方法默认为什么类型

OC 中的协议是一个方法列表,且多少有点相关。它的特点是可以被任何类使用(实现),但它并不是类(这里我们需要注意),自身不会实现这样方法, 而是又其他人来实现协议经常用来实现委托对象(委托设计模式)。如果一个类采用了一个协议,那么它必须实现协议中必须需要实现的方法,在协议中的方法默认是必须实现(@required),添加关键字@optional,表明一旦采用该协议,这些“可选”的方法是可以选择不实现的。

简述类目 `category` 优点和缺点

优点：

- 不需要通过增加子类而增加现有类的行为(方法),且类目中的方法与原始类方法基本没有区别;
- 通过类目可以将庞大一个类的方法进行划分,从而便于代码的日后的维护、更新以及提高代码的阅读性;

缺点：

- 无法向类目添加实例变量,如果需要添加实例变量,只能通过定义子类的方式;
- 类目中的方法与原始类以及父类方法相比具有更高优先级,如果覆盖父类的方法,可能导致 `super` 消息的断裂。因此,最好不要覆盖原始类中的方法。

类别的作用

- 给系统原有类添加方法，不能扩展属性。如果类别中方法的名字跟系统的方法名一样，在调用的时候类别中的方法优先级更高；
- 分散类的实现：如：

```
+ (NSIndexPath *)indexPathForRow:(NSInteger)row
```

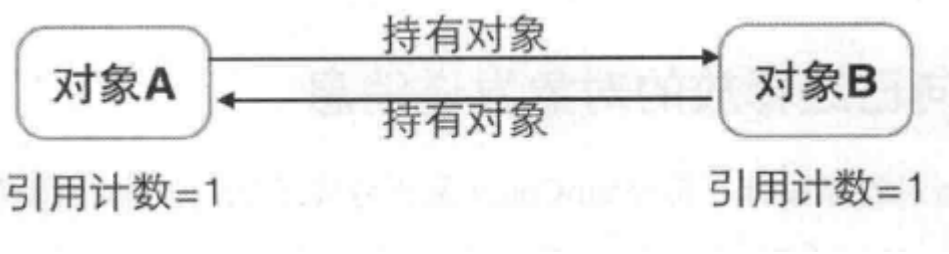
```
inSection:(NSInteger)section
```

原本属于 NSIndexPath 的方法，但因为这个方法经常使用的表的时候调用、跟表的关系特别密切，因此把这个方法一类别的形式、声明在 UITableView.h 中。

- 声明私有方法，某一个方法只实现，不声明，相当于私有方法。
- 类别不能声明变量，类别不可以直接添加属性。`property` 描述 setter 方法，就不会报错。

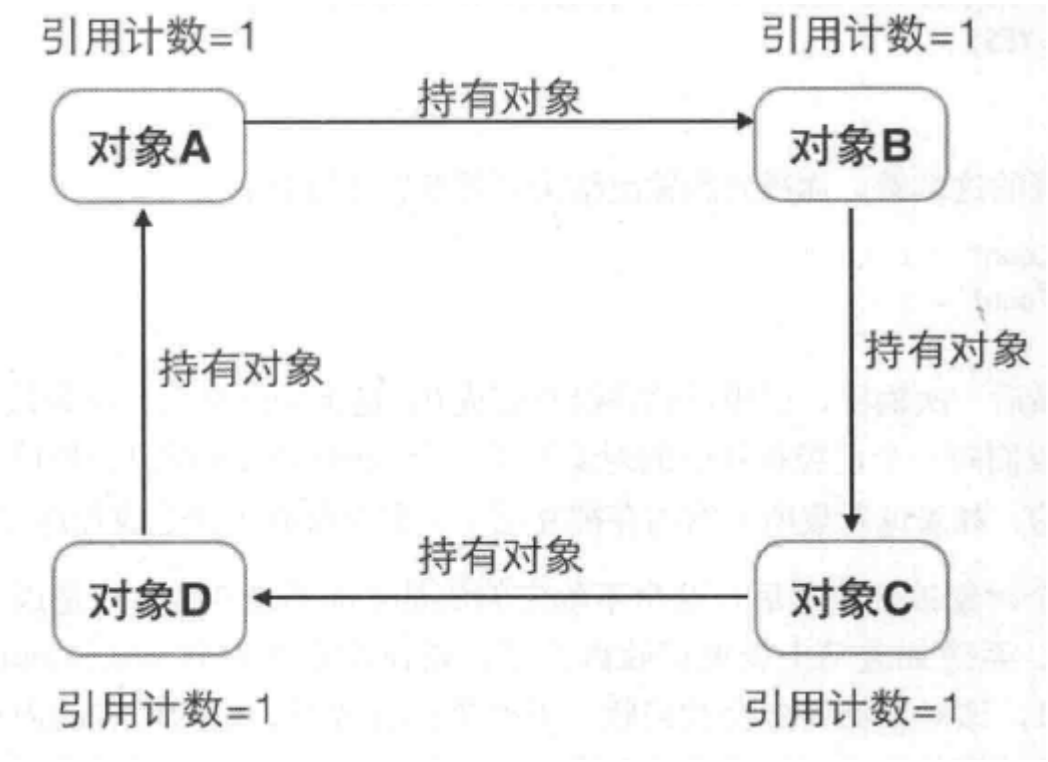
循环引用的产生原因，以及解决方法

- 产生原因：如下图所示，对象 A 和对象 B 相互引用了对方作为自己的成员变量，只有自己销毁的时候才能将成员变量的引用计数减 1。对象 A 的销毁依赖于对象 B 的销毁，同时对象 B 销毁也依赖与对象 A 的销毁，从而形成**循环引用**，此时，即使外界没有任何指针访问它，它也无法释放。



循环引用示例图

多个对象间依然会存在循环引用问题，形成一个环，在编程中，形成的环越大越不容易察觉，如下图所示：



多个对象引用示例图

解决方法：

- 事先知道存在循环引用的地方，在合理的位置主动断开一个引用，是对象回收；
- 使用弱引用的方法。

键路径(keyPath)、键值编码（KVC）、键值观察（KVO）

键路径

- 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。

键值编码 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，非对象类型的变量将被自动封装或者解封成对象，很多情况下会简化程序代码；
- KVC 的缺点：一旦使用 KVC 你的编译器无法检查出错误,即不会对设置的键、键路径进行错误检查,且执行效率要低于合成存取器方法和自定的 setter 和 getter 方法。因为使用 KVC 键值编码,它必须先解析字符串,然后在设置或者访问对象的实例变量。

键值观察 KVO

- 键值观察机制是一种能使得对象获取到其他对象属性变化的通知 ， 极大的简化了代码。
- 实现 KVO 键值观察模式,被观察的对象必须使用 KVC 键值编码来修 改它的实例变量,这样才能被观察者观察到。因此,KVC 是 KVO 的基础。

Demo

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。

KVC 机制通过 `key` 找到 `value` 的原理

- 当通过 KVC 调用对象时，比如：`[self valueForKey:@"someKey"]`时，程序会自动试图通过下面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。
- 补充：KVC 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_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。

在 Objective-C 中如何实现 KVO

- 注册观察者(注意：观察者和被观察者不会被保留也不会被释放)

```
- (void)addObserver:(NSObject *)observer forKeyPath:(NSString *)keyPath
options:(NSKeyValueObservingOptions)options
```

```
context:(void *)context;
```

- 接收变更通知

```
- (void)observeValueForKeyPath:(NSString *)keyPath
```

```
ofObject:(id)object change:(NSDictionary *)change  context:(void
*)context;
```

- 移除对象的观察者身份

```
- (void)removeObserver:(NSObject *)observer
```

```
forKeyPath:(NSString *)keyPath;
```

- KVO 中`谁要监听谁注册`，然后对响应进行处理，使得观察者与被观察者完全解耦。KVO 只检测类中的属性，并且属性名都是通过 NSString 来查找，编译器不会检错和补全，全部取决于自己。

代理的作用

- 代理又叫委托，是一种设计模式，代理是对象与对象之间的通信交互，代理解除了对象之间的耦合性。
- 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。
- 代理的属性常是`assign`的原因：防止循环引用,以至对象无法得到正确的释放。

NSNotification、Block、Delegate 和 KVO 的区别

- 代理是一种回调机制，且是一对一的关系，通知是一对多的关系，一个对向所有的观察者提供变更通知；
- 效率：Delegate 比 NSNOtification 高；
- Delegate 和 Block 一般是一对一的通信；
- Delegate 需要定义协议方法，代理对象实现协议方法，并且需要建立代理关系才可以实现通信；
- Block：Block 更加简洁，不需要定义繁琐的协议方法，但通信事件比较多的话，建议使用 Delegate；

Objective-C 中可修改和不可以修改类型

- 可修改不可修改的集合类，就是可动态添加修改和不可动态添加修改。
- 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

当我们调用一个静态方法时,需要对对象进行 release 吗?

- 不需要,静态方法(类方法)创建一个对象时,对象已被放入自动释放池。在自动释放池被释放时,很有可能被销毁。

当我们释放我们的对象时,为什么需要调用[super dealloc]方法,它的位置又是如何的呢?

- 因为子类的某些实例是继承自父类的,因此需要调用`[super dealloc]`方法，来释放父类拥有的实例,其实也就是子类本身的。一般来说我们优先释放子类拥 有的实例,最后释放父类所拥有的实例。

对谓词的认识

- Cocoa 中提供了一个 `NSPredicate` 的类,该类主要用于指定过滤器的条件，每一个对象通过谓词进行筛选,判断条件是否匹配。如果需要了解使用方法，请看谓词的具体使用

static、self、super 关键字的作用

- 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.
- 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.
- 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.
- 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.
- self:当前消息的接收者。
- super:向父类发送消息。

#include 与#import 的区别、#import 与@class 的区别

- `#include` 和`#import` 其效果相同,都是查询类中定义的行为(方法);
- `#import` 不会引起交叉编译,确保头文件只会被导入一次 ；
- `@class` 的表明,只定 义了类的名称,而具体类的行为是未知的,一般用于.h 文件 ；
- `@class` 比`#import` 编译效率更高。
- 此外`@class` 和`#import` 的主要区别在于解决引用死锁的问题。

@public、@protected、@private 它们的含义与作用

- `@public`:对象的实例变量的作用域在任意地方都可以被访问 ；

- `@protected`:对象的实例变量作用域在本类和子类都可以被访问 ；
- `@private`:实例变量的作用域只能在本类(自身)中访问 .

解释 id 类型

任意类型对象，程序运行时才决定对象的类型。

switch 语句 if 语句区别与联系

均表示条件的判断,switch 语句表达式只能处理的是整型、字符型和枚举类型,而选择流程语句则没有这样的限制。但 switch 语句比选择流程控制语句效率更高。

isMemberOfClass 和 isKindOfClass 联系与区别

- 联系：两者都能检测一个对象是否是某个类的成员
- 区别：`isKindOfClass` 不仅用来确定一个对象是否是一个类的成员,也可以用来确定一个对象是否派生自该类的类的成员 ,而 `isMemberOfClass` 只能做到第一点。
- 举例：如 `ClassA` 派 生 自 `NSObject` 类 , `ClassA *a = [ClassA alloc] init];`,`[a isKindOfClass:[NSObject class]]` 可以检查出 a 是否是 `NSObject` 派生类 的成员,但 `isMemberOfClass` 做不到。

iOS 开发中数据持久性有哪几种?

数据存储的核心都是写文件。

- 属性列表：只有 NSString、NSArray、NSDictionary、NSData 可 writeToFile；存储依旧是 plist 文件。plist 文件可以存储的 7 中数据类型：array、dictionary、string、bool、data、date、number。
- 对象序列化（对象归档）：对象序列化通过序列化的形式，键值关系存储到本地，转化成二进制流。通过 runtime 实现自动化归档/解档，请参考这个文章。实现 NSCoding 协议必须实现的两个方法：

1.编码（对象序列化）：把不能直接存储到 plist 文件中得到数据，转化为二进制数据，NSData，可以存储到本地；

2.解码（对象反序列化）：把二进制数据转化为本来的类型。

- SQLite 数据库：大量有规律的数据使用数据库。
- CoreData ：通过管理对象进行增、删、查、改操作的。它不是一个数据库，不仅可以使用 SQLite 数据库来保持数据，也可以使用其他的方式来存储数据。如：XML。

CoreData 的介绍：

- CoreData 是面向对象的 API，CoreData 是 iOS 中非常重要的一项技术，几乎在所有编写的程序中，CoreData 都作为数据存储的基础。
- CoreData 是苹果官方提供的一套框架，用来解决与对象声明周期管理、对象关系管理和持久化等方面相关的问题。
- 大多数情况下，我们引用 CoreData 作为持久化数据的解决方案，并利用它作为持久化数据映射为内存对象。提供的是对象-关系映射功能，也就是说，CoreData 可以将 Objective-C 对象转换成数据，保存到 SQL 中，然后将保存后的数据还原成 OC 对象。

CoreData 的特征：

- 通过 CoreData 管理应用程序的数据模型，可以极大程度减少需要编写的代码数量。
- 将对象数据存储在 SQLite 数据库已获得性能优化。
- 提供 NSFetchResultsController 类用于管理表视图的数据，即将 Core Data 的持久化存储在表视图中，并对这些数据进行管理：增删查改。
- 管理 undo/redo 操纵；
- 检查托管对象的属性值是否正确。

Core Data 的 6 成员对象

- 1.**NSManageObject**:被管理的数据记录 Managed Object Model 是描述应用程序的数据模型，这个模型包含实体（Entity）、特性（Property）、读取请求（Fetch Request）等。
- 2.**NSManageObjectContext**：管理对象上下文，持久性存储模型对象，参与数据对象进行各种操作的全过程，并监测数据对象的变化，以提供对 undo/redo 的支持及更新绑定到数据的 UI。
- 3.**NSPersistentStoreCoordinator**:连接数据库的 Persistent Store Coordinator 相当于数据文件管理器，处理底层的对数据文件的读取和写入，一般我们与这个没有交集。
- 4.**NSManagedObjectModel**：被管理的数据模型、数据结构。
- 5.**NSFetchRequest**：数据请求；
- 6.**NSEntityDescription**：表格实体结构，还需知道.xcdatamodel 文件编译后为.momd 或者.mom 文件。

Core Data 的功能

- 对于 KVC 和 KVO 完整且自动化的支持，除了为属性整合 KVO 和 KVC 访问方法外，还整合了适当的集合访问方法来处理多值关系；
- 自动验证属性（property）值；
- 支持跟踪修改和撤销操作；
- 关系维护，Core Data 管理数据的关系传播，包括维护对象间的一致性；
- 在内存上和界面上分组、过滤、组织数据；
- 自动支持对象存储在外部数据仓库的功能；
- 创建复杂请求：无需动手写 SQL 语句，在获取请求（fetch request）中关联 NSPredicate。NSPreadicate 支持基本功能、相关子查询和其他高级的 SQL 特性。它支持正确的 Unicode 编码、区域感知查询、排序和正则表达式；

- 延迟操作：Core Data 使用懒加载（lazy loading）方式减少内存负载，还支持部分实体化延迟加载和复制对象的数据共享机制；
- 合并策略：Core Data 内置版本跟踪和乐观锁（optimistic locking）来支持多用户写入冲突的解决，其中，乐观锁就是对数据冲突进行检测，若冲突就返回冲突的信息；
- 数据迁移：Core Data 的 Schema Migration 工具可以简化应对数据库结构变化的任务，在某些情况允许你执行高效率的数据库原地迁移工作；
- 可选择针对程序 Controller 层的集成，来支持 UI 的显示同步 Core Data 在 IPhone OS 之上，提供 NSFetchedResultsController 对象来做相关工作，在 Mac OS X 上我们用 Cocoa 提供的绑定（Binding）机制来完成的。

对象可以被 copy 的条件

- 只有实现了 `NSCopying` 和 `NSMutableCopying` 协议的类的对象才能被拷贝,分为不可变拷贝和可变拷贝,具体区别戳这；
- `NSCopying` 协议方法为：

```
- (id)copyWithZone:(NSZone *)zone {
 MyObject *copy = [[[self class] allocWithZone: zone] init];
 copy.username = [self.username copyWithZone:zone];
 return copy;
}
```

-

自动释放池工作原理

- 自动释放池是`NSAutorelease`类的一个实例,当向一个对象发送`autorelease`消息时,该对象会自动入池,待池销毁时,将会向池中所有对象发送一条`release`消息,释放对象。
- `[pool release]、 [pool drain]`表示的是池本身不会销毁,而是池子中的临时对象都被发送`release`,从而将对象销毁。

在某个方法中 self.name = _name， name = _name 它 们有区别吗,为什么?

- 前者是存在内存管理的 setter 方法赋值,它会对_name 对象进行保留或者拷贝操作
- 后者是普通赋值
- 一般来说，在对象的方法里成员变量和方法都是可以访问的，我们通常会重写 Setter 方法来执行某些额外的工作。比如说，外部传一个模型过来，那么我会直接重写 Setter 方法，当模型传过来时，也就是意味着数据发生了变化，那么视图也需要更新显示，则在赋值新模型的同时也去刷新 UI。

解释 self = [super init]方法

- 容错处理,当父类初始化失败,会返回一个 nil,表示初始化失败。由于继承的关系,子类是需要拥有父类的实例和行为,因此,我们必须先初始化父类,然后再初始化子类

定义属性时,什么时候用 assign、retain、copy 以及它们的之间的区别

- `assign`:普通赋值,一般常用于基本数据类型,常见委托设计模式, 以此来防止循环引用。(我们称之为弱引用).
- `retain`:保留计数,获得到了对象的所有权,引用计数在原有基础上加 1.

- `copy`:一般认为,是在内存中重新开辟了一个新的内存空间,用来 存储新的对象,和原来的对象是两个不同的地址,引用计数分别为 1。但是当 `copy` 对象为不可变对象时,那么 `copy` 的作用相当于 `retain`。因为,这样可以节约内存空间

堆和栈的区别

- `栈区(stack)`由编译器自动分配释放 ,存放方法(函数)的参数值, 局部变量的值等, 栈是向低地址扩展的数据结构, 是一块连续的内存的区域。即栈顶的地址和栈的最大容量是系统预先规定好的。
- `堆区(heap)`一般由程序员分配释放, 若程序员不释放,程序结束时由 OS 回收, 向高地址扩展的数据结构, 是不连续的内存区域, 从而堆获得的空间比较灵活。
- `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。
- `全局区(静态区)(static)`,全局变量和静态变量的存储是放在一块 的,初始化的全局变量和静态变量在一块区域, 未初始化的全局变量和未初始化的静态变量在相邻的另一块区域。程序结束后有系统释放。
- `文字常量区`—常量字符串就是放在这里的。程序结束后由系统释放。
- `程序代码区`—存放函数体的二进制代码

怎样使用 performSelector 传入 3 个以上参数，其中一个为结构体

- 因为系统提供的 `performSelector` 的 API 中，并没有提供三个参数。因此，我们只能传数组或者字典，但是数组或者字典只有存入对象类型，而结构体并不是对象类型,我们只能通过对象放入结构作为属性来传过去了.

```
- (id)performSelector:(SEL)aSelector;
- (id)performSelector:(SEL)aSelector withObject:(id)object;
- (id)performSelector:(SEL)aSelector withObject:
```

```
(id)object1 withObject:(id)object2;
```

- 具体实现如下：

```
typedef struct HYBStruct {
int a;
int b;
} *my_struct;

@interface HYBObject : NSObject

@property (nonatomic, assign) my_struct arg3;
@property (nonatomic, copy)  NSString *arg1;
@property (nonatomic, copy) NSString *arg2;

```

```
@end
@implementation HYBObject

// 在堆上分配的内存，我们要手动释放掉
- (void)dealloc {
free(self.arg3);
}

```

```
@end
```

- 测试：

```
my_struct str = (my_struct)(malloc(sizeof(my_struct)));
str->a = 1;
str->b = 2;
HYBObject *obj = [[HYBObject alloc] init];
obj.arg1 = @"arg1";
obj.arg2 = @"arg2";
obj.arg3 = str;
[self performSelector:@selector(call:) withObject:obj];
// 在回调时得到正确的数据的
- (void)call:(HYBObject *)obj {
 NSLog(@"%d %d", obj.arg3->a, obj.arg3->b);
```

```
}
```

UITableViewCell 上有个 UILabel，显示 NSTimer 实现的秒表时间，手指滚动 cell 过程中，label 是否刷新，为什么？

这是否刷新取决于 timer 加入到 Run Loop 中的 Mode 是什么。Mode 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 滑动时会切换到该 Mode
- `UIInitializationRunLoopMode`：run loop 启动时，会切换到该 mode
- `NSRunLoopCommonModes（kCFRunLoopCommonModes）`：Mode 集合
  苹果公开提供的 Mode 有两个：
- `NSDefaultRunLoopMode（kCFRunLoopDefaultMode）`
- `NSRunLoopCommonModes（kCFRunLoopCommonModes）`
- 在编程中：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。当我们滚动的时候，也希望不调度，那就应该使用默认模式。但是，如果希望在滚动时，定时器也要回调，那就应该使用 common mode。

对于单元格重用的理解

- 当屏幕上滑出屏幕时，系统会把这个单元格添加到重用队列中，等待被重用，当有新单元从屏幕外滑入屏幕内时，从重用队列中找看有没有可以重用的单元格，若有，就直接用，没有就重新创建一个。

解决 cell 重用的问题

- 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示（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复用。对于不同种类的单元格使用不同的 ID，对于简单的表格，一个标示符就够了。
- 如一个 TableView 中有 10 个单元格，但屏幕最多显示 4 个，实际上 iPhone 只为其分配 4 个单元格的内存，没有分配 10 个，当滚动单元格时，屏幕内显示的单元格重复使用这 4 个内存。实际上分配的 cell 的个数为屏幕最大显示数，当有新的 cell 进入屏幕时，会随机调用已经滚出屏幕的 Cell 所占的内存，这就是 Cell 的重用。
- 对于多变的自定义 Cell，这种重用机制会导致内容出错，为解决这种出错的方法，把原来的

```
UITableViewCell *cell = [tableview
dequeueReusableCellWithIdentifier:defineString]
```

```
修改为：UITableViewCell *cell = [tableview
cellForRowAtIndexPath:indexPath];
```

这样就解决掉 cell 重用机制导致的问题。

有 a、b、c、d 4 个异步请求，如何判断 a、b、c、d 都完成执行？如果需要 a、b、c、d 顺序执行，该如何实现？

- 对于这四个异步请求，要判断都执行完成最简单的方式就是通过 GCD 的 group 来实现：

```
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, ^{ /*任务 a */ });

dispatch_group_async(group, queue, ^{ /*任务 b */ });
```

```
dispatch_group_async(group, queue, ^{ /*任务 c */ });
dispatch_group_async(group, queue, ^{ /*任务 d */ });
dispatch_group_notify(group,dispatch_get_main_queue(), ^{
 // 在 a、b、c、d 异步执行完成后，会回调这里
});
```

- 当然，我们还可以使用非常老套的方法来处理，通过四个变量来标识 a、b、c、d 四个任务是否完成，然后在 runloop 中让其等待，当完成时才退出 runloop。但是这样做会让后面的代码得不到执行，直到 Run loop 执行完毕。
- 解释：要求顺序执行，那么可以将任务放到串行队列中，自然就是按顺序来异步执行了。

使用 block 有什么好处？使用 NSTimer 写出一个使用 block 显示（在 UILabel 上）秒表的代码

- 代码紧凑，传值、回调都很方便，省去了写代理的很多代码。
- NSTimer 封装成的 block，具体实现
- 实现方法：

```
NSTimer *timer = [NSTimer scheduledTimerWithTimeInterval:1.0
                         repeats:YES
                        callback:^() {
  weakSelf.secondsLabel.text = ...
}
```

```
[[NSRunLoop currentRunLoop] addTimer:timer forMode:NSRunLoopCommonModes];
```

一个 view 已经初始化完毕，view 上面添加了 n 个 button，除用 view 的 tag 之外，还可以采用什么办法来找到自己想要的 button 来修改 button 的值

有 2 种方法解决：

- 第一种：如果是点击某个按钮后，才会刷新它的值，其它不用修改，那么不用引用任何按钮，直接在回调时，就已经将接收响应的按钮给传过来了，直接通过它修改即可。
- 第二种：点击某个按钮后，所有与之同类型的按钮都要修改值，那么可以通过在创建按钮时将按钮存入到数组中，在需要的时候遍历查找。

线程与进程的区别和联系？

- 一个程序至少要有进城,一个进程至少要有一个线程.
- 进程:资源分配的最小独立单元,进程是具有一定独立功能的程序关于某个数据集合上的一次运行活动,进程是系统进行资源分配和调度的一个独立单位.
- 线程:进程下的一个分支,是进程的实体,是 CPU 调度和分派的基本单元,它是比进程更小的能独立运行的基本单位,线程自己基本不拥有系统资源,只拥有一点在运行中必不可少的资源(程序计数器、一组寄存器、栈)，但是它可与同属一个进程的其他线程共享进程所拥有的全部资源。
- 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。
- 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。进程有独立的地址空间，一个进程崩溃后，在保护模式下不会对其它进程产生影响，而线程只是一个进程中的不同执行路径。线程有自己的堆栈和局部变量，但线程之间没有单独的地址空间，一个线程死掉就等于整个进程死掉，所以多

进程的程序要比多线程的程序健壮，但在进程切换时，耗费资源较大，效率要差一些。

- 但对于一些要求同时进行并且又要共享某些变量的并发操作，只能用线程，不能用进程。

多线程编程

- `NSThread`:当需要进行一些耗时操作时会把耗时的操作放到线程中。线程同步：多个线程同时访问一个数据会出问题，NSlock、线程同步块、@synchronized(self){}。
- `NSOperationQueue 操作队列`（不需考虑线程同步问题）。编程的重点都放在 main 里面，`NSInvocationOperation`、`BSBlockOperation`、自定义 Operation。创建一个操作绑定相应的方法，当把操作添加到操作队列中时，操作绑定的方法就会自动执行了，当把操作添加到操作队列中时，默认会调用 main 方法。
- GCD（`Grand Central Dispatch）宏大的中央调度，串行队列、并发队列、主线程队列；
- 同步和异步：同步指第一个任务不执行完，不会开始第二个，异步是不管第一个有没有执行完，都开始第二个。
- 串行和并行：串行是多个任务按一定顺序执行，并行是多个任务同时执行；
- 代码是在分线程执行，在主线程嘟列中刷新 UI。

多线程编程是防止主线程堵塞、增加运行效率的最佳方法。

- Apple 提供了 NSOperation 这个类，提供了一个优秀的多线程编程方法；
- 一个 NSOperationQueue 操作队列，相当于一个线程管理器，而非一个线程，因为你可以设置这个线程管理器内可以并行运行的线程数量等。
- 多线程是一个比较轻量级的方法来实现单个应用程序内多个代码执行路径。

- iPhoneOS 下的主线程的堆栈大小是 1M。第二个线程开始就是 512KB，并且该值不能通过编译器开关或线程 API 函数来更改，只有主线程有直接修改 UI 的能力。

定时器与线程的区别;

- 定时器;可以执行多次，默认在主线程中。
- 线程：只能执行一次。

Apple 设备尺寸和编程尺寸

设备	系统	分辨率	屏幕尺寸	倍数
iPhone				
iPhone 2GS	IOS 1	320*480	3.5英寸	1x
iPhone 3G	IOS 2	320*480	3.5英寸	1x
iPhone 3GS	IOS 3	320*480	3.5英寸	1x
iPhone 4	IOS 4	320*480	3.5英寸	2x
iPhone 4s	IOS 5	320*480	3.5英寸	2x
iPhone 5	IOS 6	320*568	4.0英寸	2x
iPhone 5s/c	IOS 7	320*568	4.0英寸	2x
iPhone 6	IOS 8	375*667	4.7英寸	2x
iPhone 6plus	IOS 8	414*736	5.5英寸	3x

iPhone 设备

- **IPod Touch**
- IPod Touch1G IOS 1 320*480 3.5英寸 1x
- IPod Touch2G IOS 2 320*480 3.5英寸 1x
- IPod Touch3G IOS 3 320*480 3.5英寸 1x
- IPod Touch4G IOS 4 320*480 3.5英寸 2x

IPod 设备

- **iPad**
- iPad IOS 3 1024*768 9.7英寸 1x
- iPad2 IOS 4 1024*768 9.7英寸 1x
- iPad3(New iPad) IOS 5 1024*768 9.7英寸 2x
- iPad4 IOS 6 1024*768 9.7英寸 2x
- iPad Air IOS 7 1024*768 9.7英寸 2x
- iPad Air2 IOS 8 1024*768 9.7英寸 2x
- **iPad mini**
- iPad mini IOS6 1024*768 7.9英寸 1x
- iPad mini2 IOS7 1024*768 7.9英寸 2x
- iPad mini3 IOS8 1024*768 7.9英寸 2x

iPad 设备

TCP 和 UDP 的区别于联系

- TCP 为传输控制层协议，为面向连接、可靠的、点到点的通信；
- UDP 为用户数据报协议，非连接的不可靠的点到多点的通信；
- TCP 侧重可靠传输，UDP 侧重快速传输。

TCP 连接的三次握手

- 第一次握手：客户端发送 syn 包（syn=j）到服务器，并进入 SYN_SEND 状态，等待服务器确认；
- 第二次握手：服务器收到 syn 包，必须确认客户的 SYN（ack=j+1），同时自己也发送一个 SYN 包，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），此发送完毕，客户端和服务器进入 ESTABLISHED 状态，完成三次状态。

Scoket 连接和 HTTP 连接的区别：

- HTTP 协议是基于 TCP 连接的，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：短连接，客户端向服务器发送一次请求，服务器响应后连接断开，节省资源。服务器不能主动给客户端响应（除非采用 HTTP 长连接技术），iPhone 主要使用类 NSURLConnection。
- Socket 连接：长连接，客户端跟服务器端直接使用 Socket 进行连接，没有规定连接后断开，因此客户端和服务器段保持连接通道，双方可以主动发送数据，一般多用于游戏.Socket 默认连接超时时间是 30 秒，默认大小是 8K（理解为一个数据包大小）。

HTTP 协议的特点，关于 HTTP 请求 GET 和 POST 的区别

`GET 和 POST 的区别`：

- HTTP 超文本传输协议，是短连接，是客户端主动发送请求，服务器做出响应，服务器响应之后，链接断开。HTTP 是一个属于应用层面向对象的协议，HTTP 有两类报文：请求报文和响应报文。

- HTTP 请求报文：一个 HTTP 请求报文由请求行、请求头部、空行和请求数据 4 部分组成。
- HTTP 响应报文：由三部分组成：状态行、消息报头、响应正文。
- GET 请求：参数在地址后拼接，没有请求数据，不安全（因为所有参数都拼接在地址后面），不适合传输大量数据（长度有限制，为 1024 个字节）。

> - GET 提交、请求的数据会附在 URL 之后，即把数据放置在 HTTP 协议头<requestline>中。
> - 以？分割 URL 和传输数据，多个参数用&连接。如果数据是英文字母或数字，原样发送，
>
> 如果是空格，转换为+，如果是中文/其他字符，则直接把字符串用 BASE64 加密。

- POST 请求：参数在请求数据区放着，相对 GET 请求更安全，并且数据大小没有限制。把提交的数据放置在 HTTP 包的包体`<request-body>`中.
- GET 提交的数据会在地址栏显示出来，而 POST 提交，地址栏不会改变。

传输数据的大小：

- GET 提交时，传输数据就会受到 URL 长度限制，POST 由于不是通过 URL 传值，理论上书不受限。

安全性：

- POST 的安全性要比 GET 的安全性高；
- 通过 GET 提交数据，用户名和密码将明文出现在 URL 上，比如登陆界面有可能被浏览器缓存。
- HTTPS：安全超文本传输协议（`Secure Hypertext Transfer Protocol`），它是一个安全通信通道，基于 HTTP 开发，用于客户计算机和服务器之间交换信息，使用安全套结字层（`SSI`）进行信息交换，即 HTTP 的安全版。

ASIHttpRequest、AFNetWorking 之间的区别

- ASIHttpRequest 功能强大，主要是在 MRC 下实现的，是对系统 CFNetwork API 进行了封装，支持 HTTP 协议的 CFHTTP，配置比较复杂，并且 ASIHttpRequest 框架默认不会帮你监听网络改变，如果需要让 ASIHttpRequest 帮你监听网络状态改变，并且手动开始这个功能。
- AFNetWorking 构建于 NSURLConnection、NSOperation 以及其他熟悉的 Foundation 技术之上。拥有良好的架构，丰富的 API 及模块构建方式，使用起来非常轻松。它基于 NSOperation 封装的，AFURLConnectionOperation 子类。
- ASIHttpRequest 是直接操作对象 ASIHttpRequest 是一个实现了 NSCoding 协议的 NSOperation 子类；AFNetWorking 直接操作对象的 AFHttpClient，是一个实现 NSCoding 和 NSCopying 协议的 NSObject 子类。
- 同步请求：ASIHttpRequest 直接通过调用一个 `startSynchronous` 方法；AFNetWorking 默认没有封装同步请求，如果开发者需要使用同步请求，则需要重写 `getPath:paraments:success:failures` 方法，对于 AFHttpRequestOperation 进行同步处理。
- 性能对比：AFNetworking 请求优于 ASIHttpRequest；

XML 数据解析方式各有什么不同，JSON 解析有哪些框架？

- XML 数据解析的两种解析方式：DOM 解析和 SAX 解析；
- DOM 解析必须完成 DOM 树的构造，在处理规模较大的 XML 文档时就很耗内存，占用资源较多，读入整个 XML 文档并构建一个驻留内存的树结构（节点树），通过遍历树结构可以检索任意 XML 节点，读取它的属性和值，通常情况下，可以借助 XPath 查询 XML 节点；
- SAX 与 DOM 不同，它是事件驱动模型，解析 XML 文档时每遇到一个开始或者结束标签、属性或者一条指令时，程序就产生一个事件进行相应的处

理，一边读取 XML 文档一边处理，不必等整个文档加载完才采取措施，当在读取解析过程中遇到需要处理的对象，会发出通知进行处理。因此，SAX 相对于 DOM 来说更适合操作大的 XML 文档。

-JSON 解析：性能比较好的主要是第三方的 JSONKIT 和 iOS 自带的 JSON 解析类，其中自带的 JSON 解析性能最高，但只能用于 iOS5 之后。

如何进行真机调试

- 1.首先需要用钥匙串创建一个钥匙（key）；
- 2.将钥匙串上传到官网，获取 iOS Development 证书；
- 3.创建 App ID 即我们应用程序中的 Boundle ID；
- 4.添加 Device ID 即 UDID；
- 5.通过勾选前面所创建的证书：App ID、Device ID；
- 6.生成 mobileprovision 文件；
- 7.先决条件：申请开发者账号 99 美刀

APP 发布的上架流程

- 1.登录应用发布网站添加应用信息；
- 2.下载安装发布证书；
- 3.选择发布证书，使用 Archive 编译发布包，用 Xcode 将代码（发布包）上传到服务器；
- 4.等待审核通过;
- 5.生成 IPA：菜单栏->Product->Archive.

SVN 的使用

- SVN=版本控制+备份服务器，可以把 SVN 当成备份服务器，并且可以帮助你记住每次上服务器的档案内容，并自动赋予每次变更的版本；

- SVN 的版本控制：所有上传版本都会帮您记录下来，也有版本分支及合并等功能。SVN 可以让不同的开发者存取同样的档案，并且利用 SVN Server 作为档案同步的机制，即您有档案更新时，无需将档案寄送给您的开发成员。SVN 的存放档案方式是采用差异备份的方式，即会备份到不同的地方，节省硬盘空间，也可以对非文字文件进行差异备份。
- SVN 的重要性：备份工作档案的重要性、版本控管的重要性、伙伴间的数据同步的重要性、备份不同版本是很耗费硬盘空间的；
- 防止冲突：
  1.防止代码冲突：不要多人同时修改同一文件，例如：A、B 都修改同一个文件，先让 A 修改，然后提交到服务器，然后 B 更新下来，再进行修改；
  2.服务器上的项目文件 Xcodeproj，仅让一个人管理提交，其他人只更新，防止文件发生冲突。

如何进行网络消息推送

- 一种是 Apple 自己提供的通知服务（APNS 服务器）、一种是用第三方推送机制。
- 首先应用发送通知，系统弹出提示框询问用户是否允许，当用户允许后向苹果服务器(APNS)请求 deviceToken，并由苹果服务器发送给自己的应用，自己的应用将 DeviceToken 发送自己的服务器，自己服务器想要发送网络推送时将 deviceToken 以及想要推送的信息发送给苹果服务器，苹果服务器将信息发送给应用。
- 推送信息内容，总容量不超过 256 个字节；
- iOS SDK 本身提供的 APNS 服务器推送，它可以直接推送给目标用户并根据您的方式弹出提示。
  优点：不论应用是否开启，都会发送到手机端；
  缺点：消息推送机制是苹果服务端控制，个别时候可能会有延迟，因为苹果服务器也有队列来处理所有的消息请求；

- 第三方推送机制，普遍使用 Socket 机制来实现，几乎可以达到即时的发送到目标用户手机端，适用于即时通讯类应用。
  优点：实时的，取决于心跳包的节奏；
  缺点：iOS 系统的限制，应用不能长时间的后台运行，所以应用关闭的情况下这种推送机制不可用。

网络七层协议

- **应用层**：
  1.用户接口、应用程序；
  2.Application 典型设备：网关；
  3.典型协议、标准和应用：TELNET、FTP、HTTP
- **表示层**：
  1.数据表示、压缩和加密 presentation
  2.典型设备：网关
  3.典型协议、标准和应用：ASCLL、PICT、TIFF、JPEG|MPEG
  4.表示层相当于一个东西的表示，表示的一些协议，比如图片、声音和视频 MPEG。
- **会话层**：
  1.会话的建立和结束；
  2.典型设备：网关；
  3.典型协议、标准和应用：RPC、SQL、NFS、X WINDOWS、ASP
- **传输层**：
  1.主要功能：端到端控制 Transport；
  2.典型设备：网关；
  3.典型协议、标准和应用：TCP、UDP、SPX
- **网络层**：
  1.主要功能：路由、寻址 Network；
  2.典型设备：路由器；
  3.典型协议、标准和应用：IP、IPX、APPLETALK、ICMP；

- **数据链路层**：

  1.主要功能：保证无差错的疏忽链路的 data link；

  2.典型设备：交换机、网桥、网卡；

  3.典型协议、标准和应用：802.2、802.3ATM、HDLC、FRAME RELAY；

- **物理层**：

  1.主要功能：传输比特流 Physical；

  2.典型设备：集线器、中继器

  3.典型协议、标准和应用：V.35、EIA/TIA-232.

对 NSUserDefaults 的理解

- NSUserDefaults：系统提供的一种存储数据的方式，主要用于保存少量的数据，默认存储到 library 下的 Preferences 文件夹。

SDWebImage 原理

调用类别的方法：

- 从内存中（字典）找图片（当这个图片在本次程序加载过），找到直接使用；
- 从沙盒中找，找到直接使用，缓存到内存。
- 从网络上获取，使用，缓存到内存，缓存到沙盒。

OC 中是否有二维数组，如何实现二维数组

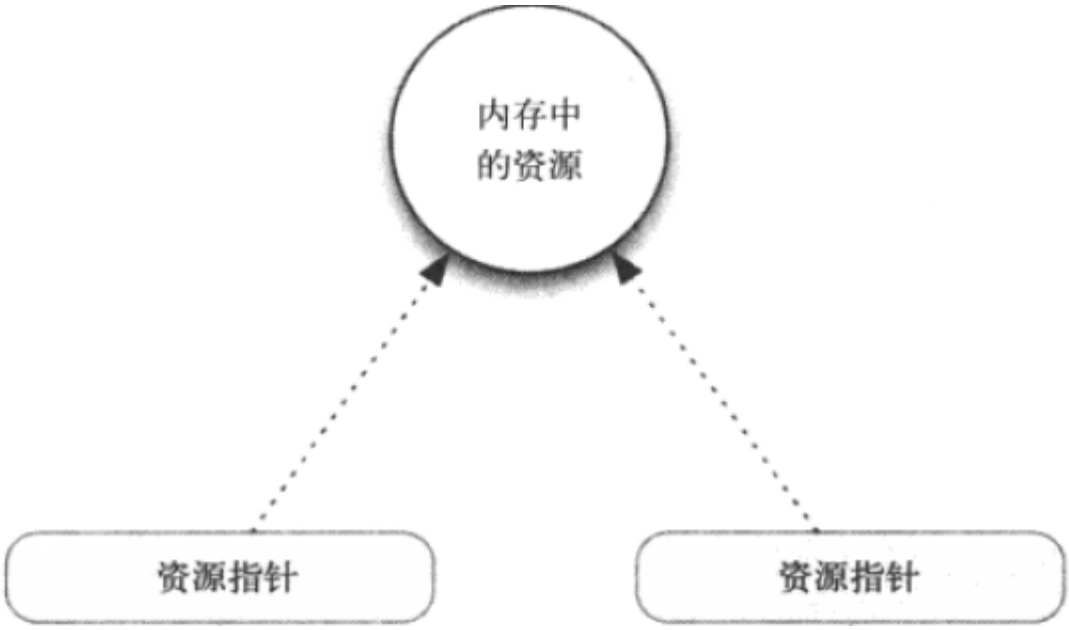
- OC 中没有二维数组，可通过嵌套数组实现二维数组。

LayoutSubViews 在什么时候被调用？

- 当 View 本身的 frame 改变时，会调用这个方法。

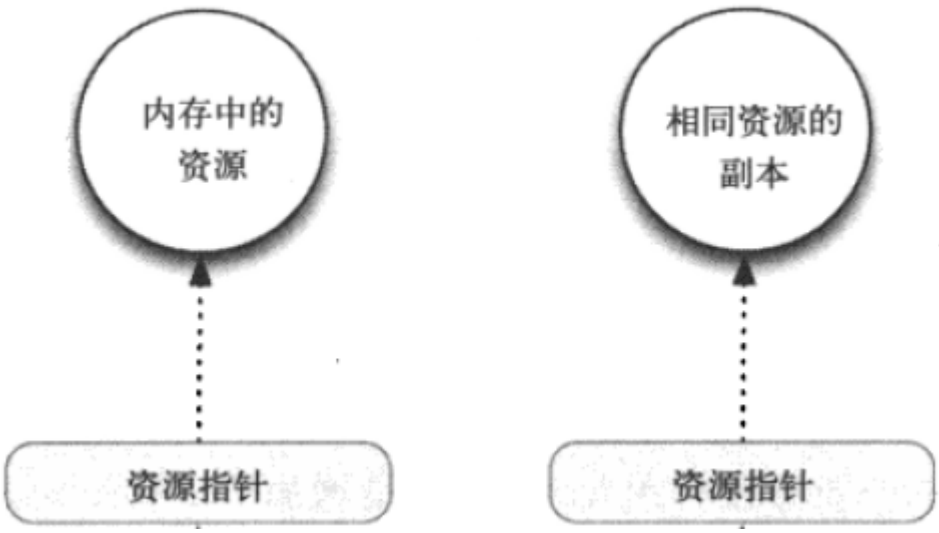
深拷贝和浅拷贝

- 如果对象有个指针型成员变量指向内存中的某个资源，那么如何复制这个对象呢？你会只是复制指针的值传给副本的新对象吗？指针只是存储内存中资源地址的占位符。在复制操作中，如果只是将指针复制给新对象，那么底层的资源实际上仍然由两个实例在共享。



示例图 1

- 浅复制：两个实例的指针仍指向内存中的同一资源，只复制指针值而不是实际资源；
- 深复制：不仅复制指针值，还复制指向指针所指向的资源。如下图：



示例图 2

单例模式理解与使用

- 单例模式是一种常用设计模式，单例模式是一个类在系统中只有一个实例对象。通过全局的一个入口点对这个实例对象进行访问；
- iOS 中单例模式的实现方式一般分为两种：非 ARC 和 ARC+GCD。

对沙盒的理解

- 每个 iOS 应用都被限制在“沙盒”中，沙盒相当于一个加了仅主人可见权限的文件夹，及时在应用程序安装过程中，系统为每个单独的应用程序生成它的主目录和一些关键的子目录。苹果对沙盒有几条限制:

- 1. 应用程序在自己的沙盒中运作，但是不能访问任何其他应用程序的沙盒；
- 2. 应用之间不能共享数据，沙盒里的文件不能被复制到其他
- 应用程序的文件夹中，也不能把其他应用文件夹复制到沙盒中；
- 3. 苹果禁止任何读写沙盒以外的文件，禁止应用程序将内容写到沙盒以外的文件夹中；
- 4. 沙盒目录里有三个文件夹：Documents——存储
- 应用程序的数据文件，存储用户数据或其他定期备份的信息；
- Library 下有两个文件夹，Caches 存储应用程序再次启动所需的信息，
- Preferences 包含应用程序的偏好设置文件，不可在这更改偏好设置；

temp 存放临时文件即应用程序再次启动不需要的文件。

- 获取沙盒根目录的方法，有几种方法：用 NSHomeDirectory 获取。
- 获取 Document 路径：

`NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,NSUserDomainMask, YES)`.

对瀑布流的理解

- 首先图片的宽度都是一样的，1.将图片等比例压缩，让图片不变形；2.计算图片最低应该摆放的位置，哪一列低就放在哪；3.进行最优排列，在 ScrollView 的基础上添加两个 tableView，然后将之前所计算的 scrollView 的高度通过 tableView 展示出来。
- 如何使用两个 TableView 产生联动：将两个 tableView 的滚动事件禁止掉，最外层 scrollView 滚动时将两个 TableView 跟着滚动，并且更改 contentOffset，这样产生效果滚动的两个 tableView。

ViewController 的 loadView,、viewDidLoad,、viewDidUnload 分别是在什么时候调用的？

- `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` 。

关键字 volatile 有什么含意?并给出三个不同的例子：

- 一个定义为 `volatile` 的变量是说这变量可能会被意想不到地改变，这样，编译器就不会去假设这个变量的值了。精确地说就是，优化器在用到这个变量时必须每次都小心地重新读取这个变量的值，而不是使用保存在寄存器里的备份。下面是 `volatile` 变量的几个例子：
  - 并行设备的硬件寄存器（如：状态寄存器）；
  - 一个中断服务子程序中会访问到的非自动变量(Non-automatic variables)；
  - 多线程应用中被几个任务共享的变量。

@synthesize、@dynamic 的理解

- `@synthesize` 是系统自动生成 getter 和 setter 属性声明;`@synthesize` 的意思是，除非开发人员已经做了，否则由编译器生成相应的代码，以满足属性声明；
- `@dynamic` 是开发者自己提供相应的属性声明,`@dynamic` 意思是由开发人员提供相应的代码：对于只读属性需要提供 `setter`，对于读写属性需要提供 `setter` 和 `getter`。查阅了一些资料确定`@dynamic` 的意思是告诉编译器,属性的获取与赋值方法由用户自己实现，不自动生成。

frame 和 bounds 有什么不同？

- 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）

view 的 touch 事件有哪些？

```
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event;

- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event;

- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event;

- (void)touchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event;
```

自定义实现 UITabbarController 的原理

- 运用字典，点击五个按钮的一个可以从字典里选择一个控制器对象，将其 View 显示到主控制器视图上。

iOS 中的响应者链的工作原理

- 每一个应用有一个响应者链，我们的视图结构是一个 N 叉树(一个视图可以有多个子视图，一个子视图同一时刻只有一个父视图),而每一个继承 `UIResponder` 的对象都可以在这个 N 叉树中扮演一个节点。
- 当叶节点成为最高响应者的时候，从这个叶节点开始往其父节点开始追朔出一条链，那么对于这一个叶节点来讲，这一条链就是当前的响应者链。响应者链将系统捕获到的 UIEvent 与 UITouch 从叶节点开始层层向下分发，期间可以选择停止分发，也可以选择继续向下分发。
- 如需了解更多细节，请读这篇文章。

View 和 View 之间传值方式

- 对象的 property 属性传值；
- 方法参数传值；
- NSUserDefault 传值；
- 块传值。

property 属性的修饰符的作用

- getter=getName、setter=setName：设置 setter 与 getter 的方法名；
- readwrite、readonly：设置可供访问级别；
- assign：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；
- retain：其 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；
- copy：其 setter 方法进行 copy 操作，与 retain 处理流程一样，先对旧值 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。
- 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。

对于 Run Loop 的理解

- RunLoop，是多线程的法宝，即一个线程一次只能执行一个任务，执行完任务后就会退出线程。主线程执行完即时任务时会继续等待接收事件而不退出。非主线程通常来说就是为了执行某一任务的，执行完毕就需要归还资源，因此默认是不运行 RunLoop 的；
- 每一个线程都有其对应的 RunLoop，只是默认只有主线程的 RunLoop 是启动的，其它子线程的 RunLoop 默认是不启动的，若要启动则需要手动启动；
- 在一个单独的线程中，如果需要在处理完某个任务后不退出，继续等待接收事件，则需要启用 RunLoop；
- NSRunLoop 提供了一个添加 NSTimer 的方法，可以指定 Mode，如果要让任何情况下都回调，则需要设置 Mode 为 Common 模式；
- 实质上，对于子线程的 runloop 默认是不存在的，因为苹果采用了懒加载的方式。如果我们没有手动调用`[NSRunLoop currentRunLoop]`的话，就不会去查询是否存在当前线程的 RunLoop，也就不会去加载，更不会创建。

SQLite 中常用的 SQL 语句

- **创建表:**creat table 表名 (字段名 字段数据类型 是否为主键, 字段名 字段数据类型, 字段名 字段数据类型...)；
- **增:** insert into 表名 (字段 1, 字段 2...) values (值 1, 值 2...)；
- **删**: delete from 表名 where 字段 = 值；

XIB 与 Storyboards 的优缺点

优点：

- XIB：在编译前就提供了可视化界面，可以直接拖控件，也可以直接给控件添加约束，更直观一些，而且类文件中就少了创建控件的代码，确实简化不少，通常每个 XIB 对应一个类。
- Storyboard：在编译前提供了可视化界面，可拖控件，可加约束，在开发时比较直观，而且一个 storyboard 可以有很多的界面，每个界面对应一个类文件，通过 storybard，可以直观地看出整个 App 的结构。

缺点：

- XIB：需求变动时，需要修改 XIB 很大，有时候甚至需要重新添加约束，导致开发周期变长。XIB 载入相比纯代码自然要慢一些。对于比较复杂逻辑控制不同状态下显示不同内容时，使用 XIB 是比较困难的。当多人团队或者多团队开发时，如果 XIB 文件被发动，极易导致冲突，而且解决冲突相对要困难很多。
- Storyboard：需求变动时，需要修改 storyboard 上对应的界面的约束，与 XIB 一样可能要重新添加约束，或者添加约束会造成大量的冲突，尤其是多团队开发。对于复杂逻辑控制不同显示内容时，比较困难。当多人团队或者多团队开发时，大家会同时修改一个 storyboard，导致大量冲突，解决起来相当困难。

将字符串“2015-04-10”格式化日期转为 NSDate 类型

```
NSString *timeStr = @"2015-04-10";

NSDateFormatter *formatter = [[NSDateFormatter alloc] init];

formatter.dateFormat = @"yyyy-MM-dd";

formatter.timeZone = [NSTimeZone defaultTimeZone];

NSDate *date = [formatter dateFromString:timeStr];

// 2015-04-09 16:00:00 +0000

NSLog(@"%@", date);
```

队列和多线程的使用原理

在 iOS 中队列分为以下几种：

- 串行队列：队列中的任务只会顺序执行；

```
dispatch_queue_t q = dispatch_queue_create("...", DISPATCH_QUEUE_SERIAL);
```

- 并行队列： 队列中的任务通常会并发执行；

```
dispatch_queue_t q =
dispatch_queue_create("......",DISPATCH_QUEUE_CONCURRENT);
```

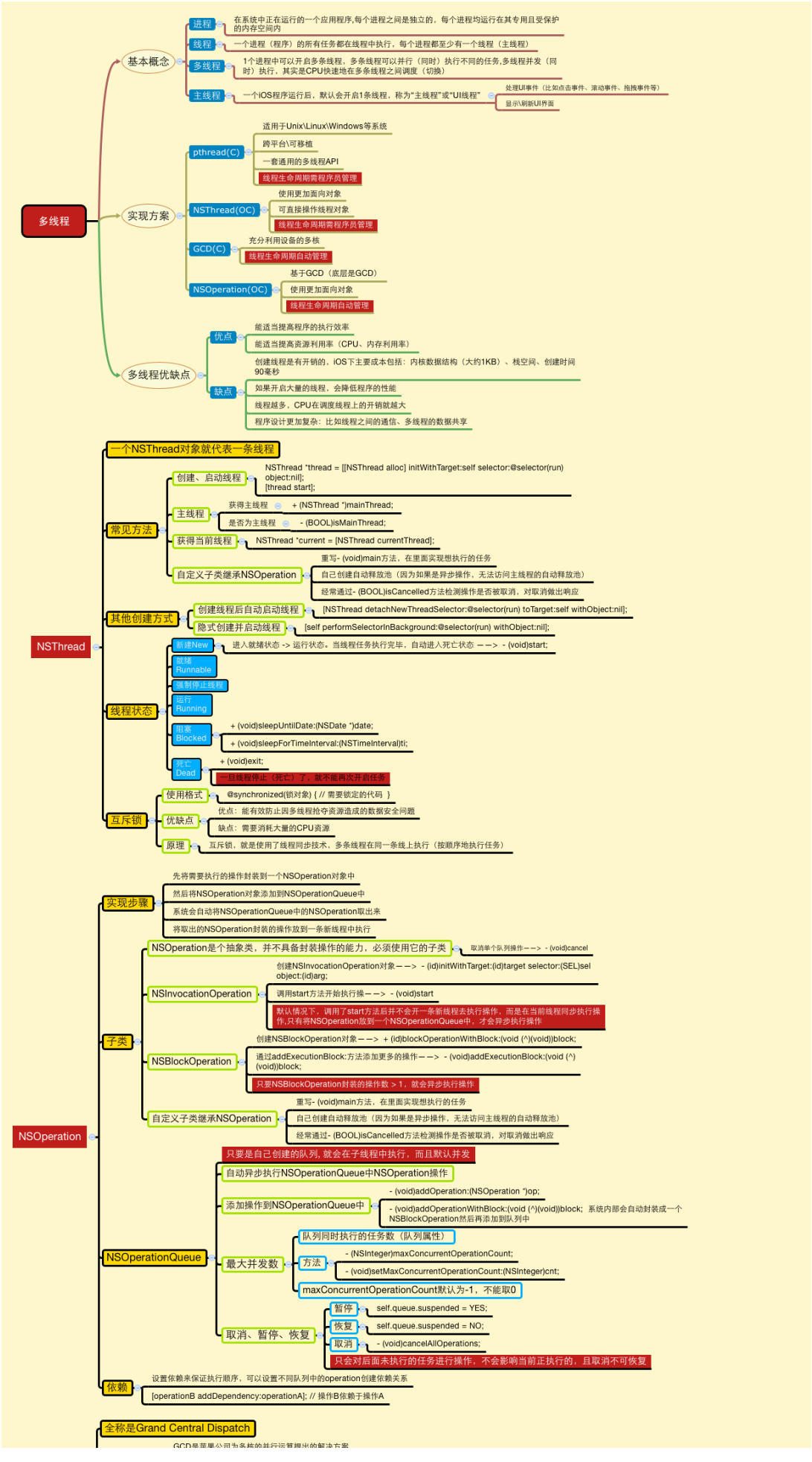
- 全局队列：是系统的，直接拿过来（GET）用就可以；与并行队列类似；

```
dispatch_queue_t q =
dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);
```

- 主队列：每一个应用程序对应唯一主队列，直接 GET 即可；在多线程开发中，使用主队列更新 UI；

```
dispatch_queue_t q = dispatch_get_main_queue();
```

- 更多细节见下图：


多线程
基本概念
进程
在系统中正在运行的一个应用程序,每个进程之间是独立的，每个进程均运行在其专用且受保护的内存空间内
线程
一个进程（程序）的所有任务都在线程中执行，每个进程都至少有一个线程（主线程）
多线程
1个进程中可以开启多条线程，多条线程可以并行（同时）执行不同的任务,多线程并发（同时）执行，其实是CPU快速地在多条线程之间调度（切换）
主线程
一个iOS程序运行后，默认会开启1条线程，称为"主线程"或"UI线程"
处理UI事件（比如点击事件、滚动事件、拖拽事件等）
显示\刷新UI界面
实现方案
pthread(C)
适用于Unix\Linux\Windows等系统
跨平台\可移植
一套通用的多线程API
线程生命周期需程序员管理
NSThread(OC)
使用更加面向对象
可直接操作线程对象
线程生命周期需程序员管理
GCD(C)
充分利用设备的多核
线程生命周期自动管理
NSOperation(OC)
基于GCD（底层是GCD）
使用更加面向对象
线程生命周期自动管理
多线程优缺点
优点
能适当提高程序的执行效率
能适当提高资源利用率（CPU、内存利用率）
缺点
创建线程是有开销的，iOS下主要成本包括：内核数据结构（大约1KB）、栈空间、创建时间90毫秒
如果开启大量的线程，会降低程序的性能
线程越多，CPU在调度线程上的开销就越大
程序设计更加复杂：比如线程之间的通信、多线程的数据共享
NSThread
一个NSThread对象就代表一条线程
常见方法
创建、启动线程
NSThread *thread = [[NSThread alloc] initWithTarget:self selector:@selector(run) object:nil];
[thread start];
主线程
获得主线程
+ (NSThread *)mainThread;
是否为主线程
- (BOOL)isMainThread;
获得当前线程
NSThread *current = [NSThread currentThread];
自定义子类继承NSOperation
重写- (void)main方法，在里面实现想执行的任务
自己创建自动释放池（因为如果是异步操作，无法访问主线程的自动释放池）
经常通过- (BOOL)isCancelled方法检测操作是否被取消，对取消做出响应
其他创建方式
创建线程后自动启动线程
[NSThread detachNewThreadSelector:@selector(run) toTarget:self withObject:nil];
隐式创建并启动线程
[self performSelectorInBackground:@selector(run) withObject:nil];
线程状态
新建New
进入就绪状态 -> 运行状态。当线程任务执行完毕，自动进入死亡状态 ——> - (void)start;
就绪Runnable
强制停止线程
运行Running
阻塞Blocked
+ (void)sleepUntilDate:(NSDate *)date;
+ (void)sleepForTimeInterval:(NSTimeInterval)ti;
死亡Dead
+ (void)exit;
一旦线程停止（死亡）了，就不能再次开启任务
互斥锁
使用格式
@synchronized(锁对象) { // 需要锁定的代码 }
优缺点
优点：能有效防止因多线程抢夺资源造成的数据安全问题
缺点：需要消耗大量的CPU资源
原理
互斥锁，就是使用了线程同步技术，多条线程在同一条线上执行（按顺序地执行任务）
NSOperation
实现步骤
先将需要执行的操作封装到一个NSOperation对象中
然后将NSOperation对象添加到NSOperationQueue中
系统会自动将NSOperationQueue中的NSOperation取出来
将取出的NSOperation封装的操作放到一条新线程中执行
子类
NSOperation是个抽象类，并不具备封装操作的能力，必须使用它的子类
取消单个队列操作——> - (void)cancel
NSInvocationOperation
创建NSInvocationOperation对象——> - (id)initWithTarget:(id)target selector:(SEL)sel object:(id)arg;
调用start方法开始执行操——> - (void)start
默认情况下，调用了start方法后并不会开一条新线程去执行操作，而是在当前线程同步执行操作,只有将NSOperation放到一个NSOperationQueue中，才会异步执行操作
NSBlockOperation
创建NSBlockOperation对象——> + (id)blockOperationWithBlock:(void (^)(void))block;
通过addExecutionBlock:方法添加更多的操作——> - (void)addExecutionBlock:(void (^)(void))block;
只要NSBlockOperation封装的操作数 > 1，就会异步执行操作
自定义子类继承NSOperation
重写- (void)main方法，在里面实现想执行的任务
自己创建自动释放池（因为如果是异步操作，无法访问主线程的自动释放池）
经常通过- (BOOL)isCancelled方法检测操作是否被取消，对取消做出响应
NSOperationQueue
只要是自己创建的队列,就会在子线程中执行，而且默认并发
自动异步执行NSOperationQueue中NSOperation操作
添加操作到NSOperationQueue中
- (void)addOperation:(NSOperation *)op;
- (void)addOperationWithBlock:(void (^)(void))block; 系统内部会自动封装成一个NSBlockOperation然后再添加到队列中
最大并发数
队列同时执行的任务数（队列属性）
方法
- (NSInteger)maxConcurrentOperationCount;
- (void)setMaxConcurrentOperationCount:(NSInteger)cnt;
maxConcurrentOperationCount默认为-1，不能取0
取消、暂停、恢复
暂停
self.queue.suspended = YES;
恢复
self.queue.suspended = NO;
取消
- (void)cancelAllOperations;
只会对后面未执行的任务进行操作，不会影响当前正执行的，且取消不可恢复
依赖
设置依赖来保证执行顺序，可以设置不同队列中的operation创建依赖关系
[operationB addDependency:operationA]; // 操作B依赖于操作A
全称是Grand Central Dispatch
GCD是苹果公司为多核的并行运算提出的解决方案

内存的使用和优化的注意事项

- **重用问题**：如 UITableViewCells、UICollectionViewCells、UITableViewHeaderFooterViews 设置正确的 reuseIdentifier，充分重用；
- **尽量把 views 设置为不透明**：当 opque 为 NO 的时候，图层的半透明取决于图片和其本身合成的图层为结果，可提高性能；
- **不要使用太复杂的 XIB/Storyboard**：载入时就会将 XIB/storyboard 需要的所有资源，包括图片全部载入内存，即使未来很久才会使用。那些相比纯代码写的延迟加载，性能及内存就差了很多；
- **选择正确的数据结构**：学会选择对业务场景最合适的数组结构是写出高效代码的基础。比如，数组: 有序的一组值。使用索引来查询很快，使用值查询很慢，插入/删除很慢。字典: 存储键值对，用键来查找比较快。集合: 无序的一组值，用值来查找很快，插入/删除很快。
  gzip/zip 压缩：当从服务端下载相关附件时，可以通过 gzip/zip 压缩后再下载，使得内存更小，下载速度也更快。
- **延迟加载**：对于不应该使用的数据，使用延迟加载方式。对于不需要马上显示的视图，使用延迟加载方式。比如，网络请求失败时显示的提示界面，可能一直都不会使用到，因此应该使用延迟加载。
- **数据缓存**：对于 cell 的行高要缓存起来，使得 reload 数据时，效率也极高。而对于那些网络数据，不需要每次都请求的，应该缓存起来，可以写入数据库，也可以通过 plist 文件存储。
- **处理内存警告**：一般在基类统一处理内存警告，将相关不用资源立即释放掉
  重用大开销对象：一些 objects 的初始化很慢，比如 `NSDateFormatter` 和 `NSCalendar`，但又不可避免地需要使用它们。通常是作为属性存储起来，防止反复创建。
- **避免反复处理数据**：许多应用需要从服务器加载功能所需的常为 JSON 或者 XML 格式的数据。在服务器端和客户端使用相同的数据结构很重要;

- **使用 Autorelease Pool**：在某些循环创建临时变量处理数据时，自动释放池以保证能及时释放内存;
- **正确选择图片加载方式**：详情阅读细读 UIImage 加载方式

UIViewController 的完整生命周期

```
-[ViewController initWithNibName:bundle:];

-[ViewController init];

-[ViewController loadView];

-[ViewController viewDidLoad];

-[ViewController viewWillAppear:];

-[ViewController viewWillLayoutSubviews:];

-[ViewController viewDidLayoutSubviews:];

-[ViewController viewDidAppear:];

-[ViewController viewWillDisappear:];

-[ViewController viewDidDisappear:];

-[ViewController viewWillUnload:];

-[ViewController viewDidUnload:];
```

UIImageView 添加圆角

- **最直接的方法就是使用如下属性设置**：

```
- imgView.layer.cornerRadius = 10;
- // 这一行代码是很消耗性能的
```

```
imgView.clipsToBounds = YES;
```

这是离屏渲染（off-screen-rendering），消耗性能的

- 给 **UIImage** 添加生成圆角图片的扩展 **API**：这是 on-screen-rendering

```
- (UIImage *)imageWithCornerRadius:(CGFloat)radius {
CGRect rect = (CGRect){0.f, 0.f, self.size};

UIGraphicsBeginImageContextWithOptions(self.size, NO,
UIScreen.mainScreen.scale);
CGContextAddPath(UIGraphicsGetCurrentContext(),
 [UIBezierPath bezierPathWithRoundedRect:rect
cornerRadius:radius].CGPath);
CGContextClip(UIGraphicsGetCurrentContext());

[self drawInRect:rect];
UIImage *image = UIGraphicsGetImageFromCurrentImageContext();

UIGraphicsEndImageContext();

return image;
```

```
}
```

文章更新记录

- 2017.03.10 更正 **UIViewController** 的生命周期

2. 写一个单例模式

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

3. iOS 应用生命周期

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，不过没有接收到事件。在没有事件处理情况下程序通常停留在这个状态。
Active 激活：程序在前台运行而且接收到了事件。这也是前台的一个正常的模式。
Backgroud 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。时间到之后会进入挂起状态(Suspended)。有的程序经过特殊的请求后可以长期处于 Backgroud 状态。
Suspended 挂起：程序在后台不能执行代码。系统会自动把程序变成这个状态而且不会发出通知。当挂起时，程序还是停留在内存中的，当系统内存低时，系统就把挂起的程序清除掉，为前台程序提供更多的内存。

iOS 的入口在 main.m 文件：

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

main 函数的两个参数，iOS 中没有用到，包括这两个参数是为了与标准 ANSI C 保持一致。 UIApplicationMain 函数，前两个和 main 函数一样，重点是后两个。

后两个参数分别表示程序的主要类(principal class)和代理类(delegate class)。如果主要类(principal class)为 nil，将从 Info.plist 中获取，如果 Info.plist 中不存在对应的 key，则默认为 UIApplication；如果代理类(delegate class)将在新建工程时创建。
根据 UIApplicationMain 函数，程序将进入 AppDelegate.m，这个文件是 xcode 新建工程时自动生成的。下面看一下 AppDelegate.m 文件，这个关乎着应用程序的生命周期。

1、**application didFinishLaunchingWithOptions**：当应用程序启动时执行，应用程序启动入口，只在应用程序启动时执行一次。若用户直接启动，lauchOptions 内无数据,若通过其他方式启动应用，**lauchOptions** 包含对应方式的内容。
2、**applicationWillResignActive**：在应用程序将要由活动状态切换到非活动状态时候，要执行的委托调用，如 按下 home 按钮，返回主屏幕，或全屏之间切换应用程序等。
3、applicationDidEnterBackground：在应用程序已进入后台程序时，要执行的委托调用。
4、applicationWillEnterForeground：在应用程序将要进入前台时(被激活)，要执行的委托调用，刚好与 applicationWillResignActive 方法相对应。
5、applicationDidBecomeActive：在应用程序已被激活后，要执行的委托调用，刚好与 applicationDidEnterBackground 方法相对应。
6、applicationWillTerminate：在应用程序要完全推出的时候，要执行的委托调用，这个需要要设置 UIApplicationExitsOnSuspend 的键值。

初次启动：
iOS_didFinishLaunchingWithOptions
iOS_applicationDidBecomeActive
按下 home 键：
iOS_applicationWillResignActive
iOS_applicationDidEnterBackground
点击程序图标进入：
iOS_applicationWillEnterForeground
iOS_applicationDidBecomeActive

当应用程序进入后台时,应该保存用户数据或状态信息，所有没写到磁盘的文件或信息，在进入后台时，最后都写到磁盘去，因为程序可能在后台被杀死。释放尽可能释放的内存。

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

方法有大概 5 秒的时间让你完成这些任务。如果超过时间还有未完成的任务，你的程序就会被终止而且从内存中清除。

如果还需要长时间的运行任务，可以在该方法中调用

```
[application beginBackgroundTaskWithExpirationHandler:^{

    NSLog(@"begin Background Task With Expiration Handler");

}];
```

程序终止

程序只要符合以下情况之一，只要进入后台或挂起状态就会终止：

①iOS4.0 以前的系统

②app 是基于 iOS4.0 之前系统开发的。

③设备不支持多任务

④在 Info.plist 文件中，程序包含了 UIApplicationExitsOnSuspend 键。

系统常常是为其他 app 启动时由于内存不足而回收内存最后需要终止应用程序，但有时也会是由于 app 很长时间才响应而终止。如果 app 当时运行在后台并且没有暂停，系统会在应用程序终止之前调用 app 的代理的方法 -(void)applicationWillTerminate:(UIApplication *)application，这样可以让你可以做一些清理工作。你可以保存一些数据或 app 的状态。这个方法也有 5 秒钟的限制。超时后方法会返回程序从内存中清除。

注意：用户可以手工关闭应用程序。

4. 一工人给老板打 7 天工要求一块金条 这金条只能切 2 次 工人每天要 1/7 金条 怎么分?

这道题解决的主要难点在于：不是给出去的就收不回来了，可以用交换的方法。

把金条分成三段（就是分两次，或者切两刀），分别是整根金条的 1/7、2/7、4/7。

第一天：给 1/7 的， 第二天：给 2/7 的，收回 1/7 的； 第三天，给 1/7 的；第四天：给 4/7 的，收回 1/7 和 2/7 的 ；第五天：给 1/7 的 ；第六天：给 2/7 的，收回 1/7 的；第七天发 1/7。

5. iOS 中 socket 使用

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 协议 对应于应用层
tcp 协议 对应于传输层
ip 协议 对应于网络层
三者本质上没有可比性。 何况 HTTP 协议是基于 TCP 连接的。

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。

我 们在传输数据时，可以只使用传输层（TCP/IP），但是那样的话，由于没有应用层，便无法识别数据内容，如果想要使传输的数据有意义，则必须使用应用层 协议，应用层协议很多，有 HTTP、FTP、TELNET 等等，也可以自己定义应用层协议。WEB 使用 HTTP 作传输层协议，以封装 HTTP 文本信息，然后使用 TCP/IP 做传输层协议将它发送到网络上。

SOCKET 原理
1、套接字（socket）概念
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。

应 用层通过传输层进行数据通信时，TCP 会遇到同时为多个应用程序进程提供并发服务的问题。多个 TCP 连接或多个应用程序进程可能需要通过同一个 TCP 协议端口传输数据。为了区别不同的应用程序进程和连接，许多计算机操作系统为应用程序与 TCP / IP 协议交互提供了套接字(Socket)接口。应 用层可以和传输层通过 Socket 接口，区分来自不同应用程序进程或网络连接的通信，实现数据传输的并发服务。

2 、建立 socket 连接
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。

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。

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。

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。

3、SOCKET 连接与 TCP 连接
创建 Socket 连接时，可以指定使用的传输层协议，Socket 可以支持不同的传输层协议（TCP 或 UDP），当使用 TCP 协议进行连接时，该 Socket 连接就是一个 TCP 连接。

4、Socket 连接与 HTTP 连接
由 于通常情况下 Socket 连接就是 TCP 连接，因此 Socket 连接一旦建立，通信双方即可开始相互发送数据内容，直到双方连接断开。但在实际网络应用 中，客户端到服务器之间的通信往往需要穿越多个中间节点，例如路由器、网关、防火墙等，大部分防火墙默认会关闭长时间处于非活跃状态的连接而导致 Socket 连接断连，因此需要通过轮询告诉网络，该连接处于活跃状态。

而 HTTP 连接使用的是“请求—响应”的方式，不仅在请求时需要先建立连接，而且需要客户端向服务器发出请求后，服务器端才能回复数据。

很 多情况下，需要服务器端主动向客户端推送数据，保持客户端与服务器数据的实时与同步。此时若双方建立的是 Socket 连接，服务器就可以直接将数据传送给 客户端；若双方建立的是 HTTP 连接，则服务器需要等到客户端发送一次请求后才能将数据传回给客户端，因此，客户端定时向服务器端发送连接请求，不仅可以 保持在线，同时也是在“询问”服务器是否有新的数据，如果有就将数据传给客户端。

下面这篇文章是 AsyncSocket 的使用教程，大家可以看看。

6. 网络请求中 post 和 get 的区别

GET 是用于获取数据的，POST 一般用于将数据发给服务器之用。

普遍答案
1.GET 使用 URL 或 Cookie 传参。而 POST 将数据放在 BODY 中。
2.GET 的 URL 会有长度上的限制，则 POST 的数据则可以非常大。
3.POST 比 GET 安全，因为数据在地址栏上不可见。

不过也有文章说其实上面的是错误的，具体参考这篇文章

7. 时间复杂度和空间复杂度

由于打不出数字符号，只能贴图了。

按数量级递增排列，常见的时间复杂度有：常数阶O(1),对数阶O(log_2n),线性阶O(n), 线性对数阶O($nlog_2n$),平方阶O(n^2)，立方阶O(n^3),...， k次方阶O(n^k),指数阶O(2^n)。随着问题规模n的不断增大，上述时间复杂度不断增大，算法的执行效率越低。
时间复杂度

求时间复杂度
【1】如果算法的执行时间不随着问题规模 n 的增加而增长，即使算法中有上千

条语句，其执行时间也不过是一个较大的常数。此类算法的时间复杂度是O(1)。

```
x=91; y=100;while(y>0) if(x>100) {x=x-10;y--;} else x++;解答:
T(n)=O(1)
```

这段程序的运行是和 n 无关的，就算它再循环一万年，我们也不管他，只是一个常数阶的函数。

【2】当有若干个循环语句时，算法的时间复杂度是由嵌套层数最多的循环语句中最内层语句的频度 f(n)决定的。

```
 x=1;
for(i=1;i<=n;i++)
    for(j=1;j<=i;j++)
       for(k=1;k<=j;k++)
           x++;
```

该程序段中频度最大的语句是(5)，内循环的执行次数虽然与问题规模 n 没有直接关系，但是却与外层循环的变量取值有关，而最外层循环的次数直接与 n 有关，因此可以从内层循环向外层分析语句(5)的执行次数： 则该程序段的时间复杂度为

$T(n)=O(n^3/6+低次项)=O(n^3)$

2

【3】算法的时间复杂度不仅仅依赖于问题的规模，还与输入实例的初始状态有关。
在数值 A[0..n-1]中查找给定值 K 的算法大致如下：

```
i=n-1;
while(i>=0&&(A[i]!=k))
  i--;
return i;
```

此算法中的语句(3)的频度不仅与问题规模 n 有关，还与输入实例中 A 的各元素取值及 K 的取值有关: ①若 A 中没有与 K 相等的元素，则语句(3)的频度 f(n)=n； ②若 A 的最后一个元素等于 K,则语句(3)的频度 f(n)是常数 0。

空间复杂度
一个程序的空间复杂度是指运行完一个程序所需内存的大小。利用程序的空间复杂度，可以对程序的运行所需要的内存多少有个预先估计。一个程序执行时除了需要存储空间和存储本身所使用的指令、常数、变量和输入数据外，还需要一些对数据进行操作的工作单元和存储一些为现实计算所需信息的辅助空

间。程序执行时所需存储空间包括以下两部分。

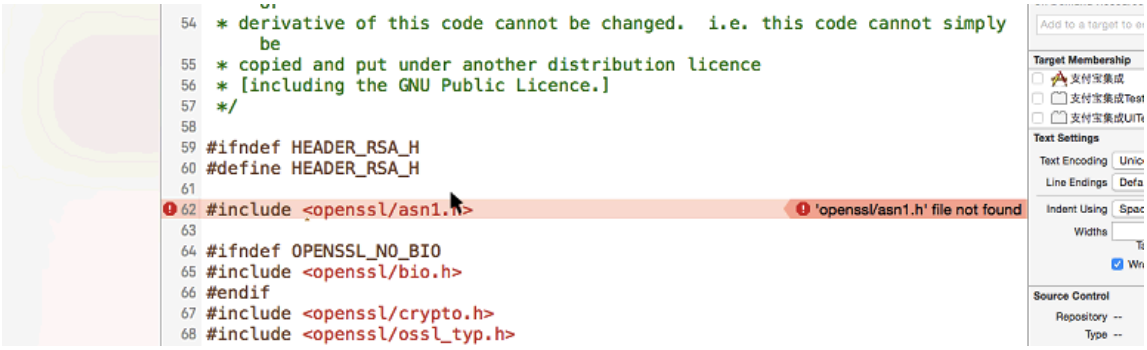
（1）固定部分。这部分空间的大小与输入/输出的数据的个数多少、数值无关。主要包括指令空间（即代码空间）、数据空间（常量、简单变量）等所占的空间。这部分属于静态空间。
（2）可变空间，这部分空间的主要包括动态分配的空间，以及递归栈所需的空间等。这部分的空间大小与算法有关。

一个算法所需的存储空间用 f(n)表示。S(n)=O(f(n))　　其中 n 为问题的规模，S(n)表示空间复杂度。

8. 支付宝 SDK 使用

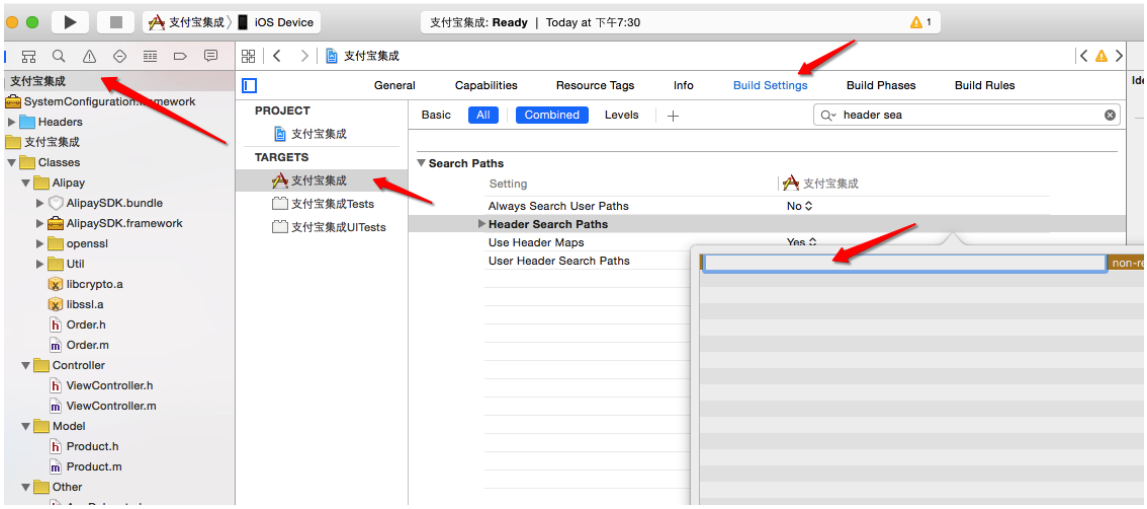
使用支付宝进行一个完整的支付功能，大致有以下步骤：向支付宝申请, 与支付宝签约，获得商户 ID（partner）和账号 ID（seller）和私钥(privateKey)。下载支付宝 SDK，生成订单信息,签名加密调用支付宝客户端，由支付宝客户端跟支付宝安全服务器打交道。支付完毕后,支付宝客户端会自动跳回到原来的应用程序，在原来的应用程序中显示支付结果给用户看。

集成之后可能遇到的问题
1）集成 SDK 编译时找不到 openssl/asn1.h 文件



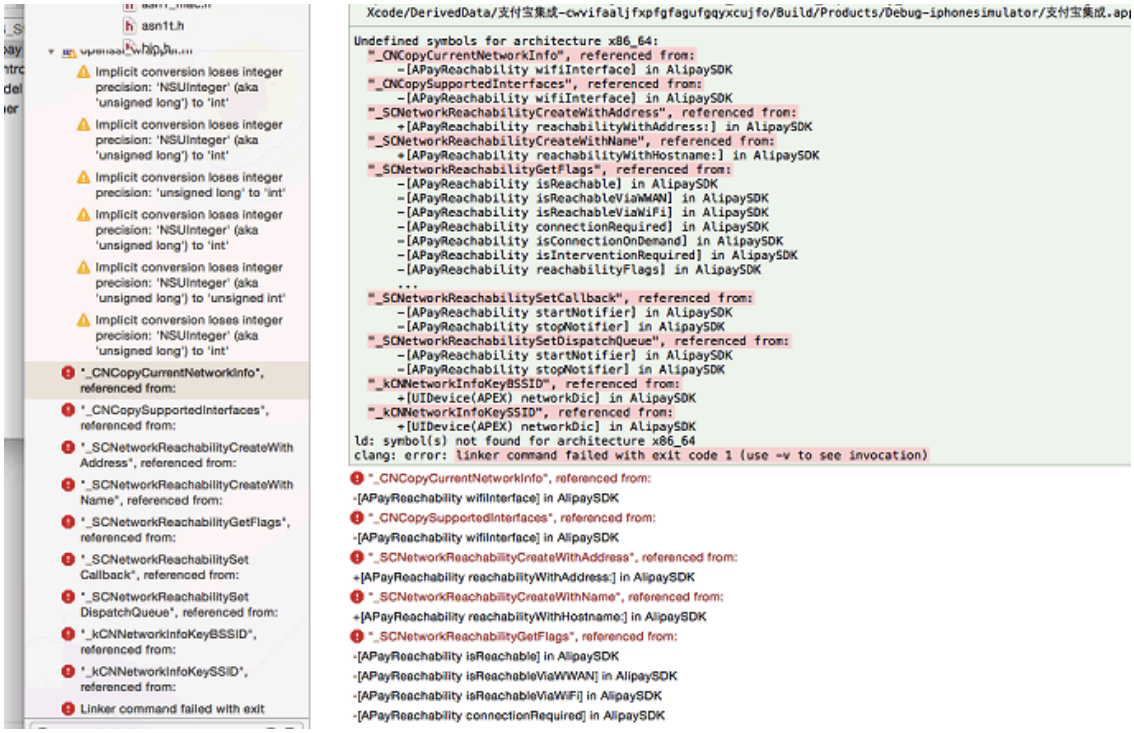
技术分享

解决方案：Build Settings --> Search Paths --> Header Search paths : $(SRCROOT)/支付宝集成/Classes/Alipay



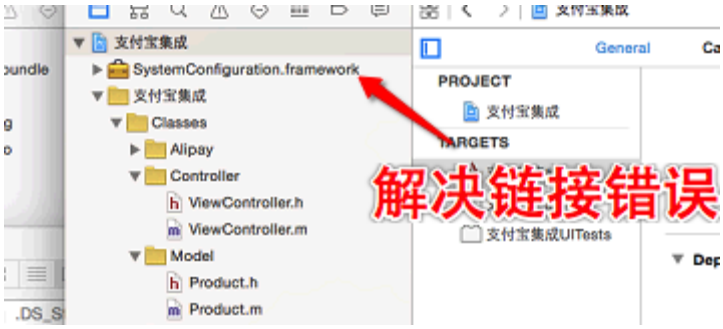
技术分享

2）链接时：找不到 SystemConfiguration.framework 这个库



技术分享

解决方案：



技术分享

打开支付宝客户端进行支付(用户没有安装支付宝客户端,直接在应用程序中添加一个 WebView,通过网页让用户进行支付)
// 注意:如果是通过网页支付完成,那么会回调该 block:callback

```
[[AlipaySDK defaultService] payOrder:orderString
fromScheme:@"jingdong" callback:^(NSDictionary *resultDic) { }];
```

在 AppDelegate.m

```
// 当通过别的应用程序,将该应用程序打开时,会调用该方法
- (BOOL)application:(UIApplication *)app openURL:(NSURL *)url
options:(NSDictionary<NSString *,id> *)options{ // 当用户通过支付宝客户
端进行支付时,会回调该 block:standbyCallback
[[AlipaySDK defaultService] processOrderWithPaymentResult:url
standbyCallback:^(NSDictionary *resultDic) { NSLog(@"result
= %@",resultDic); }]; return YES;}
```

9. 远程推送

当服务端远程向 APNS 推送至一台离线的设备时，苹果服务器 Qos 组件会自动保留一份最新的通知，等设备上线后，Qos 将把推送发送到目标设备上

远程推送的基本过程
1.客户端的 app 需要将用户的 UDID 和 app 的 bundleID 发送给 apns 服务器,进行注册,apns 将加密后的 device Token 返回给 app
2.app 获得 device Token 后,上传到公司服务器
3.当需要推送通知时,公司服务器会将推送内容和 device Token 一起发给 apns 服务器
4.apns 再将推送内容送到客户端上

创建证书的流程：
1.打开钥匙串，生成 CertificateSigningRequest.certSigningRequest 文件
2.将 CertificateSigningRequest.certSigningRequest 上传进 developer，导出.cer 文件
3.利用 CSR 导出 P12 文件

4.需要准备下设备 token 值（无空格）
5.使用 OpenSSL 合成服务器所使用的推送证书

本地 app 代码参考
1.注册远程通知

```
 - (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions//中注册远
程通知
{
[[UIApplication sharedApplication]
registerForRemoteNotificationTypes:(UIRemoteNotificationTypeAlert |
UIRemoteNotificationTypeBadge | UIRemoteNotificationTypeSound)];
}
```

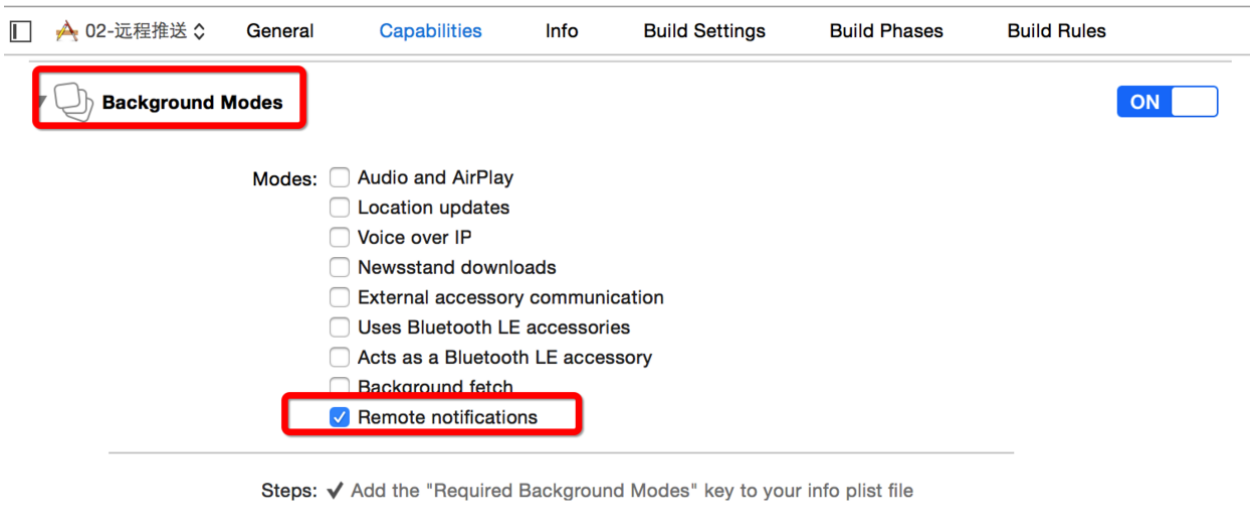
2,实现几个代理方法：

```
//获取 deviceToken 令牌
-(void)application:(UIApplication *)application
didRegisterForRemoteNotificationsWithDeviceToken:(NSData
*)deviceToken
{
//获取设备的 deviceToken 唯一编号
NSLog(@"deviceToken=%@",deviceToken);
NSString *realDeviceToken=[NSString
stringWithFormat:@"%@",deviceToken];
//去除<>
realDeviceToken = [realDeviceToken
stringByReplacingOccurrencesOfString:@"<" withString:@""];
realDeviceToken = [realDeviceToken
stringByReplacingOccurrencesOfString:@">" withString:@""];
NSLog(@"realDeviceToken=%@",realDeviceToken);
[[NSUserDefaults standardUserDefaults] setValue:realDeviceToken
forKey:@"DeviceToken"];  //要发送给服务器
}

 //获取令牌出错
-(void)application:(UIApplication *)application
didFailToRegisterForRemoteNotificationsWithError:(NSError *)error
{
//注册远程通知设备出错
NSLog(@"RegisterForRemoteNotification error=%@",error);
}
//在应用在前台时受到消息调用
```

```
-(void)application:(UIApplication *)application
didReceiveRemoteNotification:(NSDictionary *)userInfo
{
 //打印推送的消息
NSLog(@"%@",[[userInfo objectForKey:@"aps"] objectForKey:@"alert"]):
}
```

配置后台模式



一般我们是使用开发版本的 Provisioning 做推送测试,如果没有问题,再使用发布版本证书的时候一般也应该是没有问题的。为了以防万一,我们可以在越狱的手机上安装我们的使用发布版证书的 ipa 文件(最好使用 debug 版本,并打印出获取到的 deviceToken),安装成功后在;XCode->Window->Organizer-找到对应的设备查看 console 找到打印的 deviceToken。

在后台的推送程序中使用发布版制作的证书并使用该 deviceToken 做推送服务.使用开发和发布证书获取到的 deviceToken 是不一样的。

10. @protocol 和 category 中如何使用 @property

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

作者：吴白

链接：http://www.jianshu.com/p/7214f164141b

來源：简书

前言

随着移动互联网科技不断的发展和创新，如今无论是公司还是开发者或设计师个人而言，面试都是一项耗时耗钱的项目，而面对 iOS 开发者及设计师在面试时可能会遇到的问题进行了筛选与汇总。下面我们一起来一下看看吧。

一、如何绘制 UIView?

绘制一个 `UIView` 最灵活的方法就是由它自己完成绘制。实际上你不是绘制一个 `UIView`，而是子类化一个 `UIView` 并赋予绘制自己的能力。当一个 `UIView` 需要执行绘制操作时，`drawRect`：方法就会被调用，覆盖此方法让你获得绘图操作的机会。当 `drawRect`：方法被调用，当前图形的上下文也被设置为属于视图的图形上下文，你可以使用 Core Graphic 或者 UIKit 提供的方法将图形画在该上下文中。

二、什么是 MVVM？主要目的是什么？优点有哪些？

MVVM 即 Model-View-ViewModel

1. `View` 主要用于界面呈现，与用户输入设备进行交互、

2. `ViewModel` 是 MVVM 架构中最重要的部分，`ViewModel` 中包含属性，方法，事件，属性验证等逻辑，负责 `View` 与 `Model` 之间的通讯

3. `Model` 就是我们常说的数据模型，用于数据的构造，数据的驱动，主要提供基础实体的属性。

MVVM 主要目的是分离视图和模型

MVVM 优点：低耦合，可重用性，独立开发，可测试

三、get 请求与 post 请求的区别

1. `get` 是向服务器发索取数据的一种请求，而 `post` 是向服务器提交数据的一种请求

2. `get` 没有请求体，`post` 有请求体

3. `get` 请求的数据会暴露在地址栏中，而 `post` 请求不会，所以 `post` 请求的安全性比 get 请求号

4.`get`请求对 url 长度有限制，而`post`请求对 url 长度理论上是不会收限制的，但是实际上各个服务器会规定对`post`提交数据大小进行限制。

四、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？

好处：

1.使用多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 3 种多线程编程的技术：1.NSThread，2.NSOperationQueue，3.gcd；

五、XMPP 工作原理；xmpp 系统特点

原理：

1.所有从一个`client`到另一个`client`的`jabber`消息和数据都要通过 xmpp server

2.`client`链接到 server

3.`server`利用本地目录系统的证书对其认证

4.`server`查找，连接并进行相互认证

5.`client`间进行交互

特点：1）客户机/服务器通信模式；2）分布式网络；3）简单的客户端；4）XML 的数据格式

六、地图的定位是怎么实现的？

1.导入了 CoreLocation.framework
2.ios8 以后，如果需要使用定位功能，就需要请求用户授权，在首次运行时会弹框提示
3.通过本机自带的 gps 获取位置信息(即经纬度)

七、苹果内购实现流程

程序通过`bundle`存储的`plist`文件得到产品标识符的列表。

程序向 App Store 发送请求，得到产品的信息。
App Store 返回产品信息。
程序把返回的产品信息显示给用户（App 的 store 界面）
用户选择某个产品
程序向 App Store 发送支付请求
App Store 处理支付请求并返回交易完成信息。
App 获取信息并提供内容给用户。

八、支付宝，微信等相关类型的 sdk 的集成

1.在支付宝开发平台创建应用并获取 APPID
2.配置密钥

3.集成并配置 SDK
4.调用接口（如交易查询接口，交易退款接口）

九、 gcd 产生死锁的原因及解锁的方法

产生死锁的必要条件：1.互斥条件，2.请求与保持条件，3.不剥夺条件，4.循环等待条件。
解决办法：采用异步执行 block。

十、生成二维码的步骤

1.使用 `CIFilter` 滤镜类生成二维码

2.对生成的二维码进行加工，使其更清晰
3.自定义二维码背景色、填充色
4.自定义定位角标
5.在二维码中心插入小图片

十一、在使用 XMPP 的时候有没有什么困难

发送附件（图片，语音，文档...）时比较麻烦
XMPP 框架没有提供发送附件的功能，需要自己实现
实现方法，把文件上传到文件服务器，上传成功后获取文件保存路径，再把附件的路径发送给好友

十二、是否使用过环信，简单的说下环信的实现原理

环信是一个即时通讯的服务提供商
环信使用的是 XMPP 协议，它是再 XMPP 的基础上进行二次开发，对服务器 Openfire 和客户端进行功能模型的添加和客户端 SDK 的封装，环信的本质还是使用 XMPP，基于 Socket 的网络通信
环信内部实现了数据缓存，会把聊天记录添加到数据库，把附件（如音频文件，图片文件）下载到本地，使程序员更多时间是花到用户体验体验上。

总结

以上就是这篇文章的全部内容了，希望本文的内容对各位 iOS 开发者们在面试的时候能有所帮助，如果有问题大家可以留言交流。