

#####1、HomeKit?

是苹果2014年发布的智能家居平台。

#####2、什么是 OpenGL、Quartz 2D?

Quartz 2d 是Apple提供的基本图形工具库。只是适用于2D图形的绘制。

OpenGL，是一个跨平台的图形开发库。适用于2D和3D图形的绘制。

#####3、ffmpeg框架?

ffmpeg 是音视频处理工具，既有音视频编解码功能，又可以作为播放器使用。

#####4、谈谈 UITableView 的优化

- 1). 正确的复用cell。
- 2). 设计统一规格的Cell
- 3). 提前计算并缓存好高度（布局），因为heightForRowAtIndexPath:是调用最频繁的方法；
- 4). 异步绘制，遇到复杂界面，遇到性能瓶颈时，可能就是突破口；
- 4). 滑动时按需加载，这个在大量图片展示，网络加载的时候很管用！
- 5). 减少子视图的层级关系
- 6). 尽量使所有的视图不透明化以及做切圆操作。
- 7). 不要动态的add 或者 remove 子控件。最好在初始化时就添加完，然后通过hidden来控制是否显示。
- 8). 使用调试工具分析问题。

#####5、如何实行cell的动态的行高

如果希望每条数据显示自身的行高，必须设置两个属性，1.预估行高，2.自定义行高。

设置预估行高 tableView.estimatedRowHeight = 200。

设置定义行高 `tableView.estimatedRowHeight = UITableViewAutomaticDimension`。

如果要想自定义行高有效，必须让容器视图有一个自下而上的约束。

#####6、说说你对 block 的理解

栈上的自动复制到堆上，block 的属性修饰符是 `copy`，循环引用的原理和解决方案。

#####7、说说你对 runtime 的理解

主要是方法调用时如何查找缓存，如何找到方法，找不到方法时怎么转发，对象的内存布局。

#####8、什么是野指针、空指针？

野指针：不知道指向了哪里的指针叫野指针。即指针指向不确定，指针存的地址是一个垃圾值，未初始化。

空指针：不指向任何位置的指针叫空指针。即指针没有指向，指针存的地址是一个空地址，`NULL`。

#####9、什么是 OOA / OOD / OOP ？

OOA (Object Oriented Analysis) --面向对象分析

OOD (Object Oriented Design) --面向对象设计

OOP (Object Oriented Programming) --面向对象编程

#####10. 多线程是什么

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。

#####11. iOS 中的多线程

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).

#####12. 在项目什么时候选择使用GCD，什么时候选择NSOperation?

项目中使用NSOperation的优点是NSOperation是对线程的高度抽象，在项目中使用它，会使项目的程序结构更好，子类化NSOperation的设计思路，是具有面向对象的优点(复用、封装)，使得实现是多线程支持，而接口简单，建议在复杂项目中使用。

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

#####13 KVO, NSNotification, delegate及block区别

* KVO就是cocoa框架实现的观察者模式，一般同KVC搭配使用，通过KVO可以监测一个值的变化，比如View的高度变化。是一对多的关系，一个值的变化会通知所有的观察者。

* NSNotification是通知，也是一对多的使用场景。在某些情况下，KVO和NSNotification是一样的，都是状态变化之后告知对方。NSNotification的特点，就是需要被观察者先主动发出通知，然后观察者注册监听后再来进行响应，比KVO多了发送通知的一步，但是其优点是监听不局限于属性的变化，还可以对多种多样的状态变化进行监听，监听范围广，使用也更灵活。

* delegate 是代理，就是我不想做的事情交给别人做。比如狗需要吃饭，就通过delegate通知主人，主人就会给他做饭、盛饭、倒水，这些操作，这些狗都不需要关心，只需要调用delegate（代理人）就可以了，由其他类完成所需要的操作。所以delegate是一对一关系。

* block是delegate的另一种形式，是函数式编程的一种形式。使用场景跟delegate一样，相比delegate更灵活，而且代理的实现更直观。

* KVO一般的使用场景是数据，需求是数据变化，比如股票价格变化，我们一般使用KVO（观察者模式）。

* delegate一般的使用场景是行为，需求是需要别人帮我做一件事情，比如买卖股票，我们一般使用delegate。

* Notification一般是进行全局通知，比如利好消息一出，通知大家去买入。

* delegate是强关联，就是委托和代理双方互相知道，你委托别人买股票你就需要知道经纪人，经纪人也不要知道自己的顾客。

* Notification是弱关联，利好消息发出，你不需要知道是谁发的也可以做出相应的反应，同理发消息的人也不需要知道接收的人也可以正常发出消息。

#####14 将一个函数在主线程执行的4种方法

* GCD方法，通过向主线程队列发送一个block块，使block里的方法可以在主线程中执行。

```
...
```

```
dispatch_async(dispatch_get_main_queue(), ^{
```

```
//需要执行的方法
```

```
});
```

```
...
```

```
* NSOperation 方法
```

```
...
```

```
NSOperationQueue *mainQueue = [NSOperationQueue mainQueue]; //主队列
```

```
NSBlockOperation *operation = [NSBlockOperation blockOperationWithBlock:^(
```

```
//需要执行的方法
```

```
});
```

```
[mainQueue addOperation:operation];
```

```
...
```

```
* NSThread 方法
```

```
...
```

```
[self performSelector:@selector(method) onThread:[NSThread mainThread] withObject:nil  
waitUntilDone:YES modes:nil];
```

```
[self performSelectorOnMainThread:@selector(method) withObject:nil waitUntilDone:YES];
```

```
[[NSThread mainThread] performSelector:@selector(method) withObject:nil];
```

```
RunLoop方法
```

```
[[NSRunLoop mainRunLoop] performSelector:@selector(method) withObject:nil];
```

```
...
```

```
* RunLoop方法
```

```
...
```

```
[[NSRunLoop mainRunLoop] performSelector:@selector(method) withObject:nil];
```

```
...
```

#####15、 如何让计时器调用一个类方法

* 计时器只能调用实例方法，但是可以在这个实例方法里面调用静态方法。

* 使用计时器需要注意，计时器一定要加入RunLoop中，并且选好model才能运行。
scheduledTimerWithTimeInterval方法创建一个计时器并加入到RunLoop中所以可以直接使用。

* 如果计时器的repeats选择YES说明这个计时器会重复执行，一定要在合适的时机调用计时器的invalid。不能在dealloc中调用，因为一旦设置为repeats 为yes，计时器会强持有self，导致dealloc永远不会被调用，这个类就永远无法被释放。比如可以在viewDidDisappear中调用，这样当类需要被回收的时候就可以正常进入dealloc中了。

...

```
[NSTimer scheduledTimerWithTimeInterval:1 target:self selector:@selector(timerMethod)
userInfo:nil repeats:YES];
```

```
-(void)timerMethod
```

```
{
```

```
//调用类方法
```

```
[[self class] staticMethod];
```

```
}
```

```
-(void)invalid
```

```
{
```

```
[timer invalid];
```

```
timer = nil;
```

```
}
```

...

#####16、 如何重写类方法

1、在子类中实现一个同基类名字一样的静态方法

2、在调用的时候不要使用类名调用，而是使用[self class]的方式调用。原理，用类名调用是早绑定，在编译期绑定，用[self class]是晚绑定，在运行时决定调用哪个方法。

#####17、 NSTimer创建后，会在哪个线程运行。

用scheduledTimerWithTimeInterval创建的，在哪个线程创建就会被加入哪个线程的RunLoop中就运行在哪个线程

自己创建的Timer，加入到哪个线程的RunLoop中就运行在哪个线程。

#####18、 id和NSObject * 的区别

id是一个 objc_object 结构体指针，定义是

...

```
typedef struct objc_object *id
```

...

* id可以理解为指向对象的指针。所有oc的对象 id都可以指向，编译器不会做类型检查，id调用任何存在的方法都不会在编译阶段报错，当然如果这个id指向的对象没有这个方法，该崩溃还是会崩溃的。

* NSObject *指向的必须是NSObject的子类，调用的也只能是NSObject里面的方法否则就要做强制类型转换。

* 不是所有的OC对象都是NSObject的子类，还有一些继承自NSProxy。NSObject *可指向的类型是id的子集。

#####19、 浅谈iOS开发中方法延迟执行的几种方式？

* Method1\ performSelector方法

* Method2\ NSTimer定时器

* Method3\ NSThread线程的sleep

* Method4\ GCD

公用延迟执行方法:

...

```
- (void)delayMethod{  
    NSLog(@"delayMethodEnd");  
}
```

...

#####Method1:performSelector

...

```
[self performSelector:@selector(delayMethod) withObject:nil]
```

...

可传任意类型参数/ afterDelay:2.0];

注：此方法是一种非阻塞的执行方式，未找到取消执行的方法。

> 程序运行结束

```
> `2015-08-31 10:56:59.361 CJDelayMethod[1080:39604] delayMethodStart2015-08-31  
10:56:59.363 CJDelayMethod[1080:39604] nextMethod2015-08-31 10:57:01.364  
CJDelayMethod[1080:39604] delayMethodEnd`
```

Method2:NSTimer定时器

...

```
NSTimer *timer = [NSTimer scheduledTimerWithTimeInterval:2.0 target:self  
selector:@selector(delayMethod) userInfo:nil repeats:NO];
```

...

注：此方法是一种非阻塞的执行方式，

取消执行方法：`- (void)invalidate;`即可

> 程序运行结束

```
> `2015-08-31 10:58:10.182 CJDelayMethod[1129:41106] delayMethodStart2015-08-31
10:58:10.183 CJDelayMethod[1129:41106] nextMethod2015-08-31 10:58:12.185
CJDelayMethod[1129:41106] delayMethodEnd`
```

Method3:NSThread线程的sleep

...

```
[NSThread sleepForTimeInterval:2.0];
```

...

注：此方法是一种阻塞执行方式，建议放在子线程中执行，否则会卡住界面。但有时还是需要阻塞执行，如进入欢迎界面需要沉睡3秒才进入主界面时。

没有找到取消执行方式。

> 程序运行结束

```
> `2015-08-31 10:58:41.501 CJDelayMethod[1153:41698] delayMethodStart2015-08-31
10:58:43.507 CJDelayMethod[1153:41698] nextMethod`
```

Method4:GCD

...

```
__block ViewController/*主控制器*/ *weakSelf = self;
```

```
dispatch_time_t delayTime = dispatch_time(DISPATCH_TIME_NOW, (int64_t)(2.0/*延迟执行
时间*/ * NSEC_PER_SEC));
```

```
dispatch_after(delayTime, dispatch_get_main_queue(), ^{
```

```
[weakSelf delayMethod];
```

```
});`
```

...

注：此方法可以在参数中选择执行的线程，是一种非阻塞执行方式。没有找到取消执行方式。

> 程序运行结束

```
> `2015-08-31 10:59:21.652 CJDelayMethod[1181:42438] delayMethodStart2015-08-31
10:59:21.653 CJDelayMethod[1181:42438] nextMethod2015-08-31 10:59:23.653
CJDelayMethod[1181:42438] delayMethodEnd`
```

#####20、NSPersistentStoreCoordinator , NSManagedObjectContext 和NSManagedObjectContext 中的那些需要在线程中创建或者传递?

答: NSPersistentStoreCoordinator是持久化存储协调者, 主要用于协调托管对象上下文和持久化存储区之间的关系。NSManagedObjectContext使用协调者的托管对象模型将数据保存到数据库, 或查询数据。

#####20、您是否做过一部的网络处理和通讯方面的工作? 如果有, 能具体介绍一下实现策略么

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

#####21、你使用过Objective-C的运行时编程 (Runtime Programming) 么? 如果使用过, 你用它做了什么? 你还能记得你所使用的相关的头文件或者某些方法的名称吗?

答: Objective-C的重要特性是Runtime (运行时), 在#import <objc/runtime.h> 下能看到相关的方法, 用过objc_getClass()和class_copyMethodList()获取过私有API;使用

```
``objective-c
```

```
Method method1 = class_getInstanceMethod(cls, sel1);
```

```
Method method2 = class_getInstanceMethod(cls, sel2);
```

```
method_exchangeImplementations(method1, method2);
```

```
...
```

代码交换两个方法, 在写unit test时使用到。

#####22、Core开头的系列的内容。是否使用过CoreAnimation和CoreGraphics。UI框架和CA, CG框架的联系是什么? 分别用CA和CG做过些什么动画或者图像上的内容。(有需要的话还可以涉及Quartz的一些内容)

答：UI框架的底层有CoreAnimation，CoreAnimation的底层有CoreGraphics。

UIKit |

----- |

Core Animation |

Core Graphics |

Graphics Hardware|

#####23、是否使用过CoreText或者CoreImage等？如果使用过，请谈谈你使用CoreText或者CoreImage的体验。

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

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

答：NSNotification是通知模式在iOS的实现，

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

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

#####25、你用过NSOperationQueue么？如果用过或者了解的话，你为什么要使用NSOperationQueue，实现了什么？请描述它和G.C.D的区别和类似的地方（提示：可以从两者的实现机制和适用范围来描述）。

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

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

#####26、既然提到G.C.D，那么问一下在使用G.C.D以及block时要注意些什么？它们两是一回事儿么？block在ARC中和传统的MRC中的行为和用法有没有什么区别，需要注意些什么？

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

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

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

#####28、你实现过一个框架或者库以供别人使用么？如果有，请谈一谈构建框架或者库时候的经验；如果没有，请设想和设计框架的public的API，并指出大概需要如何做、需要注意一些什么方面，来使别人容易地使用你的框架。

答：抽象和封装，方便使用。首先是对问题有充分的了解，比如构建一个文件解压压缩框架，从使用者的角度出发，只需关注发送给框架一个解压请求，框架完成复杂文件的解压操作，并且在适当的时候通知给是哦难过者，如解压完成、解压出错等。在框架内部去构建对象的关系，通过抽象让其更为健壮、便于更改。其次是API的说明文档。