最新面试题汇总

序言

目前形势，参加到iOS队伍的人是越来越多，培训机构都是一火车地向用人单位输送iOS开发人员，找过工作人可能会更深刻地体会到2016年的就业形势不容乐观，有点打破了生态圈的动态平衡。不过我们的iOS岗位仍然很多，就看我们应聘者的能力有多高有多强，比如技术能力，动手能力，解决问题能力，自学能力，沟通表达能力等等要求越来越高。接下来呢，轩哥和轩哥的朋友一起帮大家整理一下面试题，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

```
1 @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，请查看唐巧的这篇文章。

http://blog.devtang.com/2015/11/02/mvc-and-mvvm/

或者http://www.cocoachina.com/ios/20160107/14916.html

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,或者runtime的形式达到增加成员变量的目的;

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(不过如果通过类别更改系统方法现在的编译器会报警，有些根本改不了的，所有还是不要改了);

分散类的实现：如：

```
1
2
```

```
+ (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）和键值观察（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(或者setter方法或者增加触发kvo的方法,详细请看苹果的官方文档有详细说明)来修 改它的实例变量,这样才能被观察者观察到。
苹果kvo 官方文档:https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/KeyValueObserving/Articles/KVOCompliance.html#//apple_ref/doc/uid/20002178-BAJEAIEE

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。

首先查看苹果的官方文档:https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/KeyValueCoding/Articles/Compliant.html#//apple_ref/doc/uid/20002172

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

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。

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。

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。

NSManagedObjectModel：被管理的数据模型、数据结构。

NSFetchRequest：数据请求；

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,具体区别戳这里
NSCopying协议方法为：

```
- (id)copyWithZone:(NSZone *)zone {
 MyObject *copy = [[[self class] allocWithZone: zone] in
it];
 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 arg
3;
@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:defin
eString]
```

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:NSRunLoopCommonM
odes];
```

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

iPhone设备

iPod设备

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：

1.并行设备的硬件寄存器（如：状态寄存器）；
2.一个中断服务子程序中会访问到的非自动变量(Non-automatic variables)；
3.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：

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 viewWillDisappear:];
-[ViewController viewWillAppear:];
-[ViewController viewDidAppear:];
-[ViewController viewDidDisappear:];
```

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;
}
```