基础

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以及NSNotification是使用场景

- `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。
- 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。

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];
```

深拷贝与浅拷贝

- 深拷贝同浅拷贝的区别：浅拷贝是指针拷贝，对一个对象进行浅拷贝，相当于对指向对象的指针进行复制，产生一个新的指向这个对象的指针，那么就是有两个指针指向同一个对象，这个对象销毁后两个指针都应该置空。深拷贝是对一个对象进行拷贝，相当于对对象进行复制，产生一个新的对象，那么就有两个指针分别指向两个对象。当一个对象改变或者被销毁后拷贝出来的新的对象不受影响。
- 实现深拷贝需要实现NSCoying协议，实现- (id)copyWithZone:(NSZone *)zone 方法。当对一个property属性含有copy修饰符的时候，在进行赋值操作的时候实际上就是调用这个方法。

父类实现深拷贝时，子类如何实现深度拷贝。父类没有实现深拷贝时，子类如何实现深度拷贝

- 父类实现深拷贝之后，子类只要重写copyWithZone方法，在方法内部调用父类的copyWithZone方法，之后实现自己的属性的处理
- 父类没有实现深拷贝，子类除了需要对自己的属性进行处理，还要对父类的属性进行处理。

简介计时器

- 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。

计时器调用静态方法

```
[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;
}
```

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。

如何重写类方法

- 在子类中实现一个同基类名字一样的静态方法
- 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。

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NSObjec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。

iOS 核心框架

- CoreAnimation　核心动画
- CoreGraphics　核心图层
- CoreLocation　核心定位
- AVFoundation　多媒体

- Foundation　核心基本功能

iOS核心机制

- UITableView 重用
- ObjC内存管理；自动释放池，ARC如何实现
- RunLoop
- Runtime
- Block的定义、特性、内存区域、如何实现
- NSOperation
- GCD

面向对象编程

- 封装、继承、多态
- 设计模式6个原则
    1. 设计一个类的功能，如何划分粒度（单一职责）
    2. 接口隔离。
    3. 如果有一个鸟类，有飞的动作，一个鸵鸟继承它是合适的吗（里氏替换）
    4. 类之间的依赖如何依赖偶合度最小（依赖倒转）
    5. 高层依赖低层，低层不能依赖高层。依赖接口，不能依赖具体的类。（迪米特）
    6. 设计类，能做到只增加代码，而不修改代码，有哪些经验（开放封闭）

数据结构

- 8大排序算法
- 二叉树实现
- 二分查找实现

计算机基础技术

- 计算机网络： TCP/IP、HTTPCDN、SPDY
- 计算机安全： RSA、AES、DES
- 操作系统：线程、进程、堆栈、死锁、调度算法

iOS新特性、新技术

- iOS7 UIDynamic、SpritKit、新布局、扁平化
- iOS8 应用程序扩展、HealthKit、SceneKit、CoreLocation、TouchID、PhotoKit
- iOS9 3D-Touch
- Apple Watch

简述OC中内存管理机制

- 内存管理机制:使用引用计数管理,分为`ARC`和`MRC`
- `MRC`需要程序员自己管理内存,`ARC`则不需要.但是并不是所有对象在`ARC`环境下均不需要管理内存,子线程和循环引用并不是这样
- 与`retain`配对使用的是`release`,`retain`代表引用计数+1,`release`代表引用计数-1,当引用计数减为0时,对象则被系统自动销毁.与`alloc`配对使用的是`dealloc`,`alloc`代表为对象开

辟内存空间,`dealloc`则代表销毁对象的内存空间

readwrite,readonly,assign,retain,copy,nonatomic,atomic,strong,weak的作用

- 读写属性:readonly和readwrite；语义属性:assign/retain/copy； 原子性:nonatomic/atomic
- readwrite代表可读,可写,即有setter和getter方法,是默认属性.readonly代表只可读,即只有get方法,因为不会生成setter方法,所以它不可以和copy/retain/assign组合使用
- weak和assign均是弱引用,assign修饰基本数据类型,weak修饰对象类型.strong和weak用于ARC下(ARC下的代理使用weak,block块使用copy).strong相当于retain.weak相当于assign;assign/retain/copy这些属性用于指定set访问器的语义,也就是说,这些属性决定了以何种方式对数据成员赋值.
- assign,直接赋值,引用计数不改变,适用于基本数据类型
- retain,浅拷贝,使用的是原来的内存空间,只能适用于Objective-C对象类型,而不能适用于Core Foundation对象(retain会增加对象的引用计数,而基本数据和Core Foundation对象都没有引用计数).
- copy:对象的拷贝,新申请一块内存空间,并把原始内容复制到那片空间.新对象的引用计数为1,此属性只对那些遵循了NSCopy协议的对象类型有效.
- nonatomic,非原子性访问,不加同步,是异步操作.默认为atomic,原子操作,atomic是Objc使用的一种线程保护技术,基本上来讲,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,而这种机制是消耗系统内存资源的,所以在移动端,都选择nonatomic.

请简述内存的5个区

- 栈区：由编译器自动分配释放，不需要管理内存
- 堆区：一般由程序员分配释放
- 全局区：存放全局变量和静态变量
- 文字常量区：存放常量字符创
- 程序代码区：存放二进制代码

类变量的@protected,@private,@public,@package的含义

- @protected 受保护的，本类以及子类可见
- @private 私有的，类内可见
- @public 公有的，类内，子类外部均可访问
- @package 可见度@protected和@public之间，这个类型最常用于矿建的实例变量

浅谈对线程和进程的理解

- 进程：正在运行的程序，负责程序的内存分配
- 线程：线程是进程中一个独立执行的控制单元，一个进行至少包含一个线程，即主线程
- 创建线程的目的：开辟一个新的执行路径，运行指定的代码，与主线程的代码实现同时执行

iOS中多线程的开发

- 多线程的使用场景：防止卡顿，可以同时完成多个任务，并且不影响主线程，把耗时操作放在子线程中执行但是会消耗内存
- iOS实现多线程的方式：（1）NSThread，内存需要自己管理（2）GCD （3）

NSOperationQueue，不必关注线程

线程同步和异步的区别?ios中如何实现线程的同步?

- 同步：任务顺序执行，下一个任务以来上一个任务的完成
- 异步：任务执行顺序不一定，一起执行
- 实现：设置依赖NSOpreationQueue、GCD中的串行队列.

iOS类是否可以多继承,如果没有,怎么实现

- 不可以多继承.
- 可以通过类目,延展,协议实现多继承
- 类目:类目也叫分类,英文category,在没有原类.m文件的基础上,给该类添加方法.类目里不能添加实例变量,不能添加和原始类方法名相同的方法,否则会发生覆盖.一个类可以添加多个类目,类目中的方法可以成为原始类的一部分,和原始类方法级别相同,可以被子类继承.
- 延展:Extension,是一种特殊形式的类目,主要是在一个类的.m里面声明与实现.作用:就是给某类添加私有方法或者私有变量.
- 虽然延展是给一个类定义私有方法,但是OC没有绝对的私有方法,其实还是可以调用的,延展里面声明的变量只能在该类内部使用,外界访问不了.如果是新建文件建的的某类延展.h文件,则不能添加实例变量,如果括号里没有类目名,则认为延展里面的方法为全都必须实现,如果有,则可选实现.
- 类目写的方法必须实现,延展写的方法非必须.

堆和栈的区别

- 栈：内存系统管理，系统凯里，系统释放，先进后出
- 堆：内存自己管理，自己开辟，自己释放，先进先出

iOS本地数据存储都有几种方式

- Write写入方式:永久保存在磁盘中.具体:a.获得文件保存的路径.b.生成该路径下的文件,c,往文件中写入数据.d.从文件中读出数据.
- NSUserDefaults:用来保存应用程序设置和属性,用户保存的数据.用户再次打开程序或者开机后这些数据仍然存在.NSUserDefaults可以存储的数据类型包括:NSData,NSString,NSNumber,NSDate,NSArray.NSDictionary,其他类型的数据需要先行转换.
- NSkeyedArchiver:采用归档的形式来保存数据,该数据对象需要遵守NSCoding协议,并且该对象对应的类必须提供encodeWithCoder:和initWithCoder:方法.前一个方法告诉系统怎么对对象进行编码,而后一个方法则是告诉系统怎么对对象进行解码.
- SQLite:采用SQLite数据库来存储数据,SQLite作为一种轻量级数据库.具体:a.添加SQLite相关的库以及头文件,b.使用数据库存数数据:打开数据库,编写数据库语句,执行,关闭数据库.另:写入数据库,字符串可以采用char方式,而从数据库中取出char类型,当char类型有表示中文字符时,会出现乱码,这是因为数据库默认使用ascII编码方式,所以想要正确从数据库中取出中文,需要使用NSString来接受从数据库取出的字符串.
- CoreData:原理是对SQLite的封装,开发者不需要接触sql语句,就可以对数据库进行操作.

iOS动态类型和动态绑定

- 多态：父类指针指向子类对象
- 动态类型：只有在运行期，才能确定其真正类型

- 动态加载：根据不同的条件，加载不同的资源

深拷贝和浅拷贝的理解

- 深拷贝：拷贝内容
- 浅拷贝：拷贝指针

怎么实现一个单例类

- 单例是一种设计模式，对象只有一个。
- 缺点：对象不会被释放，如果创建很多的话会占用很多内存
- 优点：可以当做工具类使用

```
static Single *single = nil ;
    (Single *)shareInstance{
    @synchronized(self){
  if(!single){
     single = [[Single alloc]init] ;
   }
   }
```

return single ; }

什么是安全释放

先释放再置空

RunLoop是什么

- 事件循环,是线程里面的一个组件.主线程的RunLoop是自动开启的.分为:计时源(timer source),事件源(输入源):input source.防止CPU中断(保证程序执行的线程不会被系统终止).
- `Runloop`提供了一种异步执行代码的机制,并不能并行执行任务,是事件接收和分发机制的一个实现.每一个线程都有其对应的RunLoop,但是默认非主线程的RunLoop是没有运行的,需要为RunLoop添加至少一个事件源,然后run它.
- 一般情况下我们是没有必要去启动线程的RunLoop的,除非你在一个单独的线程中需要长时间的检测某个事件.
- `RunLoop`,正如其名所示,是线程进入和被线程用来响应事件以及调用事件处理函数的地方.
- `input source`传递异步事件,通常是来自其他线程和不同程序的消息.
- `timer source`传递同步事件.
- 当有事件发生时,RunLoop会根据具体的事件类型通知应用程序作出响应.
- 当没有事件发生时,RunLoop会进入休眠状态,从而到达省电的目的.
- 当事件再次发生时,RunLoop会被重新唤醒,处理事件.
- 一般在开发中很少会主动创建RunLoop,而通常会把事件添加到RunLoop中.

什么是序列化和反序列化,可以用来做什么?如何在OC中实现复杂对象的存储.

- `序列化和反序列化`:归档和反归档,进行本地化,进行数据存储.
- `CoreData`:数据托管.有四种存储方式:xml,sqlite,二进制,内存.
- 遵循NSCoding协议之后,进行归档即可实现复杂对象的存储.

写一个标准宏MIN，这个宏输入两个参数并返回较小的一个

> define MIN(A,B) (A)>(B)?((B):(A)

iPhone OS 有没有垃圾回收机制,简易阐述一下OC内存管理

没有，oc的内存管理是依赖引用计数，ARC和MRC两个管理方式

简述应用程序按HOME键后进入后台的生命周期，一起从前台进入后台时的生命周期

前者:

> - (void)applicationWillResignActive:(UIApplication *)application
> - (void)applicationDidEnterBackground:(UIApplication *)application

后者:

> - (void)applicationWillEnterForeground:(UIApplication *)application*
> - (void)applicationDidBecomeActive:(UIApplication *)application*

解释程序启动后回调方法的含义

告诉代理进程启动但还没进入状态保存

> - (BOOL)application:(UIApplication *)application willFinishLaunchingWithOptions:(NSDictionary *)launchOptions

告诉代理启动基本完成程序准备开始运行

> - (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
> 当应用程序将要入非活动状态执行，在此期间，应用程序不接收消息或事件，比如来电话了
> - (void)applicationWillResignActive:(UIApplication *)application
> 当应用程序入活动状态执行，这个刚好跟上面那个方法相反
> - (void)applicationDidBecomeActive:(UIApplication *)application*
> 当程序被推送到后台的时候调用。所以要设置后台继续运行，则在这个函数里面设置即可
> - (void)applicationDidEnterBackground:(UIApplication *)application
> 当程序从后台将要重新回到前台时候调用，这个刚好跟上面的那个方法相反。
> - (void)applicationWillEnterForeground:(UIApplication *)application
> 当程序将要退出是被调用，通常是用来保存数据和一些退出前的清理工作。这个需要要设置UIApplicationExitsOnSuspend的键值。
> - (void)applicationWillTerminate:(UIApplication *)application
> 当程序载入后执行
> - (void)applicationDidFinishLaunching:(UIApplication)*application*

ViewController的alloc,loadView,viewDidLoad,viewWillAppear,viewDidUnload,dealloc,init分别是在什么时候调用?在自定义ViewController的时候这几个函数里面应该做什么工作?

- alloc:申请内存时调用.
- loadView:加载视图时调用.
- viewDidLoad;视图已经加载后调用.
- viewWillAppear:视图将要出现时调用.
- viewDidUnload:视图已经加载但是没有加载出来时调用.
- dealloc:销毁该视图时调用.
- init;初始化该视图时调用.

描述应用程序的启动顺序

1. 程序入口main函数创建UIApplication实例和UIApplication代理实例.
2. 在UIApplication代理实例中重写启动方法,设置根ViewController
3. 在第一ViewController中添加控件,实现应用程序界面.

为什么很多内置类如UITableViewControl的delegate属性都是assign而不是retain

- 防止循环引用.
- 如:对象A引用了对象B,对象B引用了对象C,对象C引用了对象B,这个时候B的引用计数是2,而C的引用计数是1,当A不再使用B的时候,就释放了B的所有权,这个时候C还引用对象B,所以B不会释放,引用计数为1,因为B也引用着对象C,B不释放,那么C也就不会被释放,所以他们的引用计数都为1,并且永远不会被释放,形成了循环引用.

使用UITableView的时候必须要实现的几种方法

2个数据源方法.分别是:

> - (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section;
> - (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath;

写一个遍历构造器

```
+(id)leftModelWith{ leftModel * model = [[[self alloc]]init](#)];   return model }
```

UIImage初始化图片的方法

- imageNamed:系统会先检查系统缓存中是否有该名字的image,如果有的话,则直接返回,如果没有,则先加载图像到缓存,然后再返回.
- initWithContentsOfFile:系统不会检查缓存,而直接从文件系统中记载并返回.
- imageWithCGImage:scale:orientation 当scale= 1的时候图像为原始大小,orientation指定绘制图像的方向.

person的retainCount值

> Person * per = Person allocinit]; self.person = per; 1或者2.看person是什么类型修饰的. alloc+1,assign+0,retain+1.

下面这段代码有何问题

@implementation Person

- (void)setAge:(int)newAge {
  ` self.age = newAge;

  ` }
  @end
  死循环

这段代码有什么问题,如何修改

for (int i = 0; i < someLargeNumber; i++) { NSString *string = @”Abc”; string = string lowercaseString; string = string stringByAppendingString:@"xyz"; NSLog(@“%@”, string); }

加入自动释放池@autoreleasepool; for (int i = 0; i < someLargeNumber; i++) { @antoreleasepool { NSString *string = @”Abc”; string = string lowercaseString; NSLog(@“%@”, string); } }

截取字符串"20 | http://www.baidu.com"中，"|"字符前面和后面的数据，分别输出它们。

"20 | http://www.baidu.com" componentSeparatedByString:@"|";

用obj-c 写一个冒泡排序.

NSMutableArray *ary = [@"1", @"2", @"3", @"4", @"6", @"5" mutableCopy];

```
for (int i = 0; i < ary.count - 1; i++) {
   for (int j = 0; j < ary.count - i - 1; j++) {
      if ([aryj integerValue] < [aryj + 1 integerValue]) {
         ary exchangeObjectAtIndex:j withObjectAtIndex:j + 1;
      }
   }
}
```

NSLog(@"%@", ary);

简述对UIView，UIWindow和CALayer的理解

- UIWindow是应用的窗口,继承于UIResponder
- UIView继承于UIView,是创建窗口中的一个视图,可以响应交互事件.一个程序只有一个主window,可以有多个window
- CALayer图层,一个view可有多个图层,不可以响应事件

写一个完整的代理,包括声明,实现

代理:遵守协议的对象.

@class MyView;

第一步:指定协议:(协议名:类名+Delegate)

@protocol MyViewDelegate <NSObject>

@required

-(void)changeViewBackgroudColor:(MyView *)view;

@optional

-(void)test;

@end

@interface MyView : UIView

第二步:指定代理

@property (nonatomic,assign)id<MyView> delegate;

@end

第三步:代理遵循协议.

第四步:代理实现协议里面的必须实现的方法和其他可选方法.

第五步:委托方通知代理开始执行方法.

分析json.xml的区别,底层如何实现

- Json:键值对.数据小,不复杂.便于解析,有框架支持,适合轻量级传输.作为数据包个数传输的时候效率更高.
- xml:标签套内容.xml数据两较大,比较复杂.适合大数据量的传输.xml有丰富的编码工具,比如:Dom4j,JDom.解析方式有两种,一是通过文芳模型解析,另外一种遍历节点.

ViewController的didReceiveMemoryWarning是在什么时候被调用的

- 当应用程序的内存使用接近系统的最大内存使用时,应用会向系统发送内存警告,这时候系统会调用方法向所有ViewController发送内存警告
- 打开系统相机.
- 加载高清图片.
  默认操作:把里面没有用的对象进行释放.

面向对象的三大特征,简单介绍

- 封装:代码模块化,方便以后调用.
- 继承:子类继承父类的所有方法和属性.
- 多态:父类指针指向子类对象.

用MRC实现set和get

- (void)setName:(NSString *)name{

```
    if(_name != name){
        _name retain;
        _name release;
        _name = name;
    }
}

- (NSString *)name{


    return [_name retainautorelease];
}
```

简述NotificationCenter.KVC,KVO,Delegate?并说明它们之间的区别?

- NotificationCenter:消息中心.消息通知.
- KVC:利用键-值间接访问类中的某个属性.

`self setValue:@"123" forKeyPath:@"name"; ` `NSLog(@"%@",self valueForKeyPath:@"name");
`

- KVO:利用键-路径间接访问类中的某个属性,也就是观察者模式(KVO+通知中心).基于KVC和通知中心,观察的是实例变量.
- Delegate:用于多个类之间的传值.

对MVC的理解,好处

- MVC:是一种架构.model:数据处理,view:视图显示,controller:逻辑控制,负责视图和模型之间的通信.
- 高类聚,低耦合,提高代码的复用性.

监测键盘的弹出

通知. [NSNotificationCenter defaultCenteraddObserver:self selector:@selector() name:UIKeyboardWillShowNotification object:nil];

介绍响应者链

- 当用户点击屏幕,能够产生响应的对象组成的链.
- 继承自NSResponder,响应者链能够中断.

传值方式

通知,单例,代理,属性,block

NSString * test = [NSData alloc init],test在编译时和运行时分别是什么类型的对象

编译时是NSString,运行时是NSData

OC中对象的交互是如何实现的

消息机制

目标-动作机制

Target - action

什么是沙盒?沙盒里包含哪些文件,如何获取文件路径

- 沙盒:程序可操作的磁盘空间,系统为之开辟.
- 包含了3个文件夹.

1.Documents:存放一些比较重要的文件,但是放入Documents中的文件不能过大. 2.Library :是一个资源库,存储一些不太重要的数据.里面包含了两个子文件夹,Caches文件夹,用于缓存, Preferences文件夹,系统偏好设置,用户对应用程序的设置,如密码.perferences路径无法找到,只能通过NSUserDefaults.如:NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES) firstObject;

介绍一下XMPP

- 基于XML的点对点通讯协议,实现通讯功能.
- 优点:可以跨平台开发.
- 缺点:丢包,只能发文字(图片发送发的是链接).

应用程序如何省电

获取请求不能过频.优化算法

递归的使用

-(NSInteger)digui:(NSInteger)i{ if (i>0) { return i*self digui:(i-1); }else{ return 1; } }

NSArray 和 NSMutableArray 的区别?多线程下那个更安全

- NSArray: 不可变数组.
- NSMutableArray: 可变数组.
- 多线程下NSArray更安全.

isKindOfClass,isMemberOfClass作用分别是什么

- isKindOfClass是某个类的实例或者子类的实例
- isMemberOfClass是某个类的实例

请分别写出SEL,id的意思

- SEL:选择器
- id:范类型
- OC中的对象就是C语言的指针

iPhone上,能被应用程序直接调用的系统程序是什么

- 能:相册,相机,通讯录,音乐.
- 不能:计算器,天气,日历,指南针

以.mm为扩展名的文件里,可以包含哪些代码

C++,C,OC

说说后台如何运行程序

- 在plist配置Application does not run in background设置NO(默认就是NO)的前提下.
- 添加required background modes,值是App registers for location updates和App plays auto or streams audio/video using AirPlay

sizeof和strlen的区别和联系

- sizeof:占用空间大小.
- strlen:字符串大小.

sprintf,strcpy,memcpy的功能?使用上要注意哪些地方?

- sprintf:将某些类型转换成字符串类型
- strcpy:拷贝字符串,会越界,'/0'
- memcpy:拷贝内存

写一个代码片实现输入一个字符串"20130322152830",输出一个NSDate类型的对象,打印该对象输出2013-03-11 15:28:32

NSString * str = @"20130322152832"; NSDateFormatter * format = [NSDateFormatter alloc]init]; format.dateFormat = @"yyyyMMddHHmmss";//设置格式 NSLog(@"%@",[format dateFromString:str dateByAddingTimeInterval:8*60*60]);

用变量a写出以下定义

a、一个整型数int a = 10 b、一个指向整型数的指针int *p = 10 c、一个指向指针的指针，它指向的指针是指向一个整型数int **p =10 d、一个有10个整型数的数组 int a[10] e、一个有10个指针的数组，该指针是指向一个整型数的int *a[10] f、一个指向有10个整型数数组的指针int *a = {1,2,3,4,5,6,7,8,9,10}; g、一个指向函数的指针，该函数有一个整型参数，并返回一个整型数 int *a(int b){   return b; }

cocoa和 cocoa touch

- cocoa包含Foundation和AppKit框架，可用于开发Mac OS X系统的应用程序
- cocoa touch包含Foundation和UIKit框架，可用于开发iPhone OS 系统的应用程序
- Cocoa是Mac OS X的开发环境，cocoa Touch是 Iphone OS的开发环境

网络从下往上分为几层

- 从下往上：物理层、数据链路层、网络层、传输层、会话层、表示层、应用层。
- IP 协议对应网络层，TCP 协议对应传输层，HTTP 协议对应于应用层。
- socket 则是对 TCP/IP协议的封装和应用。也可以说，TCP/IP协议是传输层协议，主要解决数据如何在网络中传输，而 HTTTP 是应用层协议，主要解决

多线程的底层实现

线程：进程中一个特立独行的控制单元（路径）。多线程：一个进程至少有一个线程，即主线程。①、Mach 是第一个以多线程方式处理任务的系统，因此多线程的底层实现机制就是基于 Mach 的线程。②、开发中很少用到 Mach 级的线程，因为 Mach级的线程没有提供多线程的基本特征，线程之间是独立的。④、开发中实现多线程的方案： NSThread、GCD、NSOperationQueue.NSOperation

线程之间怎么通信

①.performSelect:onThread:withObject:waitUntilDone: ②.NSMachPort

网络图片问题中怎么解决一个相同的网络地址重复请求的问题

利用字典:图片地址为 key, 下载操作为 value

用 NSOperation和 NSOperationQueue处理 A.B.C三个线程,要求执行完 A.B 后才能执行

```
//创建队列

NSOperationQueue * queue = [NSOperationQueue alloc]init];

//创建三个操作

NSOperation * A = [NSBlockOperation blockOperationWithBlock:^{ ](#)
    NSLog{@"A"};
}];

NSOperation * B = [NSBlockOperation blockOperationWithBlock:^{ ](#)
    NSLog{@"B"};
}];

NSOperation * C = [NSBlockOperation blockOperationWithBlock:^{ ](#)
    NSLog{@"C"};
}];

// 添加依赖

C addDependency:a;

C addDependency:b;

//执行操作

queue addOperation:a;

queue addOperation:b;

queue addOperation:c;
```

GCD内部怎么实现的

1. OS和 OSX 的核心是 XNU 内核, GCD是基于 XNU 内核实现的(是由苹果电脑发展起来的操作系统内核).
2. GCD 的 API 全部在 libdispatch 库中.
3. GCD 底层实现主要有 Dispatch Queue(管理 block)和 Dispatch Source(处理事件).

怎么保证多人开发进行内存泄露检查

使用Analuze进行代码的静态分析，为避免麻烦，多人开发尽量使用ARC

非自动内存管理情况下怎么做单例模式

- 创建一个单例对象的静态实例，并初始化为nil。
- 创建一个类的类工厂方法，当且仅当这个类的实例为nil时生成一个类的实例。
- 实现NScopying协议，覆盖allocWithZone：方法，确保用户在直接分配对象时，不会产生另一个对象。
- 覆盖release、autorelease、retain、retainCount方法，确保单例的状态。

对于类方法（静态方法）默认是autorelease的，所有类方法都会这样吗

系统自带的绝大数类方法返回的对象，都是经过autorelease

block在ARC中和MRC中的方法有何区别？需要注意什么？

1. 对于没有引用外部变量的Block，无论在ARC还是MRC下，类型都是*NSGlobalBlock*,这种类型的block可以理解为一种全局的block,不需要考虑作用域的问题。同时，对它进行Copy和Retain操作也是无效的。
2. 避免循环引用。

根据isa指针，block一共有3种类型的block

_NSConcreteGlobalBlock 全局静态

_NSConcreteStackBlock 保存在栈中，出函数作用域就销毁

NSConcreteMallocBlock 保存在堆中，retainCount == 0销毁

什么情况下会发生内存泄露和内存溢出

- 当程序在申请内存后，无法释放已经申请的内存空间（例如一个对象或者变量在用完后没有释放，这个对象就一直占用着内存），一次内存泄露可以忽略，但如果泄露过多的话，就会造成内存溢出。
- 当程序在申请内存时，但存入了更大的数据，出现内存溢出。

NSArray arrayWithobject<id>这个方法添加对象后，需要对这个数组进行释放操作吗

不需要，这个对象会被放到自动释放池中

自动释放池如何实现

自动释放池以栈的形式实现，当你创建一个新的自动释放池时，它将被添加到栈顶，当一个对象收到发送autorelease消息时，它将添加到当前线程的处于栈顶的自动释放池中，当自动释放池被回收时，它们从栈中被删除并且会给池子里所有对象都做一次release操作

KVO内部实现原理

1. KVO是基于runtime机制实现的。
2. 当某个类的对象第一次被观察时，系统就会在运行期动态的创建该类的一个派生类，在这个派生

类中重写基类中任何被观察属性的setter方法。
派生类在被重写setter方法中实现了真正的通知机制。（Person->NSKVONotification Person）

Foundation对象与Core Foundation对象有何区别

- Foundation对象是OC的，Core Foundation对象是C对象
- 数据类型之间的转换：
- ARC：*bridge_retained*、bridge_transfer
- 非ARC:*bridge*

不用第三变量，交换AB的值

A=A+B B=A-B A=A-B 或者 A=A^B B=A^B A=A^B

runtime实现的机制是什么？怎么用，一般用于干嘛

运行时机制，runtime库里面包含了跟类、成员变量、方法相关的API，比如获取类里面的所有成员变量，为类动态添加成员变量、动态改变类的方法实现，为类动态添加新的方法等，需要导入<objc/message.h><objc/message.h>

1. runtime,运行时机制，它是一套C语言库。
2. 实际上我们编写的所有OC代码，最终都是转换成为了runtime库的东西，比如类转换成了runtime库里面的结构体等数据类型，方法转换成了runtime库里面的C语言函数，平时调方法都是转成了objc_msgSend函数（所以说OC有个消息发送机制）
3. 因此，可以说runtime是OC的底层实现，是OC的幕后执行者。
4. 有了runtime库，能做什么呢？可以获取类里面的所有成员变量、为类动态的添加成员变量、动态的改变类的方法实现、为类动态添加新的方法等等。

是否使用Core Text 或者 Core Image

Core Text 随意修改文本的样式 图文混排（纯C语言） Core Image(滤镜处理) 能够调节图片的各种属性（对比度、色温、色差等）

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,他们的实现有何区别？如果用protocol和delegate来实现类似的功能可能吗？可能的话有何问题？不可能的话why?

- 通知比较灵活，一个通知能被多个对象接受，一个对象可以接受多个通知。
- 代理不交规范，但是代码较多（默认是一对一）
- KVO性能不好（底层会产生新的类），只能监听某个对象属性的变化，不推荐使用。

block内部的实现原理

Objective-C是对C语言的扩展，block的实现是基于指针和函数指针

怎么解决缓存池满的问题

iOS中不存在缓存池满的情况，通常在对象需要创建时才创建，比如UITableView中一般只会创建刚开始在屏幕中的cell，之后都是从缓存池里取，不会再创建新对象

控制器View的生命周期及相关函数是什么？你在开发中是如何使用的?

1. 首先判断控制器是否有视图，如果没有就调用loadView方法创建：通过storyBoard或者代码
2. 随后调用viewDidLoad，可以进行下一步的初始化操作，只会被调用一次。
3. 在视图显示之前调用viewWillAppear,该函数可以多次调用。
4. 视图viewDidAppear
5. 在布局变化前后，调用viewWill/DidLayoutSubViews处理相关信息。

有些图片加载比较慢怎么处理？你是怎么优化程序的性能的?

1. 图片下载放在异步线程。
2. 图片下载过程使用占位图片。
3. 如果图片比较大，可以使用多线程断点下载。

App需要加载大量数据，给服务器发送请求，但是服务器卡住了怎么办

设置请求超时，给用户提示请求超时，根据用户操作再次请求

SDWebImage具体如何实现

其实就是沙盒缓存机制，主要由三块组成：内存图片缓存，内存操作缓存，磁盘沙盒缓存。

1. 利用NSOperationQueue和NSOperation下载图片，还使用了GCD（解析GIF图片）。
2. 利用URL作为key，NSOperation作为value.
3. 利用URL作为key，UIImage作为value

AFNetWorking实现原理

基于NSURL.采用block的方法处理请求，直接返回的是json、XML数据。AFN直接操作对象是AFHTTPClient,是一个实现了NSCoding和NSCopying协议的NSObject子类。AFGTTPClient是一个封装了一系列操作方法的工具类。AFN默认没有封装同步请求，如果开发者需要使用同步请求，需要重写相关的方法（getPath:parameters:failure），对AFHTTPRequestOperation进行同步处理。

Extension 是什么？能列举几个常用的 Extension 么?

Extension是扩展,没有分类名字,是一种特殊的分类,类扩展可以扩展属性,成员变量和方法 常用的扩展是在.m文件中声明私有属性和方法,这里不知道还哪些,请大家补充

App Bundle 里面都有什么

- Info.plist:此文件包含了应用程序的配置信息.系统依赖此文件以获取应用程序的相关信息
- 可执行文件:此文件包含应用程序的入口和通过静态连接到应用程序target的代码
- 资源文件:图片,声音文件一类的
- 其他:可以嵌入定制的数据资源

iOS 的签名机制大概是怎样的

假设，我们有一个APP需要发布，为了防止中途篡改APP内容，保证APP的完整性，以及APP是由指定的私钥发的。首先，先将APP内容通过摘要算法，得到摘要，再用私钥对摘要进行加密得到密文，将源文本、密文、和私钥对应的公钥一并发布即可。那么如何验证呢？ 验证方首先查看公钥是否是私钥方的，然后用公钥对密文进行解密得到摘要，将APP用同样的摘要算法得到摘要，两个摘要进行比对，如果相等那么一切正常。这个过程只要有一步出问题就视为无效。

iOS 7的多任务添加了哪两个新的 API

- 后台获取（Background Fetch):后台获取使用场景是用户打开应用之前就使app有机会执行代码来获取数据，刷新UI。这样在用户打开应用的时候，最新的内容将已然呈现在用户眼前，而省去了所有的加载过程。
- 推送唤醒（Remote Notifications):使用场景是使设备在接收到远端推送后让系统唤醒设备和我们的后台应用，并先执行一段代码来准备数据和UI，然后再提示用户有推送。这时用户如果解锁设备进入应用后将不会再有任何加载过程，新的内容将直接得到呈现

UIScrollView 大概是如何实现的，它是如何捕捉、响应手势的?

对UIScrollView的理解是frame就是他的contentSize,bounds就是他的可视范围,通过改变bounds从而达到让用户误以为在滚动,以下是一个简单的UIScrollView实现

+load 和 +initialize 的区别是什么

+(void)load; 当类对象被引入项目时, runtime 会向每一个类对象发送 load 消息 load 方法会在每一个类甚至分类被引入时仅调用一次,调用的顺序：父类优先于子类, 子类优先于分类 load 方法不会被类自动继承 +(void)initialize; 也是在第一次使用这个类的时候会调用这个方法

如何让 Category 支持属性

使用runtime可以实现

NSOperation 相比于 GCD 有哪些优势

- 提供了在 GCD 中不那么容易复制的有用特性。
- 可以很方便的取消一个NSOperation的执行
- 可以更容易的添加任务的依赖关系
- 提供了任务的状态：isExecuteing, isFinished.

strong / weak / unsafe_unretained 的区别

- weak只能修饰OC对象,使用weak不会使计数器加1,对象销毁时修饰的对象会指向nil
- strong等价与retain,能使计数器加1,且不能用来修饰数据类型
- unsafe_unretained等价与assign,可以用来修饰数据类型和OC对象,但是不会使计数器加1,且对象销毁时也不会将对象指向nil,容易造成野指针错误_

如何为 Class 定义一个对外只读对内可读写的属性

在头文件中将属性定义为readonly,在.m文件中将属性重新定义为readwrite

Objective-C 中，meta-class 指的是什么

meta-class 是 Class 对象的类,为这个Class类存储类方法,当一个类发送消息时,就去那个类对应的meta-class中查找那个消息,每个Class都有不同的meta-class,所有的meta-class都使用基类的meta-class(假如类继承NSObject,那么他所对应的meta-class也是NSObject)作为他们的类

UIView 和 CALayer 之间的关系

- UIView显示在屏幕上归功于CALayer，通过调用drawRect方法来渲染自身的内容，调节CALayer属性可以调整UIView的外观，UIView继承自UIResponder，CALayer不可以响应用户事件
- UIView是iOS系统中界面元素的基础，所有的界面元素都继承自它。它内部是由Core Animation来实现的，它真正的绘图部分，是由一个叫CALayer(Core Animation Layer)的类来管理。

UIView本身，更像是一个CALayer的管理器，访问它的根绘图和坐标有关的属性，如frame，bounds等，实际上内部都是访问它所在CALayer的相关属性
- UIView有个layer属性，可以返回它的主CALayer实例，UIView有一个layerClass方法，返回主layer所使用的类，UIView的子类，可以通过重载这个方法，来让UIView使用不同的CALayer来显示

+UIView animateWithDuration:animations:completion: 内部大概是如何实现的

animateWithDuration:这就等于创建一个定时器 animations:这是创建定时器需要实现的SEL completion:是定时器结束以后的一个回调block

什么时候会发生「隐式动画」

当改变CALayer的一个可做动画的属性，它并不能立刻在屏幕上体现出来.相反，它是从先前的值平滑过渡到新的值。这一切都是默认的行为，你不需要做额外的操作,这就是隐式动画

frame 和 bounds 的区别是什么

- frame相对于父视图,是父视图坐标系下的位置和大小。bounds相对于自身,是自身坐标系下的位置和大小。
- frame以父控件的左上角为坐标原点，bounds以自身的左上角为坐标原点

如何把一张大图缩小为1/4大小的缩略图

data = UIImageJPEGRepresentation(image, 0.25)

category 和 extension 的区别

- category： 分类有名字，类扩展没i有分类名字，是一种特殊的分类
- extension： 分类只能扩展方法（属性仅仅是声明，并没真正实现），类扩展可以扩展属性、成员变量和方法

define 和 const常量有什么区别

- define在预处理阶段进行替换，const常量在编译阶段使用
- 宏不做类型检查，仅仅进行替换，const常量有数据类型，会执行类型检查
- define不能调试，const常量可以调试
- define定义的常量在替换后运行过程中会不断地占用内存，而const定义的常量存储在数据段只有一份copy，效率更高
- define可以定义一些简单的函数，const不可以

block和weak修饰符的区别

- __block不管是ARC还是MRC模式下都可以使用，可以修饰对象，也可以修饰基本数据类型
- __weak只能在ARC模式下使用，只能修饰对象（NSString），不能修饰基本数据类型
- block修饰的对象可以在block中被重新赋值，weak修饰的对象不可以

static关键字的作用

- 函数（方法）体内 static 变量的作用范围为该函数体，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它的模块内；
- 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；
- 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

堆和栈的区别

*从管理方式来讲

1. 对于栈来讲，是由编译器自动管理，无需我们手工控制；
2. 对于堆来说，释放工作由程序员控制，容易产生内存泄露(memory leak)

- 从申请大小大小方面讲

1. 栈空间比较小
2. 堆空间比较大

- 从数据存储方面来讲

1. 栈空间中一般存储基本类型，对象的地址
2. 堆空间一般存放对象本身，block的copy等

Objective-C使用什么机制管理对象内存

- MRC 手动引用计数
- ARC 自动引用计数,现在通常ARC
- 通过 retainCount 的机制来决定对象是否需要释放。 每次 runloop 的时候，都会检查对象的 retainCount，如果retainCount 为 0，说明该对象没有地方需要继续使用了，可以释放掉了

ARC通过什么方式帮助开发者管理内存

通过编译器在编译的时候,插入类似内存管理的代码

ARC是为了解决什么问题诞生的

- 首先解释ARC: automatic reference counting自动引用计数
- 了解MRC的缺点

1. 在MRC时代当我们要释放一个堆内存时，首先要确定指向这个堆空间的指针都被release了
2. 释放指针指向的堆空间，首先要确定哪些指针指向同一个堆，这些指针只能释放一次(MRC下即谁创建，谁释放，避免重复释放)
3. 模块化操作时，对象可能被多个模块创建和使用，不能确定最后由谁去释放
4. 多线程操作时，不确定哪个线程最后使用完毕

- 综上所述，MRC有诸多缺点，很容易造成内存泄露和坏内存的问题，这时苹果为尽量解决这个问题，从而诞生了ARC

ARC下还会存在内存泄露吗

- 循环引用会导致内存泄露

- Objective-C对象与CoreFoundation对象进行桥接的时候如果管理不当也会造成内存泄露
- CoreFoundation中的对象不受ARC管理，需要开发者手动释放

什么情况使用weak关键字，相比assign有什么不同

- 首先明白什么情况使用weak关键字?

1. 在ARC中,在有可能出现循环引用的时候,往往要通过让其中一端使用weak来解决,比如:delegate代理属性，代理属性也可使用assign
2. 自身已经对它进行一次强引用,没有必要再强引用一次,此时也会使用weak,自定义IBOutlet控件属性一般也使用weak；当然，也可以使用strong，但是建议使用weak

- weak 和 assign的不同点

1. weak策略在属性所指的对象遭到摧毁时，系统会将weak修饰的属性对象的指针指向nil，在OC给nil发消息是不会有什么问题的；如果使用assign策略在属性所指的对象遭到摧毁时，属性对象指针还指向原来的对象，由于对象已经被销毁，这时候就产生了野指针，如果这时候在给此对象发送消息，很容造成程序奔溃
2. assigin 可以用于修饰非OC对象,而weak必须用于OC对象

@property 的本质是什么

@property其实就是在编译阶段由编译器自动帮我们生成ivar成员变量，getter方法，setter方法 ivar、getter、setter是如何生成并添加到这个类中的？ • 使用“自动合成”(autosynthesis) • 这个过程由编译器在编译阶段执行自动合成，所以编辑器里看不到这些“合成方法”(synthesized method)的源代码 • 除了生成getter、setter方法之外，编译器还要自动向类中添加成员变量（在属性名前面加下划线，以此作为实例变量的名字）

KVO内部实现原理

- KVO是基于runtime机制实现的
- 当某个类的属性对象第一次被观察时，系统就会在运行期动态地创建该类的一个派生类，在这个派生类中重写基类中任何被观察属性的setter 方法。派生类在被重写的setter方法内实现真正的通知机制
- 如果原类为Person，那么生成的派生类名为NSKVONotifying_Person
- 每个类对象中都有一个isa指针指向当前类，当一个类对象的第一次被观察，那么系统会偷偷将isa指针指向动态生成的派生类，从而在给被监控属性赋值时执行的是派生类的setter方法
- 键值观察通知依赖于NSObject 的两个方法: willChangeValueForKey: 和 didChangevlueForKey:；在一个被观察属性发生改变之前， willChangeValueForKey: 一定会被调用，这就 会记录旧的值。而当改变发生后，didChangeValueForKey: 会被调用，继而 observeValueForKey:ofObject:change:context: 也会被调用。
- 补充：KVO的这套实现机制中苹果还偷偷重写了class方法，让我们误认为还是使用的当前类，从而达到隐藏生成的派生类

KVC的keyPath中的集合运算符如何使用

- 必须用在集合对象上或普通对象的集合属性上
- 简单集合运算符有@avg， @count ， @max ， @min ，@sum
- 格式 @"@sum.age" 或 @"集合属性.@max.age"？？？

KVC和KVO的keyPath一定是属性么

可以是成员变量

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?

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。

#import 跟#include 又什么区别，@class呢, #import<> 跟 #import""又什么区别

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。

属性readwrite，readonly，assign，retain，copy，nonatomic 各是什么作用，在那种情况下用?

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

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

- (void) setName:(NSString*) str
  {
  [str retain];
  [name release];
  name = str;
  }
- (void)setName:(NSString *)str
  {
  id t = [str copy];
  [name release];
  name = t;
  }

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?

编译时是NSString的类型;运行时是NSData类型的对象

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

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。

id 声明的对象有什么特性

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

Objective-C如何对内存管理的,说说你的看法和解决方法

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 allocinitialWithFormat:@”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.

原子(atomic)跟非原子(non-atomic)属性有什么区别?

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。

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?

========== NSMutableArray* ary = [NSMutableArray array retain]; NSString *str = NSString stringWithFormat:@"test"; strretain; aryaddObject:str; NSLog(@"%@%d",str,str retainCount); strretain; strrelease; strrelease; NSLog(@"%@%d",str,str retainCount); aryremoveAllObjects;

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

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?

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，

如何对iOS设备进行性能测试

Profile-> Instruments ->Time Profiler

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

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:

描述一下iOS SDK中如何实现MVC的开发模式

MVC是模型、试图、控制开发模式，对于iOS SDK，所有的View都是视图层的，它应该独立于模型层，由视图控制层来控制。所有的用户数据都是模型层，它应该独立于视图。所有的ViewController都是控制层，由它负责控制视图，访问模型数据。

浅复制和深复制的区别

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。

类别的作用?继承和类别在实现中有何区别?

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。

类别和类扩展的区别

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。

oc中的协议和java中的接口概念有何不同

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)”

什么是KVO和KVC

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:(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。
  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。

代理的作用

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。

oc中可修改和不可以修改类型。

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

我们说的oc是动态运行时语言是什么意思

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

通知和协议的不同之处

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。

推送消息

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。

关于多态性

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。

说说响应链

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

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)

方法和选择器有何不同

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。

OC的垃圾回收机制

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

NSOperation queue

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。

什么是延迟加载

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

是否在一个视图控制器中嵌入两个tableview控制器

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

一个tableView是否可以关联两个不同的数据源?你会怎么处理

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

什么时候使用NSMutableArray，什么时候使用NSArray

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。

在应用中可以创建多少autorelease对象，是否有限制

无

如果我们不创建内存池，是否有内存池提供给我们

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

什么时候需要在程序中创建内存池

用户自己创建的数据线程，则需要创建该线程的内存池

类NSObject的那些方法经常被使用

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

什么是简便构造方法

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。

如何使用Xcode设计通用应用

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。

UIView的动画效果有那些

有很多，如 UIViewAnimationOptionCurveEaseInOut UIViewAnimationOptionCurveEaseIn UIViewAnimationOptionCurveEaseOut UIViewAnimationOptionTransitionFlipFromLeft UIViewAnimationOptionTransitionFlipFromRight UIViewAnimationOptionTransitionCurlUpUIViewAnimationOptionTransitionCurlDown

在iPhone应用中如何保存数据

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

什么是coredata

coredata是苹果提供一套数据保存框架，其基于SQlite

什么是NSManagedObject模型

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。

什么是NSManagedobjectContext

NSManagedobjectContext对象负责应用和数据库之间的交互

什么是谓词

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;

和coredata一起有哪几种持久化存储机制

存入到文件、 存入到NSUserDefaults(系统plist文件中)、存入到Sqlite文件数据库

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。

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).

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。

什么是block

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。

block 实现原理

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)。 下面是两篇很好的介绍block实现的博文 iOS中block实现的探究 谈Objective-C Block的实现 3 block的使用 使用实例 cocoaTouch框架下动画效果的Block的调用 使用typed声明block typedef void(^didFinishBlock)(NSObject *ob); 这就声明了一个didFinishBlock类型的block， 然后便可用 @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。

原理篇

runtime怎么添加属性、方法等

- ivar表示成员变量
- class_addIvar
- class_addMethod
- class_addProperty
- class_addProtocol
- class_replaceProperty

是否可以把比较耗时的操作放在NSNotificationCenter中

- 首先必须明确通知在哪个线程中发出，那么处理接受到通知的方法也在这个线程中调用
- 如果在异步线程发的通知，那么可以执行比较耗时的操作；
- 如果在主线程发的通知，那么就不可以执行比较耗时的操作

runtime 如何实现 weak 属性

- 首先要搞清楚weak属性的特点

> weak策略表明该属性定义了一种“非拥有关系” (nonowning relationship)。 为这种属性设置新值时，设置方法既不保留新值，也不释放旧值。此特质同assign类似; 然而在属性所指的对象遭到摧毁时，属性值也会清空(nil out)

- 那么runtime如何实现weak变量的自动置nil?

> runtime对注册的类，会进行布局，会将 weak 对象放入一个 hash 表中。 用 weak 指向的对象内存地址作为 key，当此对象的引用计数为0的时候会调用对象的 dealloc 方法， 假设 weak 指向的对象内存地址是a，那么就会以a为key，在这个 weak hash表中搜索，找到所有以a为key的 weak 对象，从而设置为 nil。

一个Objective-C对象如何进行内存布局？（考虑有父类的情况）

- 所有父类的成员变量和自己的成员变量都会存放在该对象所对应的存储空间中
- 父类的方法和自己的方法都会缓存在类对象的方法缓存中，类方法是缓存在元类对象中
- 每一个对象内部都有一个isa指针,指向他的类对象,类对象中存放着本对象的如下信息

1. 对象方法列表
2. 成员变量的列表
3. 属性列表

一个objc对象的isa的指针指向什么？有什么作用?

- 每一个对象内部都有一个isa指针，这个指针是指向它的真实类型
- 根据这个指针就能知道将来调用哪个类的方法

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）

- 每一个类对象中都一个对象方法列表（对象方法缓存）
- 类方法列表是存放在类对象中isa指针指向的元类对象中（类方法缓存）
- 方法列表中每个方法结构体中记录着方法的名称,方法实现,以及参数类型，其实selector本质就是方法名称,通过这个方法名称就可以在方法列表中找到对应的方法实现.
- 当我们发送一个消息给一个NSObject对象时，这条消息会在对象的类对象方法列表里查找
- 当我们发送一个消息给一个类时，这条消息会在类的Meta Class对象的方法列表里查找
  objc中的类方法和实例方法有什么本质区别和联系
- 类方法：

1. 类方法是属于类对象的
2. 类方法只能通过类对象调用
3. 类方法中的self是类对象
4. 类方法可以调用其他的类方法
5. 类方法中不能访问成员变量
6. 类方法中不能直接调用对象方法
7. 类方法是存储在元类对象的方法缓存中

- 实例方法：

1. 实例方法是属于实例对象的
2. 实例方法只能通过实例对象调用
3. 实例方法中的self是实例对象
4. 实例方法中可以访问成员变量

5. 实例方法中直接调用实例方法
6. 实例方法中可以调用类方法(通过类名)
7. 实例方法是存放在类对象的方法缓存中

使用runtime Associate方法关联的对象，需要在主对象dealloc的时候释放么

- 无论在MRC下还是ARC下均不需要
- 被关联的对象在生命周期内要比对象本身释放的晚很多，它们会在被 NSObject -dealloc 调用的object_dispose()方法中释放
- 补充：对象的内存销毁时间表，分四个步骤

1.调用 -release ： 引用计数变为零

- 对象正在被销毁，生命周期即将结束.
- 不能再有新的 __weak 弱引用，否则将指向 nil.
- 调用 self dealloc

2. 父类调用 -dealloc

- 继承关系中最直接继承的父类再调用 -dealloc
- 如果是 MRC 代码 则会手动释放实例变量们（iVars）
- 继承关系中每一层的父类 都再调用 -dealloc

3. NSObject 调 -dealloc

- 只做一件事：调用 Objective-C runtime 中的 object_dispose() 方法

4. 调用 object_dispose()

- 为 C++ 的实例变量们（iVars）调用 destructors
- 为 ARC 状态下的 实例变量们（iVars） 调用 -release
- 解除所有使用 runtime Associate方法关联的对象
- 解除所有 __weak 引用
- 调用 free()

runloop和线程有什么关系?

- 每条线程都有唯一的一个RunLoop对象与之对应的
- 主线程的RunLoop是自动创建并启动
- 子线程的RunLoop需要手动创建
- 子线程的RunLoop创建步骤如下：

1. 在子线程中调用NSRunLoop currentRunLoop创建RunLoop对象（懒加载，只创建一次）
2. 获得RunLoop对象后要调用run方法来启动一个运行循环

runloop的mode作用是什么?

- 用来控制一些特殊操作只能在指定模式下运行，一般可以通过指定操作的运行mode来控制执行时机，以提高用户体验
- 系统默认注册了5个Mode

1. kCFRunLoopDefaultMode：App的默认Mode，通常主线程是在这个Mode下运行，对应OC中的：NSDefaultRunLoopMode

2. UITrackingRunLoopMode：界面跟踪 Mode，用于 ScrollView 追踪触摸滑动，保证界面滑动时不受其他Mode影响
3. kCFRunLoopCommonModes:这是一个标记Mode，不是一种真正的Mode，事件可以运行在所有标有common modes标记的模式中，对应OC中的NSRunLoopCommonModes，带有common modes标记的模式有：UITrackingRunLoopMode和kCFRunLoopDefaultMode
4. UIInitializationRunLoopMode：在启动 App时进入的第一个 Mode，启动完成后就不再使用
5. GSEventReceiveRunLoopMode：接受系统事件的内部Mode，通常用不到

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?

- 这里强调一点：在主线程中以+scheduledTimerWithTimeInterval...的方式触发的timer默认是运行在NSDefaultRunLoopMode模式下的，当滑动页面上的列表时，进入了UITrackingRunLoopMode模式，这时候timer就会停止
- 可以修改timer的运行模式为NSRunLoopCommonModes，这样定时器就可以一直运行了
- 以下是我的笔记补充：

1. 在子线程中通过scheduledTimerWithTimeInterval:...方法来构建NSTimer
2. 方法内部已经创建NSTimer对象，并加入到RunLoop中，运行模式为NSDefaultRunLoopMode
3. 由于Mode有timer对象，所以RunLoop就开始监听定时器事件了，从而开始进入运行循环
4. 这个方法仅仅是创建RunLoop对象，并不会主动启动RunLoop，需要再调用run方法来启动

- 如果在主线程中通过scheduledTimerWithTimeInterval:...方法来构建NSTimer，就不需要主动启动RunLoop对象，因为主线程的RunLoop对象在程序运行起来就已经被启动了

猜想runloop内部是如何实现的?

- 从字面意思看：运行循环、跑圈；
- 本质：内部就是do-while循环，在这个循环内部不断地处理各种事件(任务)，比如：Source、Timer、Observer；
- 每条线程都有唯一一个RunLoop对象与之对应，主线程的RunLoop默认已经启动，子线程的RunLoop需要手动启动；
- 每次RunLoop启动时，只能指定其中一个 Mode，这个Mode被称作 CurrentMode，如果需要切换Mode，只能退出Loop，再重新指定一个Mode进入，这样做主要是为了隔离不同Mode中的Source、Timer、Observer，让其互不影响；

不手动指定autoreleasepool的前提下，一个autorealese对象在什么时刻释放?

- 分两种情况：手动干预释放时机、系统自动去释放

1. 手动干预释放时机：指定autoreleasepool就是所谓的：当前作用域大括号结束时就立即释放
2. 系统自动去释放：不手动指定autoreleasepool，Autorelease对象会在当前的 runloop 迭代结释放

苹果为什么要废弃dispatch_get_current_queue

• 容易误用造成死锁 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）• 必须是并发队列才起作用 • 需求分析 ○ 首先，分别异步执行2个耗时的操作 ○ 其次，等2个异步操作都执行完毕后，再回到主线程执行一些操作 • 使用队列组实现上面的需求

// 创建队列组 dispatch_group_t group = dispatch_group_create(); // 获取全局并发队列 dispatch_queue_t queue = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0); // 往队列组中添加耗时操作 dispatch_group_async(group, queue, ^{ // 执行耗时的异步操作1 }); // 往队列组中添加耗时操作 dispatch_group_async(group, queue, ^{ // 执行耗时的异步操作2 }); // 当并发队列组中的任务执行完毕后才会执行这里的代码 dispatch_group_notify(group, queue, ^{ // 如果这里还有基于上面两个任务的结果继续执行一些代码，建议还是放到子线程中，等代码执行完毕后在回到主线程 // 回到主线程 dispatch_async(group, dispatch_get_main_queue(), ^{ // 执行相关代码... }); });

dispatch_barrier_async的作用是什么

• 函数定义

dispatch_barrier_async(dispatch_queue_t queue, dispatch_block_t block); • 必须是并发队列，要是串行队列，这个函数就没啥意义了 • 注意：这个函数的第一个参数queue不能是全局的并发队列 • 作用：在它前面的任务执行结束后它才执行，在它后面的任务等它执行完成后才会执 • 示例代码

-(void)barrier { dispatch_queue_t queue = dispatch_queue_create("12342234", DISPATCH_QUEUE_CONCURRENT); dispatch_async(queue, ^{ NSLog(@"----1-----%@", NSThread currentThread); }); dispatch_async(queue, ^{ NSLog(@"----2-----%@", NSThread currentThread); }); // 在它前面的任务执行结束后它才执行，在它后面的任务等它执行完成后才会执行 dispatch_barrier_async(queue, ^{ NSLog(@"----barrier-----%@", NSThread currentThread); }); dispatch_async(queue, ^{ NSLog(@"----3-----%@", NSThread currentThread); }); dispatch_async(queue, ^{ NSLog(@"----4-----%@", NSThread currentThread); }); }

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");
  }
  • 答案：主线程死锁

lldb（gdb）常用的调试命令

- po：打印对象，会调用对象description方法。是print-object的简写
- expr：可以在调试时动态执行指定表达式，并将结果打印出来，很有用的命令
- print：也是打印命令，需要指定类型
- bt：打印调用堆栈，是thread backtrace的简写，加all可打印所有thread的堆栈
- br l：是breakpoint list的简写

BAD_ACCESS在什么情况下出现

- 访问一个僵尸对象，访问僵尸对象的成员变量或者向其发消息

- 死循环

如何调试BAD_ACCESS错误

- 设置全局断点快速定位问题代码所在行

简述下Objective-C中调用方法的过程（runtime）

- Objective-C是动态语言，每个方法在运行时会被动态转为消息发送，即：objc_msgSend(receiver, selector)，整个过程介绍如下：

1. objc在向一个对象发送消息时，runtime库会根据对象的isa指针找到该对象实际所属的类
2. 然后在该类中的方法列表以及其父类方法列表中寻找方法运行
3. 如果，在最顶层的父类（一般也就NSObject）中依然找不到相应的方法时，程序在运行时会挂掉并抛出异常unrecognized selector sent to XXX
4. 但是在这之前，objc的运行时会给出三次拯救程序崩溃的机会，这三次拯救程序奔溃的说明见问题《什么时候会报unrecognized selector的异常》中的说明

- 补充说明：Runtime 铸就了Objective-C 是动态语言的特性，使得C语言具备了面向对象的特性，在程序运行期创建，检查，修改类、对象及其对应的方法，这些操作都可以使用runtime中的对应方法实现。

什么是method swizzling（俗称黑魔法）

- 简单说就是进行方法交换
- 在Objective-C中调用一个方法，其实是向一个对象发送消息，查找消息的唯一依据是selector的名字。利用Objective-C的动态特性，可以实现在运行时偷换selector对应的方法实现，达到给方法挂钩的目的
- 每个类都有一个方法列表，存放着方法的名字和方法实现的映射关系，selector的本质其实就是方法名，IMP有点类似函数指针，指向具体的Method实现，通过selector就可以找到对应的IMP
- 交换方法的几种实现方式
- 利用 method_exchangeImplementations 交换两个方法的实现
- 利用 class_replaceMethod 替换方法的实现
- 利用 method_setImplementation 来直接设置某个方法的IMP

objc中向一个nil对象发送消息将会发生什么

- 在Objective-C中向nil发送消息是完全有效的——只是在运行时不会有任何作用

1. 如果一个方法返回值是一个对象，那么发送给nil的消息将返回0(nil)
   2.如果方法返回值为指针类型，其指针大小为小于或者等于sizeof(void*)
2. float，double，long double 或者long long的整型标量，发送给nil的消息将返回0
3. 如果方法返回值为结构体,发送给nil的消息将返回0。结构体中各个字段的值将都是0
4. 如果方法的返回值不是上述提到的几种情况，那么发送给nil的消息的返回值将是未定义的

- 具体原因分析
- objc是动态语言，每个方法在运行时会被动态转为消息发送，即：objc_msgSend(receiver, selector)

三次拯救程序崩溃的机会

- Method resolution

1. objc运行时会调用+resolveInstanceMethod:或者 +resolveClassMethod:，让你有机会提供一个函数实现。
2. 如果你添加了函数并返回 YES，那运行时系统就会重新启动一次消息发送的过程
3. 如果 resolve 方法返回 NO ，运行时就会移到下一步，消息转发

- Fast forwarding

1. 如果目标对象实现了-forwardingTargetForSelector:，Runtime 这时就会调用这个方法，给你把这个消息转发给其他对象的机会
2. 只要这个方法返回的不是nil和self，整个消息发送的过程就会被重启，当然发送的对象会变成你返回的那个对象。
3. 否则，就会继续Normal Fowarding。
4. 这里叫Fast，只是为了区别下一步的转发机制。因为这一步不会创建任何新的对象，但Normal forwarding转发会创建一个NSInvocation对象，相对Normal forwarding转发更快点，所以这里叫Fast forwarding

- Normal forwarding

1. 这一步是Runtime最后一次给你挽救的机会。
2. 首先它会发送-methodSignatureForSelector:消息获得函数的参数和返回值类型。
3. 如果-methodSignatureForSelector:返回nil，Runtime则会发出-doesNotRecognizeSelector:消息，程序这时也就挂掉了。
4. 如果返回了一个函数签名，Runtime就会创建一个NSInvocation对象并发送-forwardInvocation:消息给目标对象

HTTP协议中POST方法和GET方法有那些区别?

- GET用于向服务器请求数据，POST用于提交数据
- GET请求，请求参数拼接形式暴露在地址栏，而POST请求参数则放在请求体里面，因此GET请求不适合用于验证密码等操作
- GET请求的URL有长度限制，POST请求不会有长度限制
  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），是否也考虑循环引用问题?
- 系统的某些block api中，UIView的block版本写动画时不需要考虑，但也有一些api 需要考虑

UI

Size Classes

对屏幕进行分类

UIView和CALayer是什么关系

- UIView显示在屏幕上归功于CALayer，通过调用drawRect方法来渲染自身的内容，调节CALayer属性可以调整UIView的外观，UIView继承自UIResponder，比起CALayer可以响应用户事件，Xcode6之后可以方便的通过视图调试功能查看图层之间的关系
- UIView是iOS系统中界面元素的基础，所有的界面元素都继承自它。它内部是由Core Animation来实现的，它真正的绘图部分，是由一个叫CALayer(Core Animation Layer)的类来管理。UIView本身，更像是一个CALayer的管理器，访问它的跟绘图和坐标有关的属性，如frame，

bounds等，实际上内部都是访问它所在CALayer的相关属性
- UIView有个layer属性，可以返回它的主CALayer实例，UIView有一个layerClass方法，返回主layer所使用的类，UIView的子类，可以通过重载这个方法，来让UIView使用不同的CALayer来显示

IBOutlet连出来的视图属性为什么可以被设置成weak

因为父控件的subViews数组已经对它有一个强引用

IB中User Defined Runtime Attributes如何使用?

- User Defined Runtime Attributes是一个不被看重但功能非常强大的的特性，它能够通过KVC的方式配置一些你在interface builder中不能配置的属性
- 当你希望在IB中作尽可能多得事情，这个特性能够帮助你编写更加轻量级的viewcontroller

pushViewController和presentViewController有什么区别

- 两者都是在多个试图控制器间跳转的函数
- presentViewController提供的是一个模态视图控制器(modal)
- pushViewController提供一个栈控制器数组，push/pop

请简述UITableView的复用机制

- 每次创建cell的时候通过dequeueReusableCellWithIdentifier:方法创建cell，它先到缓存池中找指定标识的cell，如果没有就直接返回nil
- 如果没有找到指定标识的cell，那么会通过initWithStyle:reuseIdentifier:创建一个cell
- 当cell离开界面就会被放到缓存池中，以供下次复用

如何高性能的给 UIImageView 加个圆角?

不好的解决方案
使用下面的方式会强制Core Animation提前渲染屏幕的离屏绘制, 而离屏绘制就会给性能带来负面影响，会有卡顿的现象出现

self.view.layer.cornerRadius = 5; self.view.layer.masksToBounds = YES; 正确的解决方案：使用绘图技术

- (UIImage *)circleImage

  {

    // NO代表透明

    UIGraphicsBeginImageContextWithOptions(self.size, NO, 0.0);

    // 获得上下文

    CGContextRef ctx = UIGraphicsGetCurrentContext();

    // 添加一个圆

```
    CGRect rect = CGRectMake(0, 0, self.size.width, self.size.height);

    CGContextAddEllipseInRect(ctx, rect);

    // 裁剪

    CGContextClip(ctx);

    // 将图片画上去

    self drawInRect:rect;

    UIImage *image = UIGraphicsGetImageFromCurrentImageContext();

    // 关闭上下文

    UIGraphicsEndImageContext();

    return image;

}
```

- 还有一种方案：使用了贝塞尔曲线"切割"个这个图片, 给UIImageView 添加了的圆角，其实也是通过绘图技术来实现的

```
UIImageView *imageView = [UIImageView alloc initWithFrame:CGRectMake(0, 0, 100, 100)];
imageView.center = CGPointMake(200, 300);
UIImage *anotherImage = UIImage imageNamed:@"image";
UIGraphicsBeginImageContextWithOptions(imageView.bounds.size, NO, 1.0);
[[UIBezierPath bezierPathWithRoundedRect:imageView.bounds
](#)                    cornerRadius:50] addClip];
anotherImage drawInRect:imageView.bounds;
imageView.image = UIGraphicsGetImageFromCurrentImageContext();
UIGraphicsEndImageContext();
self.view addSubview:imageView;
```

使用drawRect有什么影响

- drawRect方法依赖Core Graphics框架来进行自定义的绘制
- 缺点：它处理touch事件时每次按钮被点击后，都会用setNeddsDisplay进行强制重绘；而且不止一次，每次单点事件触发两次执行。这样的话从性能的角度来说，对CPU和内存来说都是欠佳的。特别是如果在我们的界面上有多个这样的UIButton实例，那就会很糟糕了
- 这个方法的调用机制也是非常特别. 当你调用 setNeedsDisplay 方法时, UIKit 将会把当前图层标记为dirty,但还是会显示原来的内容,直到下一次的视图渲染周期,才会将标记为 dirty 的图层重新建立Core Graphics上下文,然后将内存中的数据恢复出来, 再使用 CGContextRef 进行绘制

描述下SDWebImage里面给UIImageView加载图片的逻辑

- SDWebImage 中为 UIImageView 提供了一个分类UIImageView+WebCache.h, 这个分类中有一

个最常用的接口sd_setImageWithURL:placeholderImage:，会在真实图片出现前会先显示占位图片，当真实图片被加载出来后在替换占位图片

- 加载图片的过程大致如下：

1. 首先会在 SDWebImageCache 中寻找图片是否有对应的缓存, 它会以url 作为数据的索引先在内存中寻找是否有对应的缓存
2. 如果缓存未找到就会利用通过MD5处理过的key来继续在磁盘中查询对应的数据, 如果找到了, 就会把磁盘中的数据加载到内存中，并将图片显示出来
3. 如果在内存和磁盘缓存中都没有找到，就会向远程服务器发送请求，开始下载图片
4. 下载后的图片会加入缓存中，并写入磁盘中
5. 整个获取图片的过程都是在子线程中执行，获取到图片后回到主线程将图片显示出来

控制器的生命周期

就是问的view的生命周期，下面已经按方法执行顺序进行了排序

// 自定义控制器view，这个方法只有实现了才会执行

- (void)loadView
  {
  self.view = [[UIView alloc] init];
  self.view.backgroundColor = UIColor orangeColor;
  }
  // view是懒加载，只要view加载完毕就调用这个方法
  - (void)viewDidLoad

    {

    super viewDidLoad;

    NSLog(@"%s",**func**);

    }

    // view即将显示
  - (void)viewWillAppear:(BOOL)animated

    {

    super viewWillAppear:animated;

    NSLog(@"%s",**func**);

    }

    // view即将开始布局子控件
  - (void)viewWillLayoutSubviews

{

super viewWillLayoutSubviews;

NSLog(@"%s",**func**);

}

// view已经完成子控件的布局

- (void)viewDidLayoutSubviews

{

super viewDidLayoutSubviews;

NSLog(@"%s",**func**);

}

// view已经出现

- (void)viewDidAppear:(BOOL)animated

{

super viewDidAppear:animated;

NSLog(@"%s",**func**);

}

// view即将消失

- (void)viewWillDisappear:(BOOL)animated

{

super viewWillDisappear:animated;

NSLog(@"%s",**func**);

}

// view已经消失

- (void)viewDidDisappear:(BOOL)animated

{

super viewDidDisappear:animated;

```
    NSLog(@"%s",func);

}

// 收到内存警告
- (void)didReceiveMemoryWarning

{

    super didReceiveMemoryWarning;

    NSLog(@"%s",func);

}

// 方法已过期，即将销毁view
- (void)viewWillUnload

{

}

// 方法已过期，已经销毁view
- (void)viewDidUnload

{

}
```

你是怎么封装一个view的

- 可以通过纯代码或者xib的方式来封装子控件
- 建立一个跟view相关的模型，然后将模型数据传给view，通过模型上的数据给view的子控件赋值

```
/** * 纯代码初始化控件时一定会走这个方法
 */
```

- ```
  (instancetype)initWithFrame:(CGRect)frame
  {
      if(self = super initWithFrame:frame)
      {
          self setup;
      }
      return self;
  }
  /** * 通过xib初始化控件时一定会走这个方法
  ```
  ```

```
*/
- (id)initWithCoder:(NSCoder *)aDecoder
{
    if(self = super initWithCoder:aDecoder)
    {
        self setup;
    }
    return self;
}
- (void)setup
{
    // 初始化代码
}
```

如何进行iOS6、7的适配

通过判断版本来控制，来执行响应的代码

- 功能适配：保证同一个功能在6、7上都能用
- UI适配：保证各自的显示风格

// iOS版本为7.0以上（包含7.0） define iOS7 ([UIDevice currentDevice.systemVersion doubleValue]>=7.0)

如何渲染UILabel的文字

通过NSAttributedString/NSMutableAttributedString（富文本）

UIScrollView的contentSize能否在viewDidLoad中设置

能

- 因为UIScrollView的内容尺寸是根据其内部的内容来决定的，所以是可以在viewDidLoad中设置的
- 补充：（这仅仅是一种特殊情况）
- 前提，控制器B是控制器A的一个子控制器，且控制器B的内容只在控制器A的view的部分区域中显示
- 假设控制器B的view中有一个UIScrollView这样一个子控件
- 如果此时在控制器B的viewDidLoad中设置UIScrollView的contentSize的话会导致不准确的问题
- 因为任何控制器的view在viewDidLoad的时候的尺寸都是不准确的，如果有子控件的尺寸依赖父

控件的尺寸，在这个方法中设置会导致子控件的frame不准确，所以这时应该在下面的方法中设置子控件的尺寸

-(void)viewDidLayoutSubviews;

触摸事件的传递

- 触摸事件的传递是从父控件传递到子控件
- 如果父控件不能接收触摸事件，那么子控件就不可能接收到触摸事件
- 不能接受触摸事件的四种情况

1. 不接收用户交互，即：userInteractionEnabled = NO
2. 隐藏，即：hidden = YES
3. 透明，即：alpha <= 0.01
4. 未启用，即：enabled = NO

事件响应者链

1. 如果当前view是控制器的view，那么就传递给控制器
2. 如果控制器不存在，则将其传递给它的父控件
3. 在视图层次结构的最顶层视图也不能处理接收到的事件或消息，则将事件或消息传递给UIWindow对象进行处理
4. 如果UIWindow对象也不处理，则将事件或消息传递给UIApplication对象
5. 如果UIApplication也不能处理该事件或消息，则将其丢弃

补充：如何判断上一个响应者

- 如果当前这个view是控制器的view，那么控制器就是上一个响应者
- 如果当前这个view不是控制器的view，那么父控件就是上一个响应者

如何实现类似QQ的三角形头像

- Quartz2D
- 使用coreGraphics裁剪出一个三角形