1.VC的生命周期   
init：初始化程序   
loadView:视图初始化   
viewDidLoad:视图加载完成   
viewWillAppear:视图即将显示   
viewDidAppear:视图显示时调用   
viewWillDisappear:视图即将消失   
viewDidDisappear:视图消失时调用

2.assign 和weak的区别   
weak所有权修饰符表明该属性定义了一种非拥有关系，为这种属性设置新值时，既不保留新值，也不释放旧值。weak修饰的属性所指向的对象销毁以后，属性会自动置为nil，assign的特质与weak类似，但是assign修饰的属性所指向的对象销毁以后，不会自动置为nil。   
assign可以修饰非OC对象，weak只能用来修饰OC对象。

3.冒泡排序的时间复杂度   
冒泡排序：

NSMutableArray \*arrs = [NSMutableArray arrayWithObjects:@"17",@"23",@"11",@"47", nil];

for (int i = 0; i < arrs.count-1; i++) {

for (int j = 0; j < arrs.count-1-i; j++) {

if ([arrs[j] intValue] > [arrs[j+1] intValue]) {

int tmp = [arrs[j] intValue];

arrs[j] = arrs[j+1];

arrs[j+1] = [NSString stringWithFormat:@"%d",tmp];

}

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10

冒泡排序时间复杂度：O(n^2) —O(n的2次方)   
冒泡排序空间复杂度：O(1)

时间复杂度：执行这个算法需要消耗的时间   
①用常数1代替算法中的所有加法常数   
sum = n\*(n+1)/2;   
该算法的时间复杂度为O(1)   
②修改后的运行次数中只保留最高阶项

for (int i = 1; i < n; i++) {

x += i;

}

for (int i = 1; i < n; i++) {

for (int j = 1; j < n; j++) {

x += i+j;

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9

第一个for循环的时间复杂度为O(n)，第二个for循环的时间复杂度为O(n^2)，整个算法的时间复杂度为O(n)+O(n^2)=O(n^2)。

③去除最高阶项的系数

for (int i = 1; i < 3n; i++) {

for (int j = 1; j < n; j++) {

x += i+j;

}

}

* 1
* 2
* 3
* 4
* 5

O(3n\*n)=O(n^2)

④指数阶

int i = 1,n = 100;

while (i<n) {

i = i\*2;

}

* 1
* 2
* 3
* 4

该算法的执行次数为x, 2^x = n，x = log2n，时间复杂度为O(log2n)

空间复杂度：执行算法需要占用的内存空间   
当一个算法的空间复杂度为一个常量，即不随被处理数据量n的大小而改变时，可表示为O(1)；当一个算法的空间复杂度与以2为底的n的对数成正比时，可表示为O(10g2n)；当一个算法的空I司复杂度与n成线性比例关系时，可表示为O(n)

4.static和define的区别   
define定义的常量没有类型，static定义的常量有类型，在定义常量的时候，尽量使用static+const的方式

5.视频用的基础框架   
本地和网络流媒体音频视频播放：AVFoundation、MediaPlayer   
VLC框架本地和网络流媒体音频视频播放：MobileVLCKit、

6.afn底层原理   
AFN是对苹果的NSURLSession网络框架进行的封装，AFN分为六个部分：   
①通过AFSecurityPolicy实现安全策略，AFSecurityPolicy提供了三个安全策略，分别为无校验、公钥校验、证书校验

②AFNetworkReachabilityManager提供了四种枚举类型，实现对当前网络状态的判断，分别为未知网络、断网、流量、wifi，当网络状态发生变化时，通过block进行回调

③通过AFURLRequestSerialization将数据发送给服务器，AFURLRequestSerialization中我们经常用到的有三个部分：AFURLRequestSerialization协议返回一个序列化请求；AFHTTPRequestSerializer初始化请求类、对序列化参数编码、创建请求缓存策略、进行超时设置等；AFJSONRequestSerializer用来处理json类型请求

④通过AFURLResponseSerialization实现对服务器响应的处理，AFURLResponseSerialization中我们经常用到的有五个部分：AFURLResponseerialization协议返回一个序列化响应；AFHTTPResponseSerializer初始化响应类、设置可接收的Content-Type类型；AFJSONResponseSerializer校验并解码json类型的响应；AFXMLParserResponseSerializer校验并解码xml类型的响应；AFImageResponseSerializer校验并解码UIImage类型的响应。

⑤通过AFURLSessionManager创建不同类型的网络请求会话、设置网络请求的回响

⑥AFHTTPSessionManager根据不同需求，创建所需的GET、HEAD、POST、PUT、PATCH、DELETE方法

7.sd缓存原理   
（今天公司组织去看厉害了我的国纪录片，明天再发，好久不看了，还得现看一下）

8.简单说一下链表   
链表中的元素在内存中不是连续放置的，链表有不同的类型，分为单向链表和双向链表。单向链表中一个节点只有链向下一个节点的链接，而在双向链表中，链接是双向的，一个链向下一个元素，另一个链向前一个元素。

9.KVC、KVO原理   
KVO的原理：当观察对象B时，KVO会通过OC的runtime动态创建一个对象B当前类的子类，并为这个新的子类重写被观察属性的setter方法，setter方法随后负责通知观察对象属性的改变状况。KVO是通过setter方法实现的，需要使用self.来修改属性对象的成员变量才会有效，直接通过成员变量赋值不会触发KVO机制

KVC的原理：当一个对象调用setValue方法时，方法内部会做以下操作   
①检查是否存在相应的key的set方法，如果存在，就调用set方法   
②如果set方法不存在，就会查找与key相同并且带下划线的成员变量，如果有，则直接给成员变量属性赋值   
③如果没有找到\_key，就会查找相同名称的属性key，如果有就直接赋值   
④如果还没有找到，则调用valueForUndefinedKey和setValue:forUndefinedKey方法

10.手写单例

static XMSecureManager \*shareManager;

static dispatch\_once\_t once;

dispatch\_once(&once,^{

shareManager = [[self alloc] init];

});

return shareManager;

1，NSObject中description属性的意义，它可以重写吗?   
答案：   
每当 NSLog(@”“)函数中出现 %@ 时，都会调用description函数，所以通过重写 description 函数可以很好输出的制定特殊的格式。   
打印自定义对象会输出对象的名称和地址，这时候自定义description方法   
+description方法决定了类对象的输出结果，即类本身   
-description方法决定了实例对象的输出结果，即类创建的对象

2，写一个Objective-C实现单例模式的简单例子（注意多线程访问的问题)   
简述@synchronized(self){}的作用   
答案：   
单例设计模式：节约内存开销，保证整个应用程序只使用一份资源（大家访问的数据都是一致的），工具类设计成单例模式   
思路：

声明一个单件对象的静态实例,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时生成一个该类的实例

实现NScopying协议, 覆盖allocWithZone:方法,确保用户在直接分配和初始化对象时,不会产生另一个对象。

覆盖release、autorelease、retain、retainCount方法, 以此确保单例的状态。

在多线程的环境中,注意使用@synchronized关键字或GCD,确保静态实例被正确的创建和初始化（防止多线程抢夺资源）

* 1
* 2
* 3
* 4
* 5

代码：   
// 用来保存唯一的单例对象   
static id \_instace;

(id)allocWithZone:(struct \_NSZone \*)zone

{

static dispatch\_once\_t onceToken;

dispatch\_once(&onceToken, ^{

\_instace = [super allocWithZone:zone];

});

return \_instace;

}

(instancetype)sharedDataTool

{

static dispatch\_once\_t onceToken;

dispatch\_once(&onceToken, ^{

\_instace = [[self alloc] init];

});

return \_instace;

}

(id)copyWithZone:(NSZone \*)zone

{

return \_instace;

}

(oneway void)release { }

(id)retain { return self; }

(NSUInteger)retainCount { return 1;}

(id)autorelease { return self;}

为了保证线程的安全（避免多线程抢夺同一份资源），用@synchronized来加锁，或者用GCD

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32

3，为NSString扩展一个方法，方法能判断字符串是否是Url地址（即判断字符串是否以“http://”），返回BOOL值类型   
答案：

(BOOL)isUrl:(NSString \*)string {

if([string hasPrefix:@"http://"]){

return YES;

}else{

return NO;

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11

4，protocol的定义，实例   
答案：   
@protocol HKViewControllerDelegate

@optional

@required

@end

@interface HKViewController : UIViewController

@property (nonatomic, weak) id delegate;

@end   
实例：最常用的UITableView

5，定义一个Block并写一个应用例子，如果对变量加上\_\_block的表示，有什么意义。   
答案：   
block用来保存一段代码。是指向结构体的指针，编译器会把block存放的代码生成对应的函数，通过指针能访问函数。   
定义：

\_\_block：防止循环retain，用在非ARC   
block在ARC用strong，在MRC用copy   
block的使用场景：

1.把block保存到对象中去，等到恰当的时机再使用（外部调用），可以用来传值。

2.把block当成方法的参数使用，外部不调用，外部只是实现，函数内部调用

3.block作为返回值

* 1
* 2
* 3

链式编程思想：把要操作的值当成block的参数，block的返回值是方法调用者本身

6，这段代码有问题吗？   
for (int i = 0; i < someLargeNumber; i++)   
{   
NSString string =@”Abc”;   
string = [string lowercaseString];   
string = [string stringByAppendingString:@”xyz”];   
NSLog(@”%@”, string);   
}   
答案：   
问题出在每执行一次循环，就会有一个string加到当前NSRunloop中的自动释放池中，只有当自动释放池被release的时候，自动释放池中的标示了autorelease的这些数据所占用的内存空间才能被释放掉。假设，当someLargeNumber大到一定程度时，内存空间将被耗尽而没有被释放掉，所以就出现了内存溢出的现象。   
目前的解决方法就是在循环里面加个自动释放池   
for (int i = 0; i < someLargeNumber; i++)   
{   
NSAutoreleasePool pool = [[NSAutoreleasePool alloc] init];   
NSString \*string =@”Abc”;   
string = [string lowercaseString];   
string = [string stringByAppendingString:@”xyz”];   
NSLog(@”%@”, string);   
[pool release];   
}

7，self.name = “object”和name = “object”有何不同？   
答案：   
区别是前者会调用setName方法，后者只是赋值。

(void)setName:(NSString\*)newName

{

[newName retain];

[name relase];

name=newName;

}

* 1
* 2
* 3
* 4
* 5
* 6

8，category和inheritance的区别？   
答案：   
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。   
category的好处(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。   
inheritance可以增加，修改或者删除方法，并且可以增加属性。

9，代理的作用？   
答案：   
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。

10，消息推送（本地推送，远程推送）？   
答案：   
远程通知的几种展现形式：横幅，提醒，锁屏，app数字图标（在设置里面设置，还可以播放音效）。安卓不能开关（垃圾！）   
程序在前台，推送给不呈现；点击通知，打开app   
长连接的作用：时间校准，系统升级，传输速度快   
deviceToken = 手机的UDID + APP的BundleID   
远程推送（APNs）：设备联网之后把UDID和app的BundleID发给苹果的服务器，苹果的服务器加密之后返回给一个deviceToken。设备把deviceToken发给应用程序的服务器，服务器把用户的信息和deviceToken存储到数据库。别的用户给之前的用户发送消息的时候，先在数据库中查找用户的deviceToken，然后把deviceToken和消息体发送到苹果的服务器，然后推动给用户。

11，多态，动态类型和动态绑定。   
答案：   
不同对象以自己的方式响应相同的消息的能力叫做多态。   
向下转型之前一定要先做向上转型，正确的做法是：   
Person p = new Man(); //先将Man的实例向上转型成Person的实例（向上转型）   
Man man = (Man)p; //将被转型来的p实例进行强制转换成Man的实例man（向下转型）

动态绑定   
在objective-c中,   
一个对象是否调用指定的方法不是由编译器决定而是由运行时决定,这被称作是方法的动态绑定。   
在objective-c里,对象不调用方法,而是接收消息,消息表达式为:[reciver message];运行时系统首先确定接收者的类型(动态类型识别),然后根据消息名在类的方法列表里选择相依的方法执行,所以在源代码里消息也称为选择器(selector)   
消息函数的作用:   
– 首先通过第一个参数的receiver,找到它的isa指针,然后在isa指向的Class对象中使用第二个参数selector查找方法;   
– 如果没有找到,就使用当前Class 对象中的新的isa 指针 到上一级的父类的Class 对象中查找;   
– 当找到方法后,再依据receiver的中的self 指针找到当前 的对象,调用当前对象的具体实现的方法(IMP),然后传递参数,调用实现方法。   
– 假如一直找到NSObject的Class 对象,也没有找到你调用的方法,就会报告不能识别发送消息的错误。

动态加载：运行时加载新类   
在运行时创建一个新类,只需要3步:   
1、为 class pair分配存储空间 ,使用 objc\_allocateClassPair函数   
2、增加需要的方法使用class\_addMethod函数,增加实 例变量用class\_addIvar   
3 、用objc\_registerClassPair函数注册这个类,以便它能被别人使用。

12，事件传递链和响应者链   
事件传递链：   
用户的触摸事件首先会由系统截获，进行包装处理等。   
然后递归遍历所有的view，先看子控件能不能接收事件，在看点在不在子控件上面。   
然后进行碰触测试(hitTest)，直到找到可以处理事件的view。

- (UIView \*)hitTest:(CGPoint)point withEvent:(UIEvent \*)event; // recursively calls -pointInside:withEvent:. point is in the receiver's coordinate system

- (BOOL)pointInside:(CGPoint)point withEvent:(UIEvent \*)event; // default returns YES if point is in bounds

大致的过程application –> window –> root view –>……–>lowest view

* 1
* 2
* 3

响应者链：   
1，事件响应者链是有多个响应者对象组成的链条（响应者对象是能处理事件的对象）   
2，利用响应者链条，可以让多个响应者对象处理同一个事件，   
3，怎么利用链条向上传递，要寻找上一个响应者

如果当前的view是控制器的view，控制器就是上一个响应者   
如果当前的view不是控制器的view，这个view的父view就是上一个响应者   
当有view能够处理触摸事件后，开始响应事件。   
系统会调用view的以下方法：   
- (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;   
可以多对象共同响应事件。只需要在以上方法重载中调用super的方法。

大致的过程initial view –> super view –> …..–> view controller –> window –> Application

需要特别注意的一点是，传递链中时没有controller的，因为controller本身不具有大小的概念。但是响应链中是有controller的，因为controller继承自UIResponder。

* 1
* 2
* 3

13，方法和选择器有何不同？   
答案：   
selector是一个方法的名字，method是一个组合体，包含了名字和实现。

14，懒加载（延时加载）   
答案：   
（1）不必将创建对象的代码全部写在viewDidLoad方法中，代码的可读性更强   
（2）每个控件的getter方法中分别负责各自的实例化处理，代码彼此之间的独立性强，松耦合   
（3）只有当真正需要资源时，再去加载，节省了内存资源。   
（4）在内存警告的时候将懒加载的对象置为nil，当再次用到的时候又可以懒加载了

15，Cocoa中有虚基类的概念么？怎么简洁的实现？   
答案：   
虚基类主要解决在多重继承时，基类可能被多次继承，虚基类主要提供一个基类给派生类。   
例如：A是基类，B和C都继承了A类，而此时D又继承了B和C，那么D岂不是继承了两次A吗？为了避免重复多次继承，就应该声明B和C虚拟继承A，确保A被继承一次。

16，timer的间隔周期准吗？为什么？怎样实现一个精准的timer?   
答案：   
计时器不准确的原因是，计时器只有在 runLoop 的一次循环中被检查，所以如果在上次循环中做了什么耗时的操作，那么计时器就被延后执行了。   
正确的方法应该是新开一个线程，然后在新开的线程里设定一个 timer，并执行。   
\_\_block TestViewController \*blockSelf = self;   
dispatch\_async(dispatch\_get\_global\_queue(DISPATCH\_QUEUE\_PRIORITY\_DEFAULT, 0), ^{   
blockSelf->\_timer=[NSTimer scheduledTimerWithTimeInterval:1.0   
target:blockSelf   
selector:@selector(caculateLeftTimeForTomorrow)   
userInfo:nil   
repeats:YES] ;   
[[NSRunLoop currentRunLoop] addTimer:blockSelf->\_timer forMode:NSDefaultRunLoopMode];   
[[NSRunLoop currentRunLoop] run];   
});

17，id和instancetype   
答案：   
id可以指向任何类型的指针 或 指向任何未知类型的指针。   
instancetype只能作为返回值，不能像id那样作为参数

18，[pool release]和[pool drain]有什么区别？   
答案：   
在GC机制下，release相当于空操作，所以用[pool drain]。

19，数据持久化   
答案：   
数据持久化，实际上就是将数据存放到网络或者硬盘上，这里是存储到本地的硬盘上，应用程序的本地硬盘是沙盒，沙盒实际上就是一个文件夹，它下面有4个文件夹。分别是Documents，Library，APP包和tmp文件夹。   
Documents里面主要是存储用户长期使用的文件，itunes会自动备份，保存在这里会被苹果拒绝应用   
Library里面又有Caches和Preferences文件夹，   
Caches里面存放的是临时的文件，缓存。程序员管理。   
Preferences里面存放的是偏好设置。iTunes会备份，系统管理   
tmp里面也是临时的文件，系统管理。iTunes不备份

plist：能不能用plist存储看看能不能使用writeToFile方法,不能存储自定义对象   
偏好设置：不用关心文件名，键值对存储。单例。iOS7之前默认不会和硬盘同步   
NSKeyedArchiver：自定义对象归档，要遵守NSCoding协议，实现协议方法

encodeWithCoder告诉什么对象需要归档

initWithCoder。解档，解析文件的时候调用

调用父类的[super init]，只要父类也遵守了<NSCoding>协议，就要调用[super initWithCoder]方法;解档之后要给成员变量赋值！！

* 1
* 2
* 3

补充：调用init方法就会调用initWithFrame方法

20，在iphone上有两件事情要做,请问是在一个线程里按顺序做效率高还是两个线程里做效率高？为什么？   
答案：   
任务复杂就在一个线程顺序执行，因为要是在两个线程并发，线程切换需要时间   
线程的创建需要512k的栈内存空间，主线程需要1M

21，obj-c的优缺点   
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    
缺点:   1) 不支援命名空間   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。   
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。

22,UIView和CALayer   
答案：   
1.首先UIView可以响应用户的触摸事件，Layer不可以.   
2.View中frame getter方法，bounds和center，UIView并没有做什么工作；它只是简单的各自调用它底层的CALayer的frame，bounds和position方法。   
3.UIView主要是对显示内容的管理而 CALayer 主要侧重显示内容的绘制。   
4.在做 iOS 动画的时候，修改非 RootLayer的属性（譬如位置、背景色等）会默认产生隐式动画，而修改UIView则不会。   
每个 UIView 内部都有一个 CALayer 在背后提供内容的绘制和显示，并且 UIView 的尺寸样式都由内部的 Layer 所提供。两者都有树状层级结构，layer 内部有 SubLayers，View 内部有 SubViews.但是 Layer 比 View 多了个AnchorPoint   
在 View显示的时候，UIView 做为 Layer 的 CALayerDelegate,View 的显示内容由内部的 CALayer 的 display   
5.CALayer是定义在CoreGraphic框架中的，设置颜色和图像属性的时候要转换成CGColor和CGImage属性   
6.如果两个UIView是父子关系，那么它们内部的CALayer也是父子关系。   
7.两者最明显的区别是 View可以接受并处理事件，而 Layer 不可以；图层不能直接渲染到屏幕上。

23，写框架的时候注意点。   
答案：   
1，留的接口够不够用，调用是不是简单   
2，留的参数是不是够用   
3，是否能根据类名，方法名猜出功能   
4，是否依赖别人的框架

24，什么是NSManagedObject模型？什么是NSManagedobjectContext？   
答案：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。   
答案：NSManagedobjectContext对象负责应用和数据库之间的交互。

1> CoreData是对SQLite数据库的封装   
2> CoreData中的NSManagedObjectContext在多线程中不安全   
3> 如果想要多线程访问CoreData的话，最好的方法是一个线程一个NSManagedObjectContext   
4> 每个NSManagedObjectContext对象实例都可以使用同一个NSPersistentStoreCoordinator实例，这是因为NSManagedObjectContext会在便用NSPersistentStoreCoordinator前上锁

25，简单介绍下NSURLConnection类及+ sendSynchronousRequest:returningResponse:error:与– initWithRequest:delegate:两个方法的区别？   
答案: 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。

26，Objective C中的selector 是什么？   
答案：可以理解@selector（）就是取方法的编号,他的行为基本可以等同C语言的中函数指针,只不过C语言中，可以把函数名直接赋给一个函数指针，而 Objective-C的类不能直接应用函数指针，这样只能做一个@selector语法来取.它的结果是一个SEL类型。这个类型本质是类方法的编号 (函数地址)。

27，SQLite数据库   
答案：   
iOS中数据存储的方式：   
1>Plist（只能数组，字典）   
2>Preference(偏好设置/NSUserDefault)   
3>NSCoding（NSKeyedArchiver）   
4>SQLite3   
5>Core Data

SQL：结构化查询语言   
数据库不分大小写，语句以“;”结尾。   
DDL：数据定义语句create table/drop table   
DML：数据操作语言insert update delete   
DQL：数据查询语言select

SQLite数据库无类型   
数据库中的字符串用单引号   
count(\*)代表记录   
ORDER BY + 字段 DESC/ASC

limit可控制查询条数   
约束：NOT NULL UNIQUE，可以一起用   
主键约束：自增长，唯一，不为空

利用外键约束可以建立表和表之间的联系：一张表的某个字段，引用领一张表的主键   
表链接：需要联合多张表才能查到想要查询的数据

28，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。

GCD和block：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。

GCD:异步并发：开启多个线程异步执行   
异步串行：开一一个线程顺序执行   
异步主线程：不开启线程串行执行

同步任务在主队列会卡死主线程（因为同步任务要立即执行，但是还要等待当前函数执行完毕，所以互相等待，卡死）   
异步任务在主队列不会卡死，会让当前函数执行之后再执行任务

dispatch\_apply用于遍历数组，无序执行，可以用这个函数，快速迭代

NSOperationQueue进行挂起或者取消不会立刻终结当前任务   
自定义NSOperation要自定义main方法，加入main方法中有耗时操作，建议在每个耗时操作之间加入方法判断是不是被取消了

29，谓词(NSPredicate)   
答案：OC中的谓词操作是针对于数组类型的，他就好比数据库中的查询操作，数据源就是数组，这样的好处是我们不需要编写很多代码就可以去操作数组，同时也起到过滤的作用，我们可以编写简单的谓词语句，就可以从数组中过滤出我们想要的数据。

30，UI框架的底层有CoreAnimation，CoreAnimation的底层有CoreGraphics。

31，目标 - 动作(Target - Action)   
答案：允许一个控件对象(例如按键或滑动条) 向另外一个对象发送一条消息（即动作），以之作为对某个用户事件（例如一个点击事件）的响应。接收到消息的对象则可以对消息进行响应，并针对业务要求做出处理。

32，优化UITableView的性能（比如含有正在加载的网络图片不会卡）   
答案：

使用不透明视图。

不要重复创建不必要的table cell（注意设置identifier回收）

不要把控件放到contentView中，最佳的解决办法还是继承UITableViewCell，并在其drawRect:中自行绘制

在实现drawRect:的时候，它的rect参数就是需要绘制的区域，这个区域之外的不需要进行绘制。

避免在主线程更新UI，加载网络图片用SDWebImage

* 1
* 2
* 3
* 4
* 5

33，NSInteger和int   
答案：NSInteger会根据系统是32位还是64位来决定是本身是int还是Long。

34，沙盒的目录结构   
答案：   
Documents：保存应用运行时生成的需要持久化的数据，iTunes会自动备份该目录。苹果建议将程序中建立的或在程序中浏览到的文件数据保存在该目录下，iTunes备份和恢复的时候会包括此目录

Library：存储程序的默认设置和其他状态信息，iTunes会自动备份该目录。   
Libaray/Caches:存放缓存文件，iTunes不会备份此目录，此目录下文件不会在应用退出删除。一般存放体积比较大，不是特别重要的资源   
Libaray/Preferences:保存应用的所有偏好设置，ios的Settings（设置）应用会在该目录中查找应用的设置信息，iTunes会自动备份该目录。

tmp：保存应用运行时所需的临时数据，使用完毕后再将相应的文件从该目录删除。应用没有运行时，系统也有可能会清除该目录下的文件，iTunes不会同步该目录。iphone重启时，该目录下的文件会丢失。

35，+load和+initialize方法的区别   
答案：   
load是只要类所在文件被引用就会被调用，而initialize是在类或者其子类的第一个方法被调用前调用。所以如果类没有被引用进项目，就不会有load调用；但即使类文件被引用进来，但是没有使用，那么initialize也不会被调用。   
load方法在Main方法执行之前执行

36， Posing，顾名思义，意思是“冒充”，它跟categories类似，但本质上不一样，Posing存在的目的在于子类可以冒充父类，使得后续的代码无需把父类修改为子类，就可以很方便的让父类表现成子类的行为，从而实现非常方便的冒充。（不推荐使用的语言特征）

37，什么是runtime？   
1> runtime是一套底层的C语言API（包含很多强大实用的C语言数据类型、C语言函数）   
2> 实际上，平时我们编写的OC代码，底层都是基于runtime实现的   
也就是说，平时我们编写的OC代码，最终都是转成了底层的runtime代码（C语言代码）   
runtime有啥用？   
1、 能动态产生一个类、一个成员变量、一个方法   
2、 能动态修改一个类、一个成员变量、一个方法   
3、 能动态删除一个类、一个成员变量、一个方法

38,   
动态添加方法：   
可以用performSelector动态添加方法   
resolveInstanceMethod调用:当调用了没有实现的方法没有实现就会调用resolveInstanceMethod   
分类添加属性：   
让属性和对象产生关联：

(void)setName:(NSString )name {

// 添加属性,跟对象

// 给某个对象产生关联,添加属性

// object:给哪个对象添加属性

// key:属性名,根据key去获取关联的对象 ,void == id

// value:关联的值

// policy:策越

objc\_setAssociatedObject(self, @"name", name, OBJC\_ASSOCIATION\_RETAIN\_NONATOMIC);

}

解决KVC报错

(void)setValue:(id)value forUndefinedKey:(NSString \*)key

{

if ([key isEqualToString:@"id"]) {

\_ID = [value integerValue];

}

// key:没有找到key

// value:没有找到key对应的值

NSLog(@"%@ %@",key,value);

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21

39，静态库和动态库在使用上的区别   
静态库：链接时，静态库会被完整地复制到可执行文件中，被多次使用就有多份冗余拷贝   
动态库：链接时不复制，程序运行时由系统动态加载到内存，供程序调用，系统只加载一次，多个程序共用，节省内存   
注意：项目中如果使用了自制的动态库，不能被上传到AppStore

.a 和 .framework 的使用区别   
.a 本身是一个二进制文件，需要配上 .h 和 其它资源文件 才能使用   
.framework 本身已经包含了 .h 和 其它资源文件，可以直接使用

40，APP上架

41，UIButton和UITableView的设计模式

42，system样式的UIButton不能设置image属性，所以要把样式改成custom；属性名不能用new开头；initWith注意With的W大写

43，CocoaPods   
答案：CocoaPods基于ruby对gem（gem是一种文件组织的包，一般的ruby的很多插件都有由这种各种的包提供）的依赖管理

44，在OC中调用JavaScript代码   
使用UIWebView的stringByEvaluatingJavaScriptFromString方法即可

45，正则表达式   
拼接字符串规则   
创建正则表达式对象，将规则放到正则表达式对象中   
将传进来的字符串进行遍历，看看匹配哪些规则（表情，@，话题，URL）

46，图片加载的两种方法   
方式一：有缓存（图片所占用的内存会一直停留在程序中）

(UIImage )imageNamed:(NSString )name;

name是图片的文件名

* 1
* 2

方式二：无缓存（图片所占用的内存会在一些特定操作后被清除）   
NSString \*path = [[NSBundle mainBundle] pathForResource];

(UIImage )imageWithContentsOfFile:(NSString )path

(id)initWithContentsOfFile:(NSString \*)path;

path是图片的全路径

* 1
* 2
* 3

47,view的封装   
如果一个view内部的子控件比较多，一般会考虑自定义一个view，把它内部子控件的创建屏蔽起来，不让外界关心   
外界可以传入对应的模型数据给view，view拿到模型数据后给内部的子控件设置对应的数据

48，进程间的通信方式：   
无名管道( pipe )：管道是一种半双工的通信方式，数据只能单向流动，而且只能在具有亲缘关系的进程间使用。进程的亲缘关系通常是指父子进程关系。

高级管道(popen)：将另一个程序当做一个新的进程在当前程序进程中启动，则它算是当前程序的子进程，这种方式我们成为高级管道方式。

有名管道 (named pipe) ： 有名管道也是半双工的通信方式，但是它允许无亲缘关系进程间的通信。

消息队列( message queue ) ： 消息队列是由消息的链表，存放在内核中并由消息队列标识符标识。消息队列克服了信号传递信息少、管道只能承载无格式字节流以及缓冲区大小受限等缺点。

信号量( semophore ) ： 信号量是一个计数器，可以用来控制多个进程对共享资源的访问。它常作为一种锁机制，防止某进程正在访问共享资源时，其他进程也访问该资源。因此，主要作为进程间以及同一进程内不同线程之间的同步手段。

信号 ( sinal ) ： 信号是一种比较复杂的通信方式，用于通知接收进程某个事件已经发生。

共享内存( shared memory ) ：共享内存就是映射一段能被其他进程所访问的内存，这段共享内存由一个进程创建，但多个进程都可以访问。共享内存是最快的 IPC 方式，它是针对其他进程间通信方式运行效率低而专门设计的。它往往与其他通信机制，如信号两，配合使用，来实现进程间的同步和通信。

套接字( socket ) ： 套解口也是一种进程间通信机制，与其他通信机制不同的是，它可用于不同机器间的进程通信。

49，pch的作用   
首先定义一个宏看看是不是OC文件，防止编译错误   
优点：

1,放一些公用的宏

2,公用的头文件

3,自定义Log

缺点：

项目大时候编译的慢

* 1
* 2
* 3
* 4
* 5

50，用KVC设置模型属性

1，依次遍历字典，取出key：name，去模型中找setter方法赋值

2,找\_name方法进行赋值\_name = name;

3,找name属性直接进行赋值

全都找不到就会报错

* 1
* 2
* 3
* 4

51，UIApplication   
lication是应用程序的象征   
应用程序创建的第一个对象就是UIApplication对象   
applicationIconBadgeNumber红点点   
状态栏是不是在联网状态，打电话，打开网页。打电话发短信之所以要用application操作，是设计到应用级别的跳转，所以用应用的唯一标识。   
代理方法：启动–>获取焦点（可以用户交互）   
UIApplicationMain作用：   
创建UIApplication对象，创建UIApplication的代理，并且设置代理遵守UIApplicationDelegate协议，开启主运行循环，接收处理事件   
通过info.plist加载Main.storyBoard（没有不加载）

52，UIWindow   
直接设置窗口的根控制器会增加旋转功能，直接添加控制器的view不会做旋转功能   
makeKeyAndVisible：当前窗口成为应用程序的主窗口，并且显示   
状态栏和键盘都是窗口，弹框不是窗口。   
后创建的窗口放在最上层   
UIWindowLevelAlert > UIWindowLevelStatusBar > UIWindowLevelNormal 真实类型是CGFloat   
要想让控制器显示在最上面，就添加到UIWindow上面

53，-(void)loadView的作用   
1，自定义控制器的view   
2，第一次使用控制器的view时候调用   
3，如果没有自定义view，就不能获取控制器的view   
4，不用调用[super loadView];   
5，重写了这个方法，就不会加载storyboard描述的view了

注意：加载控制器的view，先调loadView方法，然后是viewDidLoad方法

54,默认控制器的颜色：几乎透明，clearColor，后面的view不能交互，但是能看到   
<= 0.01透明 >0.01几乎透明

55，pop控制器的时候要把动画执行完毕才销毁，一个导航控制器只有一个导航条，导航条的标题由栈顶控制器决定。什么什么item都是苹果封装的模型   
[button sizeToFit];控件的尺寸有图片决定（仅仅设置尺寸，不能设置位置）

56，控制器的生命周期   
viewDidLoad>>viewWillAppear>>viewWillLayoutSubviews>>viewDidLayoutSubviews>>   
viewDidAppear   
跳转的时候先调用第二个控制器的viewDidLoad>>viewWillLayoutSubviews>>viewDidLayoutSubviews   
然后第一个控制器viewWillDisAppear>>viewDidDisAppear   
didReceiveMemeryWarning调用之后调用viewDidUnload

57,block（相当于小弟）快捷方式：inline   
保存一段代码，用到的时候再去调用，一般不需要返回值,需要传递什么值，就把什么作为参数   
适用场合：逆传传值，处理网络的时候用block封装代码：先把更新UI的代码放在block中，等到网络加载完成后调用

A->B B要传值给A，在B中定义block属性，适当的位置调用，block的实现放在A中

58，UITabBarButton没有图片，badge显示在tabBarButton的左上角，有图片才显示在图片的右上角

59，tableView数据固定死就用静态单元格，静态单元格只能在storyBoard中用，Xib不能用。使用静态单元格的时候不用写tableView的数据源方法

60，任何控制器都能用modal展示出来,push的下一个界面和上一个界面有联系，modal没有联系   
modal出什么控制器，什么控制器就有权利dismiss

61，调用一个控制器的alloc init方法，会先加载VC的view的view.xib。没有则会加载viewController.xib。也没有则不会加载xib

62，只有继承了UIResponder的对象才能接受并处理事件，view的拖拽要设置multiple touch   
触摸事件发生后，事件交给UIApplication处理，application取出最上层的事件交给keyWindow，主窗口寻找最合适的视图去处理，父控件的alpha值小于0.01，会让子控件看不见

hitTest方法：事件传递的时候调用，为了寻找最合适的view，传入的参数point是方法调用者的坐标系   
子控件超出父控件之外不能接受事件。解决办法：转换坐标系到超出的子控件，判断点击的点是不是在超出的子控件上面，如果是，交给子控件处理，不在，调用super的hitTest方法

63，runloop（事件驱动）   
作用：保持程序一直运行，处理触摸事件（输入源/定时源），定时事件，selector事件，节省CPU资源，不用的时候休眠   
NSRunloop CFRunLoopRef。每条线程都有与之对应的RunLoop，让子线程永远活着就可以开启RunLoop。用字典来存储，key是当前线程   
一个RunLoop包含多个模式，一个模式包含Source/Timer/Observer,不同的模式对应不同的事件   
runloop要想成功运行有两个条件：模式里面有东西，调用run方法   
Observer监听Runloop的状态改变,用处：在UI点击之前做一些事情。点击按钮先调afterWaiting   
runLoop有时间限制，但是系统给的时间很大   
CF的内存管理：   
凡是带有create，copy，retain创建的对象都要进行一次release操作：CFRelease()

通过调用[NSRunloop currentRunLoop];就可以创建runloop对象   
一条线程对应一个runloop：源码写道：调用[NSRunloop currentRunLoop]就传给一个pthread对象，然后返回一个runloop对象，通过字典的形式一一对应   
和runloop相关的5个类：CFRunloopRef，CFRunloopModeRef，CFRunloopSourceRef，CFRunloopTimerRef，CFRunloopObserverRef

source0：基于非port的（比如按钮点击）   
source1：基于port的，操作系统内核或者其他线程发过来的消息，source1接收到事件是发给source0的

应用   
1，滚动操作的时候延迟显示图片   
2，常驻线程，用来监控用户的行为,联网状态   
3，自动释放池在runloop休眠之前（before waiting）会清理一次，创建常驻线程开启runloop的代码最好用@autoreleasepool包住   
kCFRunLoopEntry; // 创建一个自动释放池   
kCFRunLoopBeforeWaiting; // 销毁自动释放池，创建一个新的自动释放池   
kCFRunLoopExit; // 销毁自动释放池

面试题：   
1，什么是runloop：

是运行循环，内部是执行一个高级的do-while循环，这个循环中这行各种timer，source，observe任务，没有任务就会休眠，休眠之前会释放一下自动释放池

一个线程对应一个runloop，主线程的runloop默认开启，子线程的runloop要手动开启，开启runloop要设置模式，模式里面要有任务，没有任务就会直接退出runloop

* 1
* 2

64，NSTimer不准   
NSTimer有两种创建方式，schedule…自动加到主线程的默认模式下的runloop，模式改变（tracking模式）的时候不准。第二种[NSTimer timer…]加到主运行循环中，加上模式标签commonModes

65，父子控制器   
两个控制器的view是父子关系，两个控制器也应该是父子关系

66，static的作用

修饰局部变量会让局部变量只创建一次，内存只有一份（修改之后保存新的值），不会改变局部变量的作用域

用extern关键字可以访问全局变量，普通的全局变量不安全，会被改掉

用static修饰的全局变量作用域变成当前文件，不是全世界了

* 1
* 2
* 3

67，copy

调用multiCopy一定会产生可变字符串，产生新地址（深拷贝）

只有copy不可变对象（比如NSString）才会产生不可变字符串，不产生新地址（浅拷贝）

要想拷贝自定义对象，就要遵守<NSCopying>协议，实现copyWithZone方法

@property声明字符串用copy：将传进来的属性copy一份赋值给对象的属性，传进来的属性改变，对象的属性不改变

例子：@property(nonatomic, copy) NSMultiString \*name;

错误！真实类型是NSString。要用strong属性声明

* 1
* 2
* 3
* 4
* 5
* 6

68，const和宏

const修饰的变量是常量，定义的宏会在内存中出现多次，但是const保证全局只有一次内存（苹果的通知的常量用的都是const）

static const修饰的全局变量变成全局常量，并且只能在本文件访问

在项目中定义常量，定义Const.h和Const.m文件.m用来声明常量.h用来引用常量。为了装逼，可以把extern换成UIKIT\_EXTERN。

面试题：

//p1不能修改（不能修改指针指向的值），但是p1可以修改（可以修改p1的指向）

const int p1; == int const p1;

正确：p1 = &num; 错误：p1 = 30;

//p2不能修改（不能修改指针p2的指向），但是p2可以修改（可以修改p2指针指向的值）

int const p2;

错误：p2 = &num; 正确p2 = 30;

//不希望外部改name的值，所以用const修饰

UIKIT\_EXTERN NSString const name;

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12

数组指针面试题：   
数组的地址就是首元素的地址   
指针p的加减法运算   
指针p + N   
p里面存储的地址值 + N 指针所指向类型的占用字节数   
指针p - N   
p里面存储的地址值 - N 指针所指向类型的占用字节数   
数组名   
存储的是数组首元素的地址   
等价于:一个指向数组首元素的指针   
数组名 + 1 的跨度：数组首元素的占用字节数   
其他结论   
&num + 1的跨度：num的占用字节数   
举例：   
// int numbers[2][2][2] = {   
// {   
// {10, 20},   
// {30, 40},   
// },   
// {   
// {50, 60},   
// {70, 80}   
// }   
// };   
//   
// numbers[0][0] == &numbers[0][0][0],相当于是一个指向numbers[0][0][0]的指针   
// numbers[1] == &numbers[1][0],相当于是一个指向numbers[1][0]的指针   
// numbers == &numbers[0],相当于是一个指向numbers[0]的指针   
// &numbers == 相当于是一个指向numbers的指针

69,在Objective-C中，对象的类是isa指针决定的。isa指针指向对象所属的类。   
这就引出了元类的定义：元类是类对象的类，类对象是元类的实例。   
简单说就是：

我们以前调用 "+" 开头的类方法实际是在调用元类的对象方法

当你给对象发送消息时，消息是在寻找这个对象的类的方法列表。

当你给类发消息时，消息是在寻找这个类的元类的方法列表。

根元类的元类则就是它自己。也就是说基类的元类的isa指针指向他自己。

* 1
* 2
* 3
* 4

70，分析内存分配：Xcode->Product->Profile->Allocations/Leaks   
内存包括实际内存和虚拟内存   
/\*   
imageNamed和imageWithContentOfFile

imageNamed:加载图片   
1.当对象销毁,图片对象不会随着一起销毁   
2.加载的图片占据的内存较大:9.48   
3.相同的图片只会加载一份到内存中,如果同时使用,使用同一个对象即可

imageWithContentOfFile:加载图片   
1.当对象销毁的时候,图形对象会随着一起销毁   
2.加载的图片,占据的内存较小:6.25   
3.相同的图片会多次加载到内存中,如果同时使用图片,使用的是不同的对象

总结:   
imageName:如果一些图片在多个界面都会使用,并且图片较小,使用频率高.(图标/小的背景图)   
imageWithContentOfFile:只在一个地方使用,并且图片较大,使用频率不高.(版本新特性/相册)   
\*/

71，分析内存泄露Xcode->Product->Analyze   
1，静态内存分析：不运行程序，直接对代码进行分析。没有真正分配内存，根据代码的上下文情况分析是不是有内存泄露，不一定准确。为什么不准确？在方法中声明的变量在外面释放会被检测到内存泄露   
运用场景：在ARC用CF框架   
2，动态内存分析：让程序跑起来，用到可能泄露的代码

72，在iOS中，有2个框架可以访问用户的通讯录   
AddressBookUI.framework   
提供了联系人列表界面、联系人详情界面、添加联系人界面等   
一般用于选择联系人

AddressBook.framework   
纯C语言的API，仅仅是获得联系人数据   
没有提供UI界面展示，需要自己搭建联系人展示界面   
里面的数据类型大部分基于Core Foundation框架，使用起来极其蛋疼

从iOS6开始，需要得到用户的授权才能访问通讯录，因此在使用之前，需要检查用户是否已经授权

73,屏幕适配   
1>.为什么苹果推出SizeClasses   
iPhone3gs-4s : frame直接写死   
iPad : autoresizing—>根据父控件frame发生改变,子控件跟着一起改变   
iPhone5-iPhone5s : autolayout —>自动布局   
iPhone6和iPhone6p : size Classes—>发现屏幕变的太多样化,界面大统一

2> sizeclass

仅仅是对屏幕进行了分类, 真正排布UI元素还得使用autolayout

不再有横竖屏的概念, 只有屏幕尺寸的概念

不再有具体尺寸的概念, 只有抽象尺寸的概念

把宽度和高度各分为3种情况

1) Compact : 紧凑(小)

2) Any : 任意

3) Regular : 宽松(大)

和UIButton类比

练习

练习1:UIButton手机横屏的时候显示,竖屏的时候不显示

练习2:横屏的时候在左上角,竖屏的时候,在右下角

练习3:iPad当中也显示在中间

练习4:图片的显示 : 一般情况下显示一张图片,iPad显示特有的图片

练习5:Label的显示

练习6:约束冲突

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19

4) 符号代表

: Compact

: Any

: Regular

5) 继承性

w:Compact h:Compact 继承 (w:Any h:Compact , w:Compact h:Any , w:Any h:Any)

w:Regular h:Compact 继承 (w:Any h:Compact , w:Regular h:Any , w:Any h:Any)

w:Compact h:Regular 继承 (w:Any h:Regular , w:Compact h:Any , w:Any h:Any)

w:Regular h:Regular 继承 (w:Any h:Regular , w:Regular h:Any , w:Any h:Any)

6) 设备对应屏幕

iPhone4S,iPhone5/5s,iPhone6

竖屏：(w:Compact h:Regular)

横屏：(w:Compact h:Compact)

iPhone6 Plus

竖屏：(w:Compact h:Regular)

横屏：(w:Regular h:Compact)

iPad

竖屏：(w:Regular h:Regular)

横屏：(w:Regular h:Regular)

Apple Watch(猜测)

竖屏：(w:Compact h:Compact)

横屏：(w:Compact h:Compact)

7) sizeclass和autolayout的作用

sizeclass:仅仅是对屏幕进行了分类

autolayout:对屏幕中各种元素进行约束(位置\尺寸)

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24

74,静态库和动态库   
frameWork可以是动态的，也可以是静态的   
系统的frameWork是动态库，默认制作出来的frameWork都是动态库   
静态库：链接的时候静态库会被复制到可执行文件中，产生冗余   
动态库：系统只加载一次，多个程序可以共用。使用自制的动态库，不能上传到AppStore   
可以把MRC编译成静态库   
模拟器：   
4s-5用的都是i386架构   
5s-6Plus用的都是x86\_64架构   
真机：   
3GS-4s armv7架构   
5/5c armv7s架构,只要支持armv7，就支持armv7s   
5s-6Plus arm64架构   
debug版本包含很多符号信息

75，把NSURLConnection放到主线程默认会开启子线程处理。尝试放在子线程发送请求会不好用，因为子线程的runloop默认不启动

76，用NSURLSession实现断点下载，记录data文件，把上次下载的tmp文件保存到caches里面，开始的时候再放到tmp文件夹里面   
用NSURLConnection实现断点下载：设置http的请求头，设置Range的区间

77，如何快速上手公司的旧项目

搜索文字，搜索图片

把图片保存，关联控制器记录下来

搜索AppDelegate/int main

return;注释

* 1
* 2
* 3
* 4

78，   
当把一个对象添加到集合中的时候会把对象的引用计数+1   
当集合被销毁的时候，会对集合里面所有的对象引用计数—1   
对象从集合中移除之后，会把对象引用计数-1   
以add开头的方法，后面的对象的引用计数会+1

79设计模式

1 工厂模式

客户向工厂提出需求，不关注如何制造手机；工厂制造手机，对制造工艺进行封装

简单工厂是具体的类

工厂管理者将简单工厂的工厂类进行了一次抽象，根据不同的需求创建不同的具体工厂

直接对NSNumber/NSValue进行实例化不能创建出来对象，有实现具体工厂

* 1
* 2
* 3
* 4
* 5
* 6

80,线程的状态   
当一个线程刚刚创建的时候处于就绪状态，调用start方法的时候放到内存中的可调度线程池中，当CPU轮询到这个线程的时候就会处于运行状态，处理别的线程的时候处于就绪状态。当睡眠或者等待同步锁的时候就处于阻塞状态，这时候会从可调度线程池中移除，但是不会在内存中销毁，等到阻塞状态结束之后后就会重新放回可调度线程池中。等到线程死亡的时候就会从可调度线程池中移除，并且从内存中销毁

81，MD5和base64区别   
MD5是一种不可逆的摘要算法，无论多少二进制数据，在MD5算法一定的情况下，都会变成一个定长的数据，并且是根据内容不同而唯一。

而Base64是一种编码方式，主要用于将二进制数据转换为文本数据，方便使用HTTP协议等，是可逆的。

83，AFNetWorking的底层实现

基于NSURLSession的和NSURLConnection的封装，但是现在3.0版本已经放弃对NSURLConnection的支持了。

发送基本的HTTP请求用到的是AFHTTPSessionManager，调用+manager方法进行初始化，初始化包括指定sessionConfiguration为默认的configuration，NSOperationQueue的初始化并且设置最大并发数为1，指定session的代理为当前控制器。GET请求方法封装了dataTast和downloadTask，并且将task启动（因为通过原生的方式创建默认不启动，要手动启动，就是resume方法）。在文件上传的时候将文件的二进制文件拼接到参数中就行，内部帮我们设置好了mimeType，下载的时候内部遵守了协议，实现了三个代理方法，将下载的进度以block的方式返回回来，同时将完成结果返回回来

监控联网状态的类使用常驻线程来一直监控用户的联网状态

* 1
* 2
* 3

84，操作系统的五大管理功能:   
（1）作业管理：包括任务、界面管理、人机交互、图形界面、语音控制和虚拟现实等；   
（2）文件管理：又称为信息管理；   
（3）存储管理：实质是对存储“空间”的管理，主要指对主存的管理；   
（4）设备管理：实质是对硬件设备的管理，其中包括对输入输出设备的分配、启动、完成和回收；   
（5）进程管理：实质上是对处理机执行“时间”的管理，即如何将CPU真正合理地分配给每个任务。   
五大类型操作系统各自的特点分别是：   
（1） 批处理操作系统的特点有：a. 用户脱机使用计算机。用户提交作业之后直到获得结果之前就不再和计算机打交道。作业提交的方式可以是直接交给计算中心的管理操作员，也可以是通过远程通讯线路提交。提交的作业由系统外存收容成为后备作业。   
b.成批处理。操作员把用户提交的作业分批进行处理。每批中的作业将由操作系统或监督程序负责作业间自动调度执行。   
c.多道程序运行。按多道程序设计的调度原则，从一批后备作业中选取多道作业调入内存并组织它们运行，成为多道批处理。   
（2） 分时操作系统的特点有：a. 交互性：首先， 用户可以在程序动态运行情况下对其加以控制。其次，用户上机提交作业方便。第三，分时系统还为用户之间进行合作提供方便。   
b. 多用户同时性：多个用户同时在自己的终端上上机，共享CPU和其他资源，充分发挥系统的效率。   
c.独立性：客观效果上用户彼此间感觉不到有别人也在使用该台计算机，如同自己独占计算机一样。   
（3） 实时操作系统的特点有：a. 实时时钟管理(定时处理和延时处理)。   
b. 连续的人-机对话，这对实时控制往往是必须的。   
c.要求采取过载保护措施。例如对于短期过载，把输入任务按一定的策略在缓冲区排队，等待调度; 对于持续性过载，可能要拒绝某些任务的输入; 在实时控制系统中，则及时处理某些任务，放弃某些任务或降低对某些任务的服务频率。   
d.高度可靠性和安全性需采取冗余措施。双机系统前后台工作，包括必要的保密措施等。   
（4） 网络操作系统的特点有：a. 计算机网络是一个互连的计算机系统的群体。   
b. 这些计算机是自治的，每台计算机有自己的操作系统，各自独立工作，它们在网络协议控制下协同工作。   
c.系统互连要通过通信设施(硬件、软件)来实现。   
d.系统通过通信设施执行信息交换、资源共享、互操作和协作处理，实现多种应用要求。   
（5） 分布式操作系统的特点有：a.计算机网络的开发都遵循协议，而对于各种分布式系统并没有制定标准的协议。当然，计算机网络也可认为是一种分布式系统。   
b.分布式系统要求一个统一的操作系统，实现系统操作的统一性。   
c.分布式操作系统对用户是透明的。但对计算机网络，若一个计算机上的用户希望使用另一台计算机上的资源，则必须明确指明是哪台计算机。   
d.分布式系统的基础是网络。分布式系统已不仅是一个物理上的松散耦合系统，同时还是一个逻辑上紧密耦合的系统。   
e.分布式系统还处在研究阶段。而计算机网络已经在各个领域得到广泛的应用

85, 桥接转换

在MRC的情况下,直接进行强制类型转换就行,不用考虑对象的所有权

在ARC的情况下会报错，加上bridge桥接转换,但是不会转让对象的所有权。bridge\_transfer:CF对象的所有权交给OC对象，由ARC释放内存。bridge\_retained:OC对象的所有权交给CF对象管理。需要手动CFRelease()一下

唐巧： bridge只做类型转换，不增加引用计数，CF对象不用的时候需要调用CFRelease。

bridge\_retained:类型转换后，引用计数加1，需要调用CFRelease。 bridge\_transfer:类型转换后，将对象的引用计数交给ARC管理，不用调用CFRelease。

* 1
* 2
* 3
* 4

86,nil,Nil,NULL,NSNull   
Nil:空类   
nil:空对象,已经从内存中消失了，再次retain或者添加到数组/字典中会崩溃   
NSNull:对象的值为空，但是再次retain或者添加到数组/字典中不会导致程序崩溃   
NULL:来自C语言，代表空指针

87，iOS中导航设计模式的种类   
平铺导航：UITabBarController   
标签导航：UINavigationController   
树形导航：UIPageViewController

88，iOS单元测试框架   
OCUnit和XCTest都是官方的测试框架，OCUnit过时已经被XCTest取代。   
GHUnit和OCMock都是第三方的测试框架

89，模态视图的专用属性   
A.UIModalPresentationFullScreen,全屏状态，是默认呈现样式，iPhone只能全屏呈现。   
B.UIModalPresentationPageSheet，它的宽度是固定的768点，在iPad竖屏情况下则全屏显示。   
C.UIModalPresentationFormSheet，它的是固定的540\*620点，   
无论是横屏还是竖屏情况下呈现尺寸都不会变化。   
D.UIModalPresentationCurrentContext,它与父视图控制器有相同的呈现方式。

90，UIPopoverController控制器的常用方法和属性（ABCD）   
A. presentPopoverFromBarButtonItem：permittedArrowDirections:animated: 呈现Popover视图方法；   
B.dismissPopoverAnimated：关闭Popover视图方法；   
C.popoverVisible,判断Popover视图是否可见；   
D.popoverArrawDirection，判断Popover视图箭头的方向;

91，NSURL的构造函数有？（CD）   
A. + requestWithURL:   
B - initWithURL:   
C + URLWithString:   
D - initWithString:

92，MRR是MRC的官方名字；   
MRC是手动引用计数；   
ARC是自动引用计数；   
GC是垃圾回收（在开发OS X中使用）；

93，   
NSXML框架中核心的是NSXMLParser和它的委托协议NSXMLParserDelegate,NSXMLParserDelegate常用的方法有哪些？（ABCDE）

A. parserDidStartDocument

B. parser:foundCharacters

C. parser:didStartElement:namespaceURI:qualifiedName:attributes

D. parser:didEndElement:namespaceURI:qulificedName

E. parserDidEndDocument

95，软件的缺陷生命周期：   
1，new：缺陷被测试发现（卧槽！）   
2，Open：测试提交给开发（妈蛋！）   
3，Update：开发人员修复了但是没有给测试人员（小样！）   
4，Fix：自测通过，提交给测试人员（得意脸）   
5，Close：测试的确修复（牛逼！）   
6，Reopen：没通过，重新改（损色）

96，NSHashTable对引用的对象是weak指针

97，枚举定义1<<3   
什么时候要用到这种方式呢? 那就是一个枚举变量可能要代表多个枚举值的时候. 其实给一个枚举变量赋予多个枚举值的时候,原理只是把各个枚举值加起来罢了. 当加起来以后,就获取了一个新的值,那么为了保证这个值的唯一性,这个时候就体现了位运算的重要作用. 位运算可以确保枚举值组合的唯一性. 因为位运算的计算方式是将二进制转换成十进制,也就是说,枚举值里面存取的是 计算后的十进制值. 打个比方: 通过上面的位运算方式设定好枚举以后,打印出来的枚举值分别是: 1 2 4 8 16 这5个数字,无论你如何组合在一起,也不会产生两个同样的数字.

98，iOS9关键字   
\_Nullable:属性可以为空   
\_Nonnull:属性可以为空（只能修饰对象）

泛型一般用来修饰集合中元素的类型。通过泛型取出来的对象可以直接当指定的泛型类型使用   
\_covariant:协变性:

99，UIViewController完整生命周期   
-[ViewController initWithNibName:bundle:]；   
-[ViewController init]；   
-[ViewController loadView]；   
-[ViewController viewDidLoad]；   
-[ViewController viewWillDisappear:]；   
-[ViewController viewWillAppear:]；   
-[ViewController viewDidAppear:]；   
-[ViewController viewDidDisappear:]；

100，self = [super init];   
容错处理，先初始化父类，再传给子类，假如父类初始化失败就会返回nil。子类拥有父类的实例和方法。

103.KVO的底层实现   
调用对象的setter方法才会被KVO监听到，直接进行赋值操作不能监听到。p.age可以，p->age不可以   
修改当前对象的isa指针指向生成的派生类，派生类在前面加上NSKVONotifying\_

KVO是怎么进行通知的：   
1、 KVO是对NSObject对象的一个分类，任何一个对象都可以调用KVO

首先保存被观察对象为当前分类的成员属性，因为在分类中，只能用运行时保存，实际上就是设置和属性的关联 ，然后传入策略（copy， retain等），导入<objc/message.h>框架

然后同样是通过运行时修改被观察对象的isa指针的指向

* 1
* 2

2、在派生类中做事情

在生成的派生类中先调用super调用父类的setter方法

然后通过运行时拿到被观察的对象

调用对象的observeValueForKeyPath方法告知外界

* 1
* 2
* 3

104.objectForKey与valueForKey的区别   
从 NSDictionary 取值的时候有两个方法，objectForKey: 和 valueForKey:   
一般的key可以是任意的字符和字母的组合。但是使用valueForKey方法，如果key是以 @ 开头，系统会去掉 key 里的 @ 然后用剩下部分作为 key 执行 [super valueForKey:]，这样就会报valueForUndefinedKey错误   
建议NSDictionary用objectForKey

105.xcode6和xcode5的区别   
1.xcode6没有frameworks，会导入常见的frameworks   
2.xcode6多了launchScreen.xib   
3.没有设置启动界面，默认的可见范围是320\*480   
4.xcode5有pch文件

106.info.plist的作用   
1.是一个字典，存放一些配置的键值对   
2.Bundle name <-> 应用程序的名称   
3.Bundle identifier <-> 应用的唯一标示符,推送的时候通过唯一标示符找到对应的应用   
4.Bundle version string short <-> 上传新的版本要设置比之前的版本号高   
5.通过[NSBundle mainBundle].infoDictionary

107.pch的作用，新增的pch文件需要告诉编译器提前编译，指定pch的路径   
1.存放一些公用的宏   
2.存放一些公用的头文件   
3.自定义Log，NSLog比较消耗资源，在调试的时候使用，发布的时候需要注释   
HKLog(…) NSLog(VA\_ARGS)…表示是可变参数   
4.和C或者C++混编的时候要注意对pch文件进行处理，#ifdef OBJC #endif,否则会报错

108.NSURLConnection和Runloop的关系   
NSURLConnection请求发出去要等待服务器给它发送的数据。在当前线程的runloop中等待网络返回的数据。   
所以放在子线程中不好使，因为子线程的runloop是默认不启动的

109.xib和storyboard对版本控制工具不友好：   
他们含有xcode的版本信息和操作系统的信息，多人操作的时候即使不修改，只要打开就会修改其内容

110.弱引用持有对象，但是不增加引用计数

111.散列函数的设计和特点

1.直接定制法:设计线性函数。特点:简单，均匀，但是要事先知道关键字的分布情况

2.数字分析法:关键字的位数比较大，去数字的一部分

3.平方取中法:不知道关键字的分布，关键字比较小

4.折叠法:将关键字折叠分组，然后相加。特点：不用知道关键字的分布，关键字的位数比较多

5.除留余数法(HOT):选择的被除数最好小于表的长度(最好接近)的质数或者不包括20以下质因子的合数。

* 1
* 2
* 3
* 4
* 5

112.冲突的解决办法

1.开放定址法:一旦发现了冲突，就去寻找下一个空的散列地址

2.再散列函数法:用一个新的散列函数去计算

3.链地址法:在产生冲突的位置创建一个单链表

4.公共溢出区法:设置一个公共溢出区，用来存放出现冲突的地址。

* 1
* 2
* 3
* 4

113.NSInvocation   
NSInvocation方法包装了一次方法调用（方法调用者，方法名，方法参数，方法返回值）   
方法签名：包括了方法名称，参数，返回值，对方法的描述

114.NSTimer和CADisplayLink   
NSTimer需要设置一个时间间隔，当时间间隔和屏幕的刷新频率不一样的时候就会出现时间的堆积，出现卡顿的现象。   
CADisplayLink的selector执行的时间间隔就是屏幕的刷新频率，所以不会卡。

115.进程的特点:   
1.独立性。   
2.动态性：程序是静态的指令集合，进程是活动的指令集合，是动态消亡的   
3.并发性：CPU不断的调度进程。

116.

Core Animation是在后台执行的，不会阻塞主线程，只能作用在CALayer上面，不能作用在UIView上面

CAAnimation遵守CAMediaTiming协议，有三个子类CAAnimationGroup（动画组），CAPropertyAnimation（属性动画），CATransition（转场动画）

CAPropertyAnimation有两个子类：CABasicAnimation（基础动画）和CAKeyFrameAnimation（关键帧动画）

\* CABasicAnimation只能从一个数值（fromValue）变到另一个数值（toValue），而CAKeyframeAnimation会使 用一个NSArray保存这些数值

\* CABasicAnimation可看做是只有2个关键帧的CAKeyframeAnimation

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8

转场动画：   
CATransition：转场代码必须得要和转场动画在同一个方法当中.   
UIView也可以做转场动画

117.UIView动画与核心动画的区别?   
1.核心动画只作用在layer.   
2.核心动画修改的值都是假像.它的真实位置没有发生变化.

什么时候用UIView动画什么时候用核心动画?   
当需要与用户进行交互时用UIView,不需要与用户进行交互时两个都可以.

什么情况用核心动画最多?   
1.转场动画.   
2.帧动画.(属性动画的一种，另外一个是基础动画，可以看做是有两个关键帧的帧动画)   
3.动画组.

118，weak和assign   
weak：用weak修饰，不会让引用计数加一，如果指向的对象被销毁，指针自动置为nil   
assign：用unsafe\_unretained修饰，不会让引用计数加一，如果指向的对象被销毁，不自动将指针置为nil，会报坏内存访问错误

119.react native   
面向组件开发，类都能生成组件，插入到页面中   
FlexBox：弹性盒模型

120.Masonry的底层实现   
用到链式编程

首先创造约束制造者MASConstrainMaker，并且绑定当前的view，生成了一个保存了所有约束的数组

执行传入的block（设置约束）

约束制造者给view安装约束

\* 清空之前所有的约束

\* 遍历约束数组，一个一个安装（安装就用系统的NSLayoutConstrain）

* 1
* 2
* 3
* 4
* 5
* 6

121，RAC被称为函数响应式编程（FRP）框架

[RACReplaySubject sendNext]:

1.保存发送的值

2.遍历自己所有订阅者,发送数据

* 1
* 2

122，排序总结   
排序方法 平均情况 最好情况 最坏情况 辅助空间 稳定性

冒泡排序 O(n^2) O(n) O(n^2) O(1) 稳定   
选择排序 O(n^2) O(n^2) O(n^2) O(1) 不稳定

比较次数：O(n^2) 移动次数：O(n)

插入排序 O(n^2) O(n) O(n^2) O(1) 稳定

希尔排序 O(nlog(n))~O(n^2) O(n^1.3) O(n^2) O(1) 不稳定

堆排序 O(nlog(n)) O(nlog(n)) O(nlog(n)) O(1) 不稳定

归并排序 O(nlog(n)) O(nlog(n)) O(nlog(n)) O(n) 稳定

快速排序 O(nlog(n)) O(n\*log(n)) O(n^2) O(log(n)) 不稳定

* 1
* 2
* 3
* 4
* 5
* 6

外排序：归并排序   
堆排序中建堆过程的时间复杂度是O(n),重建堆的时间复杂度为nlogn   
关键字比较次数同记录初始排列有关的是：冒泡，直接插入，快速，归并

Prim算法的时间复杂度 邻接表存储时,是 O(n+e) 图的时候 是O(n^2)

* 1

124.中缀后缀的转换   
中->后:栈用来进出运算的符号，数字直接输出   
后->中:栈用来进出数字，符号用来将栈中的数字计算输出

125.   
深度优先搜索和谦虚二叉遍历都类似图的深度遍历，都借助栈的数据结构；   
广度优先相关的借助了队列的数据结构，类似图的层序遍历

126.js闭包   
可以读取函数内部的变量，   
让这些变量的值始终保持在内存中。

127.objc和JS相互调用   
OC调用JS:在webViewDidFinishLoad方法中执行js语句   
JS中调用OC:

当加载一个网页之后，假如网页中想调用OC的方法，要在js的方法中改变locations的url指向，然后在shouldStartLoadWithRequest代理方法中获取访问的URL地址，通过解析协议头调用早已经实现好了的方法。

通过导入JavaScriptCore.framework通过KVC获取JSContext上下文，通过上下文执行block，block中就是想   
调用的方法，这个block的名字就是网页中js调用的方法的名字，而且可以通过上下文获取到传进来的参数数组，遍历数组就能取出来相传进来的参数。

128,B-树   
B-树的阶数m决定了节点的分支数（m/2的上整，m）,但是有两个例外，一个是叶节点，另外一个是根节点，根节点允许有两个分支。

129.双向链表节点p的右边插入节点s   
s->left = p; s->right = p->right; p->right->left = s; p->right = s;

双链表的插入两种方法：   
将下面的next换成rlink,pre 换成llink即可   
一、首先处理新插入节点p的后继、前驱，然后再处理后继的前驱、   
p->next = q; //p的后继   
p->pre = q->pre;//p的前驱   
q->pre = p; //(q)的前驱   
p->pre->next = p;///新节点的前驱( p->pre)的后继

二、前两步都一样：   
p->next = q; //p的后继   
p->pre = q->pre;//p的前驱   
q->pre->next = p; //这一步要在前面，   
q->pre = p;

130.精简排序   
插入排序：每次插入的时候，想要插入的元素只和前面的元素比较一次。   
归并排序：两两归并，只比较一次。

131.一个有向图能被拓扑排序的充要条件就是它是一个有向无环图。

入队时，将元素压入s1。   
出队时，判断s2是否为空，如不为空，则直接弹出顶元素；如为空，则将s1的元素逐个“倒入”s2，把最后一个元素弹出并出队。   
这个思路，避免了反复“倒”栈，仅在需要时才“倒”一次

132贪心算法   
单源最短路径中的Dijkstra算法   
最小生成树的Prim算法   
最小生成树的Kruskal算法