1.xib为什么用weak修饰？

因为拖拽的时候回自动加到subviews数组中，进行了强引用

2.进程：进程是系统进行资源分配和调度的一个独立单位

线程是进程的一个实体，是CPU调度和分派的基本单位

3.为什么默认情况下无法修改被block捕获的值？—block都做了哪些？

默认情况下，block里边的变量，拷贝进去的是变量的值。—block修饰后，拷贝进去的是指向变量的指针，所以我们就可以修改变量的值

4.视图的混合中opaque和alpha对性能的影响

当视图混合图层比较多时，应该表明opaque属性以避免无用的alpha通道合成。opaque设置为YES时，系统就会认为时不透明的，就不会再去进行alpha合成。反之如果alpha为小于1的值的话，opaque就应该设置为NO

5.单元测试

6.SDWebImage实现机制

7.KVC/KVO

KVC：KVC是一个非正式协议，KVC提供了一种间接访问其属性方法或成员变量的机制，可以通过字符串来访问对应的属性方法和成员变量。KVC使用场景：利用KVC可以修改类的私有变量，可以修改IOS隐藏一些属性，如UITextField的placeHolderText默认style在需求中达不到要求，我们可以直接通过KVC快速定义自己的style,代码如下：

[textFieldsetValue:[UIColorredColor]forKeyPath:@"placeholderLabel.textColor"];

KVO：其实就是隐藏修改了对象的isa指针，在当对象(B)被监听时，那么系统就会在运行期动态的创建该对象类的一个子类，类名就是在该类的前面加上NSKVONotifying\_的前缀，子类并重写了任何被监听属性的setter方法，并使用willChangeValueForKey和didChangeValueForKey即手动触发方式来实现，这么做是基于设置属性会调用setter方法(KVC协议)。

8.正则相关知识

9.什么是堆和栈？他们在哪儿？

桟地址要高于堆地址，iOS系统中桟内存大小只有1M。桟由编译器自动分配释放，存放函数的参数值，局部变量等；堆由程序员手动分配释放。

10.转场动画知识

11.Swift和Objective-C异同点

Swift的安全性在于：声明的变量默认都是非空的，于是声明的时候如果不给初始化赋值，就会报错；如果声明为可选类型的变量，使用的时候就会涉及到解包的流程，就能对非空进行必要的判断，于是乎在代码编译阶段就加强了对空值的判断，大大增加了代码的稳定性

a.oc是一门动态语言，而Swift是一门静态语言,告诉机器的越多，运行就越快，提高性能，减少运行时的bug，便于后期维护扩展

b.Swift更加安全，可选类型增加非空判断

c.oc中的数组，字典等引用类型在Swift中都是值类型，栈上提升性能

d.Swift更加简洁，即是面相对象也是函数式编程

12.Runtime用法

a.methodSwizzling class\_getInstanceMethod & class\_exchangeImplementation()，用于适应不同手机屏幕字体大小的调整

b.查找所有成员变量，动态添加方法和属性

找不到selector时就会执行+(BOOL)resolveInstanceMethod:(SEL)sel方法，在其中执行class\_addMethod方法即可

c.关联对象，可用于封装alertView，在分类中动态添加属性

objc\_setAssociatedObject和objc\_getAssociatedObject

13.RunLoop相关知识

Runloop就是为了保证进程的存活，每一个线程都对应一个runloop，实时的监听事件和Timer。由五部分组成：

CFRunLoopSource0:自定义方法触发

CFRunLoopSource1:port方法触发，即系统内部触发

CFRunLoopTimer:用户监听定时器的触发

CFRunLoopMode:Default、Initialization、Tracking、Common、EventRecieve（通常用不到，用于监听系统内部）

CFRunLoopObserver:可以利用这个机制进行性能优化，例如监听runloop运行状态进行tableView的行高度缓存

Runloop的几种状态：

kCFRunLoopEntry 即将进入RunLoop

kCFRunLoopBeforeTimers 即将处理Timer

kCFRunLoopBeforeSources 即将处理Source

kCFRunLoopBeforeWaiting 即将进入休眠

kCFRunLoopAfterWaiting 刚从休眠中唤醒

kCFRunLoopExit 即将退出RunLoop

AFNetworking中希望能在后台线程接收到Delegate回调，所以单独创建了一个线程，并在这个线程中启动了一个Runloop。Runloop启动前内部必须有一个Source/Timer/Observer，所以AFNetworking在run之前先创建了一个新的NSMachPort添加进去了。通常情况下，调用者需要持有这个NSMachPort并在外部线程通过这个port发送消息到loop内。但此处添加的port只是为了让Runloop不至于退出，并没有用于实际的消息发送。当需要这个后台线程执行任务时，AFNetworking通过调用[NSObject performSelector:onThread:]将这个任务扔到了后台线程的Runloop中

14.为什么说OC是一门动态语言

a.动态类型，即运行时再决定类型，任何对象都可以被id指针所指，只有在运行时才能决定是什么类型。内置的int,NSString都属于静态类型，静态类型在编译的时候就能被识别出来，要注意区别编译时和运行时。

b.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相应的消息也被完全确定，这就是动态绑定。当发送respondsToSelector时，在OC消息转发机制被触发之前，对应类的resolveClassMethod将会被调用，可以在此动态添加方法

c.动态加载，根据不同机型动态加载不同的图片

15．项目中用过GCD吗

a.延时操作

b.dispatch\_group\_enter,dispatch\_group\_leave,dispatch\_group\_notify确保所有添加到group中的任务完成后再在notify中执行最后的操作

c.GCD使用信号量控制并发数

d.dispatch\_barrier添加任务之间的依赖关系

16.Category实现原理

通过runtime对category做了以下处理：1.将Category和它的主类注册到哈希表中；2.如果主类已实现，那么重建它的方法列表。在这里分了两种情况做处理：Category中的实例方法和属性被整合到主类中。而类方法则被整合到元类中。协议处理比较特殊，被同时整合到了主类和元类中。

最终都是通过调用static void remethodizeClass(Class cls)函数来重新整理类的数据的。通过源码可以看到真正处理Categoty方法的是attachMethodLists函数，主要作用是将类中的旧有方法和Category中新添加的方法合成一个新的方法列表，并赋值给method\_lists。

17.id可以指向任意对象的原因

Objective-C中的对象的定义是struct objc\_object,本质上是结构体对象，其中isa是它唯一的私有成员变量。Class和id分别是struct objc\_class和struct objc\_object类型的指针

18.实例方法和类方法的区别

实例方法是保存在类中的，而类方法是保存在元类中的。OC中类也是对象，也是某个类的实例，这个类就是元类（metaclass）。我们调用类方法时，给类对象发送消息。同样的，类对象能否响应这个消息也要通过isa找到类对象所属的类（元类）才能知道。而实例对象的元类就其所属的类。根元类的superClass不是nil而是根类，根类的superClass为nil



19.分类为什么可以添加方法但不能添加属性

因为Category在runtime中使用一个结构体表示的，它里边和NSObject的结构体不一样，分类中有instance\_methods,class\_methods,protocols,properties几个字段，没有ivarList，即成员变量，而表示ObjC类的结构体运行时并不能改变，所以可以添加方法但不能添加属性。

20.block为什么要用copy修饰

当block没有访问外界的变量时，是存储在代码区。当block访问外界变量时是存储在桟区，而此时的block处理作用域就会被释放，再去访问时就会出现野指针访问，所以要用copy修饰使block在堆中保留一份，相当于一直强引用着,延长其生命周期。因此如果block中用到self时，需要\_\_weak将其弱化。在ARC模式下，系统也会自动对block进行copy操作

\_\_block:block不允许修改外部变量的值，这里所说的外部变量的的值指的是桟中指针的内存地址。\_\_block所起到的作用就是将外部变量在桟中的内存地址放到了堆中,进而在block内部也可以修改“外部变量”的值。

其实还是可以修改指针指向的内存地址中的内容的，比如没有用\_\_block修饰的时候，block外部创建了一个可变数组，在block中可以通过[arr addObject:]的方式进行修改，但是不能用[NSMutableArray arrayWithObjects:]的方式进行修改。

\_\_weak:避免block中发生循环引用，只能在ARC模式下使用，只能修饰对象，不能修饰基本数据类型

\_\_strong:保证代码在执行block的时候，修饰的对象不会被释放。当block执行完后，会自动释放该strongSelf

21.热更新技术有哪些

a.JSPatch:最根本的原因是OC是一门动态语言，OC上所有方法的调用都通过OC的Runtime在运行时进行，我们可以通过类名和方法名反射得到相应的类和方法

b.Weex

c.React Native

d.Hybrid

e.DynamicCocoa

22.事件传递流程

只有继承了UIResponder的对象才能接受并处理事件，我们称之为“响应者对象”，如UIApplication,UIViewController,UIView

传递流程是从父控件传递到子控件，也就是系统将改时间加到UIApplication管理的一个队列（FIFO）中，UIApplication->window->寻找处理事件最合适的View

事件响应链则是反过来传递，调用touches的一系列方法

23.main()函数之前的过程有哪些

动态链接库，用到的所有framework都是动态连接的，写代码前动态链接器需要完成准备工作。

整个事件有dyld（the dynamic link editor）主导，完成运行环境的初始化后，配合ImageLoader将二进制文件按格式加载到内存，动态链接依赖库，并由runtime负责加载成objc定义的结构，然后再执行load方法，所有初始化工作结束后，dyld才开始用真正的main函数。之后再去执行applicationDidFinishLaunch代理方法

24.load方法相关知识

+(void)load方法与其他方法有很大的不同。它只是一个在整个文件被加载到运行时，在main函数调用之前被objc运行时调用的钩子方法，父类先于子类，类先于分类执行；+initialize方法是在第一次调用该类的方法的时候调用（有点类似于lazy load原理）。

1. 两个方法都不需要显式的调用[super load]或[super initialize]方法
2. 如果子类调用了load方法，父类的load方法也会被调用，而且先于子类，如果子类不实现此方法的话父类也不会调用；initialize不同，子类没有调用initialize的时候也会调用父类的initialize方法
3. 两个方法都能保证线程安全,load常用来处理method swizzling;initialize常用来初始化全局变量和静态变量

25.项目内存优化

a.opaque不透明时设置为YES避免不必要的渲染

b.调整图片大小与imageView大小一样，临时调整大小会非常占用内存。比如WorkOS中有一个界面就是好多图片，然后程序崩溃。解决办法使用后台线程使用CoreGraphic对图片进行一次等比缩放，然后在UIImageView中使用缩放后的图片

c.适当的使用lazy load

d.优化tableView---缓存行高(利用runloop)、减少subView的数量和视图层级、重用、缓存网络数据

26.weak修饰符的相关知识

1.用一个字典objectContainerDataStructure将所有被弱引用的对象以key-Value的形式存放起来。对象的内存地址做key,value就是这个对象所有的弱引用数组，可以用哈希表实现。

哈希表：哈希方法在“键值对”的存储位置与他的键之间建立一个确定的对应函数关系hash()，每一个键与结构中的一个唯一的存储位置相对应：存储位置=hash(键)。在搜索时，首先对键进行hash运算，把求得的值当做“键值对”的存储位置，在结构中按照此位置取“键值对”进行比较，若键相等，则表示搜索成功。在存储“键值对”的时候，依照小那个痛的hash函数计算存储位置，并按照此位置存放，这种方法就叫做哈希算法，也叫做散列方法。哈希方法中使用的转换函数hash被称作哈希函数。按照此算法构造出来的表叫做哈希表或者散列表

2.一个对象可能被多次弱引用，当这个对象被销毁的时候，我们需要找到这个对象所有的弱引用，所以我们需要将这些弱引用的地址（即指针）放在一个数组中pointerContainerDataStructure;

这样当对象被销毁的时候就可以从字典中找到这个对象所有的弱引用，将其置为nil。所以过多的使用weak属性会涉及到很多查表操作，一定程度上会影响性能。

27.SDWebImage使用过程中注意的问题

同时加载多张大图的时候会遇到内存剧增问题，是因为decodedImageWithImage将图片做的减压缩处理，通过操作像素、bitMap，如果图片分辨率过高,就会占用大量内存。此处可以将获取的图片通过Core Graphic进行重绘、宽高等比缩放来解决

图片处理内存优化问题：处理大分辨率图片时，调用drawInRect:方法时，先解码图片，再生成原始分辨率大小的bitmap，这是很消耗内存的。解决方法是用更底层的ImageIO接口CGImageSourceCreateThumbnailAtIndex:来处理，避免中间bitmap生成。

28.多线程之Dispatch

GCD引入了编程范式的变化，使从线程和线程函数的角度思考变为从任务和队列的角度思考。GCD使用队列来派发任务（串行队列和并行队列），派发的方法又分为同步派发和异步派发。队列都是先进先出的，所以串行队列和并发队列都是按照顺序执行的。不同的是串行队列需要等到前一个任务完成才可执行下一个任务，所以串行队列只需要一个线程就可以完成任务派发。而并发队列可以允许多个任务同时执行，虽然开始执行的时间是按照顺序的，但是执行完的时间并不确定。主队列是一个特殊的串行队列，主队列的任务只能在主线程执行，并且需要等待主线程Runloop空闲时才能派发。凡是派发到主队列的任务都会在主线程执行。主队列同步阻塞，异步不阻塞。

死锁：如果在当前串行队列正在执行的任务所在的线程继续向当前队列同步派发任务，就会造成死锁。

dispatch\_barrier\_sync/async（异步不影响主线程） 可以用来给线程添加依赖，实现类似于线程锁的机制，用于任务按顺序执行

dispatch\_semaphore\_create/signal/wait 信号量控制多线程并发数

dispatch\_group\_enter/leave/notify 控制同步

29.多线程NSOperation

<http://xuyafei.cn/post/draft/ios-thread>

NSOpetation是对GCD的API进行了面向对象的封装，GCD中的任务对应NSOperation对象，GCD中的队列对应NSOperationQueue对象。NSOperation还提供判断执行状态、取消任务、控制线程数量等更多任务管理的API。

NSOperation中的任务只能执行一次。将NSOperation添加到NSOperationQueue中后就会在子线程自动执行。如果不想用NSOperationQueue，也可以手动调用NSOperation的start方法来执行它。

NSBlockOperation：通过Block的形式管理多个任务并发执行。可以调用addExecutionBlock添加多个Block。调用start方法后，数组中的所有block会并发执行，第一个Block在当前线程，其余会在新建的子线程执行。当所有block都执行完后才算完成。

NSInvocationOperation：通过Invocation的形式管理单个任务的执行。只能通过initWithTarget的形式初始化，他实现了非并发的operation

NSOperationQueue：可以控制最大并发数、暂停/恢复队列任务派发，获取暂停状态、取消所有operation

30.深拷贝浅拷贝问题以及strong和copy的区别（两个不同的问题）

浅拷贝就是对内存地址的复制，让目标对象指针和源对象指针指向同一片内存区域。深拷贝就是产生一个新的对象，将源对象中的内容拷贝到新的对象中，新对象和源对象各自指向各自的内存区域，相互之间不受影响。简单说就是浅拷贝是指针拷贝，深拷贝是内容拷贝。

对于容器对象和非容器对象来说，可变对象的复制（copy和mutableCopy）都是深拷贝；不可变对象copy是浅拷贝，mutableCopy是深拷贝。对于容器类对象，其元素拷贝永远都是指针拷贝。

strong：指向并拥有对象，其修饰的对象引用计数会加1，只要引用计数不为0就不会被销毁。

Copy:作用与strong相似，不同之处在于strong的复制是多个指针指向同一个地址；而copy的复制是每次会在内存中拷贝一份新的对象，指向不同的地址。Copy一般用在修饰有可变对应类型的不可变对象上，如NSString,NSArray.

31.加密算法相关知识，了解一下有哪些

31.快速排序、遍历一个树、求最大公共子字符串

http://ask.csdn.net/questions/238451

32.自己实现线程池、通知中心等思路

33.线程安全锁

Atomic和nonatomic修饰的属性区别在于系统自动生成的setter/getter方法不一样。如果自己写setter/getter方法，那这些关键字只是起到一个提示作用，写不写都一样。前者系统会保证s/g方法的完整性，不受其他线程影响。保证在别的线程来访问这个属性之前，先执行完当前流程。比如线程A的getter方法运行到一半，线程B调用了setter方法，那么线程A还是能得到一个完好无损的对象。后者要比前者快，但是前者不保证线程安全。A调getter,B、C同时调用setter,则A得到的值就不确定。也就是说atomic保证属性读写安全，但不保证线程安全。因为别的线程还能进行读写之外的其他操作。

1. NSLock
2. @synchronized(obj)关键字来实现锁
3. GCD来实现的锁dispatch\_semaphore\_create(1)
4. NSRecursiveLock 递归锁
5. NSConditionLock 条件锁
6. NSDistributedLock 分布式锁
7. OSSpinLock 自旋锁

34.代码编译过程原理

Xcode的默认编译器是clang，他的功能是首先对OC代码做分析检查，然后将其转换为低级的类汇编代码：LLVM Intermediate Representation(LLVM中间表达码)。接着LLVM会执行相关指令将LLVM IR编译成目标平台上的本地字节码。

当一个工程build之后会执行以下步骤：

预处理、语义和语法分析、生成代码和优化、汇编、链接

iOS编译采用clang作为编译器前端，LLVM作为编译器后端

Clang的任务：

1. 预处理：以#开头的代码预处理，包括引入的自定义头文件和宏
2. 词法分析：每一个.m源文件的声明和定义从string转化为特殊的标记流
3. 语法分析：将标记流解析成一颗抽象语法树
4. 静态分析：包括类型检查和其他检查
5. 中间代码生成：LLVM代码生成

LLVM的任务：

1. 将代码进行优化并产生汇编代码

汇编器：将可读的汇编代码转化为机器代码，最终创建一个目标对象.o文件

链接器的任务：将目标文件和库相连，最终输出可运行文件a.out

App启动流程：

当用户点击一个App，App从启动到打开第一个界面的时间为t=t1+t2;其中t1包含系统dylib和自身app可执行文件（app中所有.o文件的集合）的加载，t2为main函数到appdelegate中didFinishLaunching方法执行结束这段时间。

App启动后，先加载可执行文件，再使用dyld（动态链接器）动态递归加载dylib（系统的framework，runtime的libobjc，系统级别的libSystem）。注册ObjC类，Category插入方法列表，保证每个selector都唯一这些步骤都是在后者中进行。

Main函数之前的加载时间可以用DYLD\_PRINT\_STATISTICS字段来查看

优化程序启动时间：

1. 内嵌的dylib尽可能少或合并
2. 将不必须在+load中的事延迟到+initialize中
3. Release版不要NSLog
4. 不使用xib，直接用代码加载首页图
5. 使用@class代替#import
6. 对常用工具类进行打包
7. 常用头文件放入预编译文件中

程序崩溃统计报告：每次编译之后，都会生成一个dsym文件。在App实际执行的二进制文件中，是通过地址来调用方法的。当crash发生时，第三方工具（Fabric）会帮我们抓到崩溃的调用桟，调用栈里会包含crash地址的调用信息。然后，通过dsym文件，我们就可以由地址映射到函数的具体位置。

35.MVC的理解以及MC和VC之间怎么通信

Controller与Model:C可以直接与Model对话，Model通过KVO和Notification与C间接通信。

Controller与View:C可以通过outlet直接操作V，V通过action向Controller报告事件的发生（如用户Touch我了）C是V的直接数据源和代理。

36.内存泄漏的有哪些情况

a.AFNetworking应该保证httpSessionManager全局只有一个，建议用单例去修改实现

b.block的循环引用问题（MJRefresh的block中调用self）

c.delegate的循环引用问题

d.NSTimer释放时机不当也会造成互相等待循环引用（invalidate放在dealloc中）

e.非OC对象内存处理

37.静态库和动态库的区别

库是程序代码的集合，将N个文件组织起来，是共享程序代码的一种方式。

静态库在链接时，会被完整的复制到可执行文件中，如果多个App使用了同一个静态库，那么每个App都会拷贝一份，缺点是浪费内存。常用的比如讯飞、百度地图SDK等都是静态库。还可以用来把别人写的MRC代码直接打包成静态库使用。静态库的形式有.a和.framework两种，.a+.h+资源文件 = .framework

动态库不会复制，只有一份，程序运行时动态加载到内存中，系统只会调用一次，过个程序公用一份，节约了内存。比如Foundation.framework、UIKit.framework、AppKit.framework等都是动态库。

制作静态库的过程中要注意的几个点：

1. 修改build setting中build active architecture only为NO，适配支持所有的架构，因为不同的模拟器和真机架构是不一样的
2. 可以通过终端一些关键字来查看库所支持的架构lipo -info 文件路径
3. 要把模拟器和真机的静态库合成一个，用lipo -create 库A路径 库B路径 -output 生成文件路径
4. 可以在项目中新建一个target来实现一边开发一边调试

38.单元测试相关知识

39.Hybrid开发，H5与Native互调

40.自动释放池知识

自动释放池（autorelease pool）是OC的一种内存回收机制。当你向一个对象发送一个autorelease消息的时候，cocoa就会将对象的一个引用放入到最新的自动释放池中，它仍然是一个正常的对象，因此自动释放池定义的作用域内的其他对象都可以向他发送消息。自动释放池的本质就是延长释放时机。

如何工作：OC是通过一种引用计数的方式管理内存的，对象开始分配内存的时候（alloc）引用计数为1，以后如果有copy，retain的时候都会加1，每当release和autorelease的时候就会减1，如果为0，就会被系统销毁。Autorelease和release没什么本质区别，知识引用计数减1的时机不一样而已，autorelease会在对象的使用真正结束的时候才做引用计数减1.

实现原理：自动释放池是以栈的形式实现，当你创建一个新的自动释放池的时候，他将被添加到栈顶。当一个对象收到aoturelease消息的时候，它被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，他们就从栈中被删除，并且会给池子里面的所有对象都做一次release操作。

App启动后，苹果在主线程Runloop里注册了两个Observer。第一个监视的事件是Entry，其回调内会调用\_objc\_autoreleasePoolPush()创建自动释放池，其优先级最高，保证创建释放池发生在其他所有回调之前。第二个监听了两个事件：BeforeWaiting时调用\_objc\_autoreleasePoolPop()和\_objc\_autoreleasePoolPush()来释放旧的池并创建新池。Exit时调用\_Pop()来释放自动释放池，这个Observer的order优先级最低，保证其释放池子在所有回调之后

41.APNS流程简单概括

首先APNS是基于长连接的基础之上的，iPhone与APNS建立起一个长连接。

1. 应用程序注册APNS消息推送
2. iOS从APNS Server获取devicetoken，应用程序接收devicetoken
3. 应用程序将device Token发送给程序的PUSH服务端程序
4. 服务端程序向APNS服务发送消息
5. APNS服务将消息发送给iPhone应用程序

42.iOS系统框架里使用了哪些设计模式

a.单例模式

b.观察者模式：通知、KVO(键值观察模式)

c.MVC模式

d.工厂模式：工厂方式创建类的实力

e.策略模式：抽象基类的抽取

f.代理模式

https://xiaozhuanlan.com/topic/3908156472