https://juejin.cn/post/6908303868086452237

网易

**1.**属性关键符都有什么，简单介绍一下，**atomic**是怎么保证线程安全的？**weak**的底层原理？**copy**和**strong**区别

Assign 基本数据类型 区分悬空/野指针

retain get通过地址偏移取值 区分newVlue 生分规则,objc\_retain生成新值并赋值

Copy get通过地址偏移取值 区分newVlue 生分规则,copyWithZone生成新值并赋值

Strong get/set 通过地址偏移取/赋值

weak 通过objc\_initweak传入weak引用对象指针地址,和引用对象内存地址,通过这两个地址找到对应的sideTable,

1>如果弱引用对象内存地址在weak表已经存在,调用unregister,通过sidetable拿到weak\_table,在通过弱引用对象内存地址获取到对应的weak\_entrys,利用弱引用对象地址获取hash,找打对应的index,从weak\_entrys中获取到对应的entry,拿到entry后,遍历entry中referrers数组找到与弱引用对象指针地址相同的value,置空 优化点:内存缩容(清空entry以及entry中referrers,利用entry不满1/16算法,将weak\_table缩小为原来的1/8,释放老entries 开辟新entries) hash冲突处理

2>如果有新值,通过register函数将新weak引用对象的指针地址存储在对应的弱引用表中,通过弱引用对象内存地址取出weak\_table对应entry,将weak引用对象的指针地址拼接在entry的referrers中 细节处理:会处理弱引用表referrers是动态还是内联,在容量达到3/4时,动态扩容原来容量二倍

atomic/nonatomic 多/单线程能保证setter/getter 时线程安全, 但是不能保证在一个线程进行setter/getter 时,另一个线程进行释放对象,从而crash 所以atomic不能保证线程安全

unsafe\_unretained unsafe\_unretained类似于assign 悬空指针,指向内存地址,所指对象释放,该指针还是指向同一个内存地址

**2.ARC**内存管理

1>内存布局

地地址—————>高地址

保留区—->代码段—>常量区(data)—>静态区(bss)—>堆—>栈—>内核区

保留区:

保留区就是保留的一块区域，留个系统处理nil等情况,内存有4MB保留. 就是 处理异常情况

代码段:

常是指用来存放程序执行代码的一块内存区域。这部分区域的大小在程序运行前就已经确定,并且内存区域通常属于只读, 某些架构也允许代码段为可写,即允许修改程序。

常量区:

常量区是一块比较特殊的存储区，常量区里面存放的是常量，比如整形、字符型、浮点型、常量字符串，常量区的内存是在编译阶段进行分配，程序运行时会一直存储在内存中，也是以0x1开头，只有当程序结束后才会由操作系统释放

静态区/全局区:

bss段通常是指用来存放程序中未初始化的全局变量和静态变量的一块内存区域。

通常来说如果不初始化全局变量和静态变量，编译器也会对它们进行一个隐式初始化（直接赋值就是显示初始化），赋给它们一个缺省值，是我们这里所说的未初始化。

bss段在程序执行之前会清0，所以未初始化的全局变量（静态变量）已经是0了。所以这种情况还是存放在BSS段，一旦初始化就会从BSS段中回收掉，转存到data段(数据段)中。

bss区-Block Started by Symbol（未初始化数据段）：并不给该段的数据分配空间，仅仅是记录了数据所需空间的大小

堆区:

堆是用于存放进程运行中被动态分配的内存段,它的大小并不固定,可动态扩张或缩减。当进程调用malloc等函数分配内存时,新分配的内存就被动态添加到堆上(堆被扩张); 当利用free等函数释放内存时,被释放的内存从堆中被剔除(堆被缩减)

堆向高地址扩展的数据结构，是不连续的内存区域。程序员负责在何时释放内存（如用free或delete），在iOS的ARC程序中，系统自动管理计数器，计数器为0的时候，在当次的runloop结束后，释放掉内存。堆中的所有东西都是匿名的，这样不能按名字访问，而只能通过指针访问。

对于堆来讲,频繁的new/delete势必会造成内存空间的不连续性，从而造成大量的碎片 ,使程序效率降低。

一般以0x6开头

栈区:

栈又称堆栈, 是用户存放程序临时创建的局部变量,也就是说我们函数括弧“{}” 中定义的变量(但不包括static声明的变量,static意味着在数据段中存放变量)。除此以外, 在函数被调用时,其参数也会被压入发起调用的进程栈中,并且待到调用结束后,函数的返回值 也会被存放回栈中。由于栈的后进先出特点,所以 栈特别方便用来保存/恢复调用现场。从这个意义上讲,我们可以把堆栈看成一个寄存、交换临时数据的内存区。

指针都存在栈区，用于指向分配在堆区的内存的地址,

优点：

连续区域，存储和查找速度快,一般以0x7开头

编译器自动创建和销毁，安全可靠，不会产生内存碎片

缺点

因为连续，所以大小有限制

在iOS中主线程一般是1M，其他线程是512kb

我们也可以通过线程的stackSpace去修改其大小，但是成本有些大

内核区:

2>储存方式

tagged pointer:

Tagged Pointer 为 Tag+Data 形式

tag区分标识位、类标识、数据类型做代码验证

Number date string 类型

节省存储空间,提升查找效率

https://mp.weixin.qq.com/s?\_\_biz=MzIyMTg0OTExOQ==&mid=2247486305&idx=2&sn=68856c90ba169d7af0034575cc63631a&chksm=e8373811df40b1077fec00830c9eb595a5b750a7b5120ab68ea711ace42d2ebd031c7212d2db&scene=27#wechat\_redirect

堆上存储:

Alloc—objc\_rootalloc—callalloc

1.没有默认的alloc/allocWithzone 获取当前对象内存大小,并内存对齐,调用calloc内存分配空间,初始化当前对象isa\_t

2.走消息转发流程objc\_megsend(allocWithZone/alloc)

内存对齐 : class\_getInstanceSize得到实际所占内存大小 malloc\_size获取实际分配的内存大小,一般是以16字节为最小单位进行分配,其目的是空间换时间,提高查找效率,防止内存溢出

init---objc\_rootinit 返回当前self对象

栈上存储:通1>内存布局->栈的介绍

3>内存管理方式

引用计数方式:

1>>存在isa指针中的extra\_rc中

2>>存在SideTable的RefcountMap中

Densemap

https://chy305chy.github.io/2019/01/23/iOS-Runtime%E9%9A%8F%E7%AC%94%E2%80%94%E2%80%94DenseMap%E6%B5%85%E6%9E%90/

https://draveness.me/rr/

https://www.leewong.cn/2020/08/02/retaincount/#release

Map —bucket —>first:key second:value —>扩容,缩容策略

对象dealloc时,会调用Densemap erase 将对象对应的引用计数清除

Retain 引用计数增加时,rootRatain extra\_rc 加1,extra\_rc达到上限时,has\_sidetable\_rc更新为YES,执行sidetable\_addExtraRC\_nolock,将引用计数加到densemap

4.内存释放

dealloc

释放构造函数

移除关联对象

Weak 表释放

引用计数表释放

MRC内存管理

1>内存布局  
2.储存方式

3>内存管理  
4.内存释放

**3.**介绍**runloop(**源码**)**

1.基础对象介绍

Runloop

currentmode

Modes

commonMode

\_pthread

runloopmode

Source0(集合)

触摸事件(事件队列) performselect

Source0 不能主动唤醒runloop 事件首先由source1接收,并唤醒runloop 在唤醒的runloop 中,source0中标记的未被执行的时间继续执行

Source1(集合)

系统事件

端口事件

observe(集合)

runloopobserve

beforeentry beforetimer beforesource (source0 source1) beforewaiting afterwaiting Exit AllActivities

Timer

Timer是添加到对应的mode中,切换mode时,timer同时被切换,所以 在commonmode时,会将timer添加到runloop的\_commonModeItems中,就避免了mode切换时,timer被停止的问题

2.流程

3.mode切换

退出之前runloop

再次进入runloop,mode已经发生改变

do {

result = CFRunLoopRunSpecific(CFRunLoopGetCurrent(), kCFRunLoopDefaultMode, 1.0e10, false);

CHECK\_FOR\_FORK();

} while (kCFRunLoopRunStopped != result && kCFRunLoopRunFinished != result);

由于while 的存在,runloop会不停的进入退出

4.runloop如何被唤醒 细节

mach\_msg

5.mach\_msg() \*\*\*\*重要重要重要\*\*\*\*

用户态<—>内核态

mach\_msg底层调用 mach\_msg\_trap()

用户态,内核态都必须调用同一个API(mach\_msg) 才能实现通信

runloopMode中 \_portSet 存放对应的runloop 唤醒的端口

runloopMode 中 \_timerPort 用于定时器唤醒runloop端口

6.runloop 调用run 方法 runloop 不生效

source/observe/timer 才能保活runloop

7.runloop run方法

run 内部会无限循环调用 runMode: beforeDate:方法(GNU源码)

8.runloop 子线程内执行performselect: onthread: 取消当前线程runloop

在performselect 中 拿到的是对应线程runloop.如果直接dealloc内取消runloop,拿到的是主线程的runloop,可能会取消错runloop

9.dealloc 执行后,self被释放 注意 自定义 run方法

dealloc后self被释放,在weak 引用的self会被置nil 在自定义 run 方法时

while(weakSelf && isStop){

[runloop runmode: beforeDate:];

}

https://juejin.cn/post/6913094534037504014#heading-1

https://blog.ibireme.com/2015/05/18/runloop/

https://nshipster.com/inter-process-communication/

https://blog.csdn.net/u013378438/article/details/80239686

10.事件响应

用户触发事件， IOKit.framework 生成一个 IOHIDEvent 事件并由 SpringBoard 接收，SpringBoard 会利用 mach port，产生 source1，来唤醒目标 APP 的 com.apple.uikit.eventfetch-thread 的 RunLoop。Eventfetch thread 会将 main runloop 中 \_\_handleEventQueue 所对应的 source0 设置为 signalled == Yes 状态，同时唤醒 main RunLoop。mainRunLoop 则调用 \_\_handleEventQueue 进行事件队列处理。

11.手势识别

如果有手势被触发，在 \_UIGestureRecognizerUpdateObserver 回调中会借助 UIKit 一个内部类 UIGestureEnvironment 来进行一系列处理。

12.界面刷新

当在操作 UI 时，比如改变了 Frame、更新了 UIView/CALayer 的层次时，或者手动调用了 UIView/CALayer 的 setNeedsLayout/setNeedsDisplay 方法后，这个 UIView/CALayer 就被标记为待处理，并被提交到一个全局的容器去。

 苹果注册了一个 Observer 监听 BeforeWaiting(即将进入休眠) 和 Exit (即将退出 Loop) 事件，回调去执行一个很长的函数：\_ZN2CA11Transaction17observer\_callbackEP19\_\_CFRunLoopObservermPv()。这个函数里会遍历所有待处理的 UIView/CAlayer 以执行实际的绘制和调整，并更新 UI 界面。

13.卡顿检测

• 复杂 UI 、图文混排的绘制量过大;

• 在主线程上做网络同步请求;

• 在主线程做大量的 IO 操作;

• 运算量过大，CPU 持续高占用;

• 死锁和主子线程抢锁。

导致GPU渲染帧视图不规律,每次v-sync信号到来时,从帧缓冲区取帧视图,取不到,就会丢失本次帧显示,丢的多了视觉也就有卡顿了

方案:利用子线程内部的while循环,结合信号量,监听observe的activity.需要自定义runloopObserve 加入runloop obeserve自定义回调方法内监听所有activity,并触发信号量signal

14.autoreleasepool与runloop关系

runloop创建时(CFRunLoopObserverCreate),callout参数函数设置为\_wrapRunLoopWithAutoreleasePoolHandler,在runloop的两个observe中

entry:order设置为最高序列-2147483647, \_wrapRunLoopWithAutoreleasePoolHandler 内部调用releasepoolpush

beforetime | exit : \_wrapRunLoopWithAutoreleasePoolHandler:BeforeWaiting(准备进入休眠) 时调用\_objc\_autoreleasePoolPop() 和 \_objc\_autoreleasePoolPush() 释放旧的池并创建新池；Exit(即将退出Loop) 时调用 \_objc\_autoreleasePoolPop() 来释放自动释放池。这个 Observer 的 order 是 2147483647，优先级最低，保证其释放池子发生在其他所有回调之后

15.commonmode

一个普通的mode可以将自身‘标记’为common，具体做法是，系统会将该mode的name添加到\_commonModes中。

当item被加入到common modes下时，首先，会在runloop的\_commonModeItems中添加一个条目，然后，会遍历所有的\_commonModes，将该item加入到已经被标记为common的mode中。

当一个mode作为整体被加入common modes下时，则会进行另外一番操作。系统会调用void CFRunLoopAddCommonMode(CFRunLoopRef rl, CFStringRef modeName)方法。在方法内部，系统首先会将mode置位为common（将mode name加入到\_commonModes）。然后，会将\_commonModeItems中的所有元素，添加到这个mode中。注意，这里mode中的item并不会被加入到\_commonModeItems中。

这意味着，当我们将mode作为整体加入到Common modes中时，该mode可以响应common mode item事件，而其本身自带的mode items，在别的被标记为common的mode中，却不会被响应。

被标记common的mode--> commonmodes -> commonmodeitems(timer,source,observe)-->将所有item加入到commonmodes中的每个mode

**4.**介绍自动释放池

push(传入pool\_boundary 并记录当前pool\_boundary地址) pop(传入pool\_boundary地址,将pool\_boundary地址前的对象调用release,直至pool\_boundary地址) 哨兵对象(pool\_boundary) 双链表 page

应用场景

https://zhuanlan.zhihu.com/p/321687906

https://www.jianshu.com/p/7bd2f85f03dc

创建大量临时变量

不加自动释放池 临时变量需要等当前函数执行完毕才能释放,会造成内存大量占用,使用自动释放池就能稳定的在循环结束时释放掉临时变量

首页channel 创建子vc

红包雨创建大量图片

**5.swift**泛型，写一下代码

泛型是用来使代码能安全工作。在Swift中，泛型可以在函数数据类型和普通数据类型中使用，例如类、结构体或枚举。泛型解决了代码复用的问题。

有一种常见的情况，你有一个方法，需要一个类型的参数，你为了适应另一种类型的参数还得重新再写一遍这个方法。

**Func funcName<T>(\_ a :T,\_ b: T){}**

6.swift的闭包逃逸 与非闭包逃逸

闭包逃逸:

一个接受闭包作为参数的函数，该闭包可能在函数返回后才被调用，也就是说这个闭包逃离了函数的作用域，这种闭包称为逃逸闭包。当你声明一个接受闭包作为形式参数的函数时，你可以在形式参数前写@escaping来明确闭包是允许逃逸。

非闭包逃逸:

一个接受闭包作为参数的函数， 闭包是在这个函数结束前内被调用。

区别:

为了管理内存，闭包会强引用它捕获的所有对象，比如你在闭包中访问了当前控制器的属性、函数，编译器会要求你在闭包中显示 self 的引用，这样闭包会持有当前对象，容易导致循环引用。

而对于非逃逸闭包：

• 非逃逸闭包不会产生循环引用，它会在函数作用域内释放，编译器可以保证在函数结束时闭包会释放它捕获的所有对象。

• 使用非逃逸闭包可以使编译器应用更多强有力的性能优化，例如，当明确了一个闭包的生命周期的话，就可以省去一些保留（retain）和释放（release）的调用。

• 非逃逸闭包它的上下文的内存可以保存在栈上而不是堆上。

综上所述，如果没有特别需要，开发中使用非逃逸闭包是有利于内存优化的，所以苹果把闭包区分为两种，特殊情况时再使用逃逸闭包。

7.flutter的脚手架，flutter的三棵树？

脚手架:

Scaffold脚手架是整个应用的骨架。我们可以将其抽象成还未被填词的旋律。我们拿到它只需要构建内容就完全可以了。当然，发现提供的旋律不是我们想要的也可以将某一部分替换成其他的，甚至整套样式都不喜欢我们也可将默认的样式都替换。

三棵树:

element

widget是element的一个配置

renderobject 是渲染树的一个对象

widget

Widget是描述UI元素Element的配置数据

renderobject

用于应用界面的布局和绘制，保存了元素的大小，布局等信息

MVC模式大家都很熟悉，Model发生变化时通知Controller，Controller再根据变化后的Model对View进行更新。对于Widget、Element和RenderObject这三者来说，Widget就好比是Model，它存放着绘制一个控件所需要的全部数据；RenderObject就好比是View，它用来把Model中的数据转化为屏幕上表现的图像；那么同时持有Widget和RenderObject的Element当然就是Controller了。

8.二叉树删除叶子节点 后序

Void deleteLeafNode(node \*root){

if(!root) return;

deleteLeafNode(root->leftNode);

deleteLeafNode(root->rightNode);

if(!root->leftNode && !root-> rightNode) root = nil;

}

9.数组的全排列

Leetcode 46 47

10.RN的底层原理，RN和Native交互

11.项目中遇到的难题怎么解决的？

跨部门落实方案,协同各方问题

项目重构风险把控

项目组件化

首页重构

直播间重构方案

具体业务技术难点

音娱时光

1.http的请求方法有哪些？讲一下

https://juejin.cn/post/6844903607964270599

post

POST请求会 向指定资源提交数据，请求服务器进行处理，如：表单数据提交、文件上传等，请求数据会被包含在请求体中。POST方法是非幂等的方法，因为这个请求可能会创建新的资源或/和修改现有资源。

Get

GET请求会显示请求指定的资源。一般来说GET方法应该只用于数据的读取，而不应当用于会产生副作用的非幂等的操作中。它期望的应该是而且应该是安全的和幂等的。这里的安全指的是，请求不会影响到资源的状态。

Delete

DELETE请求用于请求服务器删除所请求URI（统一资源标识符，Uniform Resource Identifier）所标识的资源。DELETE请求后指定资源会被删除，DELETE方法也是幂等的。

Put

PUT请求会身向指定资源位置上传其最新内容，PUT方法是幂等的方法。通过该方法客户端可以将指定资源的最新数据传送给服务器取代指定的资源的内容。

head

HEAD方法与GET方法一样，都是向服务器发出指定资源的请求。但是，服务器在响应HEAD请求时不会回传资源的内容部分，即：响应主体。这样，我们可以不传输全部内容的情况下，就可以获取服务器的响应头信息。HEAD方法常被用于客户端查看服务器的性能。

OPTIONS

OPTIONS请求与HEAD类似，一般也是用于客户端查看服务器的性能。 这个方法会请求服务器返回该资源所支持的所有HTTP请求方法，该方法会用'\*'来代替资源名称，向服务器发送OPTIONS请求，可以测试服务器功能是否正常。JavaScript的XMLHttpRequest对象进行CORS跨域资源共享时，就是使用OPTIONS方法发送嗅探请求，以判断是否有对指定资源的访问权限。

TRACE

TRACE请求服务器回显其收到的请求信息，该方法主要用于HTTP请求的测试或诊断

CONNECT

CONNECT方法是HTTP/1.1协议预留的，能够将连接改为管道方式的代理服务器。通常用于SSL加密服务器的链接与非加密的HTTP代理服务器的通信。

patch

PATCH方法出现的较晚，它在2010年的RFC 5789标准中被定义。PATCH请求与PUT请求类似，同样用于资源的更新。二者有以下两点不同：1.PATCH一般用于资源的部分更新，而PUT一般用于资源的整体更新。2.当资源不存在时，PATCH会创建一个新的资源，而PUT只会对已在资源进行更新。

2.http响应码有哪些

https://juejin.cn/post/6877362691350986766

3.UDP和TCP的区别

https://juejin.cn/post/6877362691350986766#heading-34

UDP: 不安全, 不校验, 无连接

TCP:安全性高,丢包检查,重传 有连接(三次握手,四次挥手)

4.UIViewController的生命周期，用过ChildViewController吗？initialize 和loadView谁先被调用

initialize 先调用,类的加载时就调用

loadView 类的view初始化的时候才调用

5.runloop的生命周期

参考网易问题

6.有几种block，block的底层实现，block对外面变量的捕获

globalBlock

没有访问临时变量

MallocBlock

stackBlock 调用了copy操作

stackBlock

访问了临时变量,mrc下捕获临时变量是stackBlock,Arc下捕获临时变量自动在copy一次,就成了mallocblock

usingBlock API中,block作为参数,以及 block 没有被强引用,例如NSLog(@“%@”,[^{} class]) 都是存在栈区

栈上的block 不会对局部变量产生引用(包括\_\_block修饰的局部变量,普通局部变量)

堆上的block 都会涉及到内存管理,对变量进行copy操作,变量之前弱引用,block在堆上也会弱引用,变量之前强引用,block在堆上也会强引用

block底层实现:

Block是封装函数以及函数调用的OC对象:block\_impl结构体的isa指向对应类对象,打印对应的superclass 最终继承自NSBlock,NSObject

底层实际是block\_impl结构体对象,结构体中,isa : 分别是类对象 funcptr :block{}内部实现 desc: block\_impl的详情介绍

调用: 通过获取\_\_main\_block\_impl\_0首地址(即impl的结构体地址),获取到funcPtr,调用funcPtr(\_\_main\_block\_func\_0)

局部变量为什么要被捕获,因为要跨函数访问,只有捕获到block\_impl中,才能被访问到

block对外面变量的捕获:

全局变量:直接访问

全局静态变量:直接访问

局部静态变量:指针拷贝

局部变量:值拷贝

捕获实例变量,实际上是捕获当前self,在通过self去查找实例变量

\_\_block:

修饰的对象或基本数据类型,被捕获到一个新生成的结构体中, 如果是对象,会涉及到引用计数管理(block\_assign | block\_dispose)

Forwarding

通过block \_\_forwarding 取值,能保证取到的值是保存在堆上的值, 栈上的block,\_\_forwarding指向自己,ARC下会将栈的blockcopy到堆上,同事\_\_forwarding也会指向堆上的block,而堆上block自己的\_\_forwarding指向自己

Block 使用copy,是为了将block的生命周期自己控制 在栈的block生命周期不可控,风险也比较高

7.OC消息发送和消息转发的流程

参考网易问题

8.介绍属性修饰符

参考网易问题

9.自动释放池对里面的对象是怎么管理的

参考网易问题

10.ARC内存管理

参考网易问题

11.frame和bounds的区别

https://blog.csdn.net/lihao\_ios/article/details/107540979

frame: 参考父视图坐标系

修改frame的origin,参考自身左上角进行移动,子视图不收影响,center会改变

修改frame的size,不影响位置和子视图布局,center会改变

旋转自身,frame可能会发生改变,因为size的大小是当前视图所占矩形的大小,例如正方形旋转成菱形,origin,size都会改变,菱形所占对应的矩形所占面积大于未旋转前的正方形,center不会改变

Bounds:参考自身坐标系

修改bounds的origin,自身不受影响,子视图受到影响.bounds的改变会使自身的坐标原点改变,子视图是依靠父视图的坐标原点进行布局,如改成(-50,-50),子视图的origin会增加50,子视图center会改变

修改bounds的size,中心点继续保持不变，长宽进行改变，通过bounds修改长宽就像是以中心点为基准点对长宽两边同时进行缩放,center是根据父容器的相对位置来计算的，无论是修改父容器的bounds还是自身的bounds，都不会改变center。况且使用bounds来缩放view，都是根据center中心点来缩放的，所以center不会改变

scrollview 的滚动就是借助手势,和父视图的bounds的改变,出现子视图滚动效果 参考scrollview的源码

12.layoutsubviews啥时候调用？

https://juejin.cn/post/6858057293088194567

https://blog.csdn.net/u013378438/article/details/80239686

在当前视图有UI修改或图层合并的标记动作时,会调用layoutsubviews,标签动作是setneedsdisplay/setneedslayout触发, 在当前的runloops进入beforewaiting时,会调用layoutsubviews

init初始化不会触发layoutSubviews。

addSubview会触发layoutSubviews。

设置view的Frame会触发layoutSubviews，当然前提是frame的值设置前后发生了变化。

滚动一个UIScrollView会触发layoutSubviews。

旋转Screen会触发父UIView上的layoutSubviews事件。

改变一个UIView大小的时候也会触发父UIView上的layoutSubviews事件。

直接调用setLayoutSubviews。

13.讲一下cookie

https://my.oschina.net/u/4030990/blog/3136476

解决的问题:让无状态的HTTP 请求,变的有状态

生成:有服务器产生,放入到Set-Cookie字段里 通过响应传给客户端,客户端保存,每次请求时,将给服务区的标识以key-value形式放到cookie中,服务端通过检查cookie中的字段,区分用户

Cookie通过自己的属性来保证cookie安全性

Session也是由服务器创建,同时创建对应的session\_id,客户端保存session\_id,其目的是防止用户的隐私信息被窃取,比cookie安全性高,session\_id 是放在cookie中进行传递,服务器每次需要校验session\_id的正确性

过期也cookie一样,需要设置,默认过期时间30分钟

Token由服务器通过用户信息生成JWT的格式数据,传输给客户端由客户端保存,由于JWT中带有多种数据,相比session,后端就不涉及到共享的问题

Token是以字符串json的格式,体积小,比session好一些

14.KVO的实现原理

instance 对象—>NSKVONotifify\_类->原instance父类

NSKVONotifify\_ 重写set方法,在方法中调用valueandnotify方法,在valueandnotify中调用willchangevalueforkey 和didchangevalueforkey ,在didchangevalueforkey中调用observevalueforkeypath:

NSKVONotifify\_类,会重写class 方法,在class中调用superclass 的class方法,其目的就是为了掩盖生成中间类

生成中间类的好处:

隐藏内部监听属性变化的变化

集合类中数据更改不会触发KVO,KVO监听的是集合类的地址,数据的改变不会影响地址变化

直接修改成员变量不会触发KVO,因为需要重写成员变量的set方法才能触发,成员变量没有set方法

15.setNeedsLayout() 和layoutIfNeeded()之间的区别

setNeedsLayout() 是标记当前UI需要刷新,等到runloop即将休眠时,会统一将标记的UI布局进行刷新(调用layoutsubviews)

layoutIfNeeded() 是要求当前UI立即刷新,前提是UI或者frame要有变化

16.项目中什么地方用到了GCD

Timer

获取验证码

登录

直播间定时器

dispatch\_source\_create(DISPATCH\_SOURCE\_TYPE\_TIMER, 0, 0, \_queue)

dispatch\_time

dispatch\_source\_set\_timer

dispatch\_source\_set\_event\_handler

dispatch\_resume

信号量

dispatch\_semaphore\_create

dispatch\_semaphore\_signal

dispatch\_semaphore\_wait

礼物大动效视频下载

流的清晰度切换卡顿问题

异步线程

dispatch\_sync

收到socket时

拉取礼物数据

首页换肤

异步组队列队列

dispatch\_group\_leave dispatch\_group\_enter

和dispatch\_group\_async 一样,dispatch\_group\_enter dispatch\_group\_leave 也是对block的管理

接口数据请求

单例

dispatch\_once

利用系统的锁,操作oncetoken,做标记控制有无实现

dispatch\_after

延迟调用

异步

任务可取消

需要自定义dispatch\_block 替换dispatch\_after中的block ,取消时调用dispatch\_block\_cancel

17.什么情况下会触发离屏渲染

https://juejin.cn/post/6847902222567604231#heading-0

离屏渲染:

由于一些渲染操作无法存到帧缓冲区,需要单独开辟一块空间,存放未渲染完的帧数据.

例如切圆角 父视图与子视图的渲染已经完成,但是需要切圆角,就需要将渲染好的帧数据放到离屏缓存区,合并帧,在进行切圆角,处理完成后,在将帧数据移到帧缓冲区

1.Cornerradius +cliptobounds && 子视图也一并裁切 才能触发离屏渲染

2.图片 Cornerradius +cliptobounds && 设置背景色 才能触发离屏渲染  
3.采用了光栅化的 layer (layer.shouldRasterize)

houldRasterize开启后，会将layer作为位图保存下来，下次直接与其他内容进行混合。这个保存的位置就是OffscreenBuffer中。这样下次需要再次渲染的时候，就可以直接拿来使用了。

4.使用了 mask 的 layer (layer.mask)

5.需要进行裁剪的 layer (layer.masksToBounds /view.clipsToBounds)

6.设置了组透明度为 YES，并且透明度不为 1 的layer (layer.allowsGroupOpacity/ layer.opacity)

7.使用了高斯模糊

8.添加了投影的 layer (layer.shadow\*)

9.绘制了文字的 layer (UILabel, CATextLayer, Core Text 等)

18.runtime的底层实现

参考网易问题

19.什么情况下会触发离屏渲染

参考17题

20.事件传递

事件传递:自底向上

响应链:自上而下

问题总结 63个问题

字节

1.崩溃日志的崩溃分哪几种？

代码层崩溃

>数组越界

项目中 用的NSArray+safe 分类 判断存入的值是否合规 取值的index是否越界

>字典存nil

项目中 用的NSDictionory + safe 分类 判断存入的value是否合规 取值的key是否为空

>方法找不到

Avoidcrash三方库 利用的消息转发方法 获取异常 并重写实现

>对象类型用错

字典当成可变字典用

>遍历修改过程中,修改集合

例如 for each

>死递归

递归不结束 占用大量函数栈 和内存 最终由于内存不够 出现crash

>野指针

对象指针指向一块已经被释放的地址,取的对象不可控,在使用中很可能奔溃

>内部NSAssert 系统断言

指定的规则中,获取的数据或者类型不能满足规则 产生crash

>线程死锁

当前主线程任务 等待 dispatch\_get\_main\_queue中的任务结束,产生死锁

>多线程读取数据

dispatch\_group\_enter 比dispatch\_group\_leave次数少

2.系统级崩溃

>看门狗

应用超出以上场景,系统就会启用看门狗机制,监控当前场景下应用的性能,超出规定时间,杀死当前应用

>内存过低(OOM)

内存过低,系统杀死APP应用

https://blog.csdn.net/TuGeLe/article/details/104004692

2.如何做的启动优化

>main函数前

**dylib loading time** 动态库加载

动态加载程序查找并读取应用程序使用的依赖动态库。每个库本身都可能有依赖项。虽然苹果系统框架的加载是高度优化的，但加载嵌入式框架可能会很耗时。为了加快动态库的加载速度，苹果建议您使用更少的动态库，或者考虑合并它们

系统动态库除外 快手分发平台SDK 人脸识别SDK

>> 1.动态库引用方式修改为静态库引用,能提高0.5s左右,但是会增大包体积

>>2.动态库二进制合并(项目较少 没有用到)

lipo create output 动态库的名字 输出路径

>>3.cocoapod 提高启动

use\_frameworks 使用 生成对应的framework 生成dynamic库,增加运行时负担,只会将连接路径拷贝到项目中

use\_frameworks 关闭生成静态文件(.a文件),会将库整个拷贝到项目中,增大项目体积,增加编译负担

rebase/binding time 镜像指针的调整

修正调整镜像内的指针(重新调整)和设置指向镜像外符号的指针(绑定)。为了加快重新定位/绑定时间，我们需要更少的指针修复

多余代码,多余的类,分类删除(OCLint)

initializer time初始化程序

Load 函数加载的使用会有4ms的耗时,减少Load函数的加载,可以使用initialize替换

**ObjC setup time** 运行时设置类 **,**分类选择器阶段

减少类 & 分类

>main函数后

三方库加载时机延迟,比如讲主播人脸识别的SDK,播放器的SDK等

>首屏渲染后

视图懒加载,耗时操作异步,比如首页换肤资源下载 礼物资源下载

监控

打印时间

DYLD\_PRINT\_STATISTICS查看各个阶段的时间

instrument

第三方工具

进程创建sysctl函数—main函数

借助fishhook hook objc\_msgsend 函数 获取函数的调用栈 用过时间star end计算时间

main函数—rootVC的viewdidAppear首屏首次绘制完成(用runloop比较准)

3.App是如何进行瘦身的，瘦身的目的是？

https://blog.csdn.net/olsQ93038o99S/article/details/117393089

删除多余指令架构

只保留64位arm64

build setting 编译二进制优化配置

代码量

利用ununsesource 查找无用文件 利用OClint 优化代码减少代码量

APPCode 的静态分析 可以查找未使用文件

资源

ununsesource删除无用图片 视频 json文件

tinypng上压缩文件 现在项目用的png

> 有损压缩

jpg webp   
> 无损压缩

png svg

Jpg 像素色彩 > png > webp > svg

体积:webp<svg<JGP<PNG

Jpg图片被替换成SVG图片,在色彩不丰富的情况下 矢量图计算色彩相对像素更消耗性能

App Thinning 有三种方式，包括：App Slicing、Bitcode、On-Demand Resources。

>App Slicing :苹果根据设备自动创建包变体

目的:提升APP store用户下载体验 超过150M WiFi网络下才能下载

4.swift是面向对象的语言吗

是,既面向对象也面向过程,具备了对象的三大特性

5.js的虚拟DOM

6.RN如何做到js和OC互调

7.RN是单线程的Native的数据是怎么传的

8.RN是如何启动的（启动时机）

9.RN渲染是怎么调用UIKit进行渲染的

10.MVC和MVVM是设计模式吗

不是,MVC MVVM是设计模式的结合的一设计理念 ,也就是设计架构

MVC 是一种职责单一,分工明确的三层架构模式

11.swift是面向对象的语言吗，是面向函数的语言吗

都是

12.你觉得自己做的产品跟竞品有什么缺点

优点是草根解说体育赛事,开启更多新玩法,用于与主播互动更紧密

缺点:1.平台注重免费体育赛事,一些顶级流量赛事资源不播,这就缺少了一部分核心竞争力

2.项目有些功能关联度不高,比如社区 用户与关注主播动态互动性不高

3.宣传力度不够

13.能够用钉钉或者企业微信，为什么你们还要开发自己的Kim呢

14.了解适配器模式和装饰器模式吗？

适配器和装饰器模式都是结构性设计模式,适配器是一种补偿设计模式,对有缺陷的接口,老版本接口进行兼容

装饰器设计模式是给对象额外的增加一些职责,利用各种组合装饰小对象,让一个对象具有更加灵活的功能

15.求 1+2+...+n ，要求不能使用乘除法、for、while、if、else、switch、case等关键字及条件判断语句

func sumNums(\_ n: Int, \_ res: inout Int ) -> Int {

\_ = n > 0 && sumNums(n - 1, & res)

res = res + n

return res

}

int result = 0;

public int sumNums(int n) {

if(n < 1) return result;

result += n--;

sumNums(n);

return result;

}

16.get请求和post请求的区别

参考音娱时光 第一题

另一个部门

1.二叉树根节点到叶子节点的所有路径和

深度优先搜索

2.热启动和冷启动的流程

热启动:

点击应用,找到保存的对应的进程,恢复应用

冷启动:

点击屏幕,springboard捕捉手势事件,创建进程,启动应用,应用启动阶段,加载可执行文件,加载链接器,加载其他动态库,编辑生成对应可执行文件,连接器链接可执行文件,解析器解析语法词组,接着调用runtime加载类,分类等文件,资源加载完之后,到main函数,创建主线程runloop, 调用appdelegate 代理方法,最后到rootVC首屏渲染完成

3.32位和64位ipa包的区别

32位IPA包是基于arm7s | arm7 架构,这两个架构,对于类型的所在空间也是不同的,比如指针所占4个字节

64位IPA包是基于arm64 架构,对于类型的所在空间也是不同的,比如指针所占8个字节

这个包由于内部内存分配的不同,对于方法查找,调用的起始地址也是不同的

64位IPA包可能会比32位IPA包大,因为64位IPA包可能含有64,32指令集

4.图文渲染怎么做的

>图像渲染

>>application 得到图元

进过CPU对图像处理生成图元

>>几何阶段,处理图元

顶点着色器:这个阶段中会将图元中的顶点信息进行视角转换、添加光照信息、增加纹理等操作

形状装配:图元中的三角形、线段、点分别对应三个 Vertex、两个 Vertex、一个 Vertex。这个阶段会将 Vertex 连接成相对应的形状

几何着色器:额外添加额外的Vertex，将原始图元转换成新图元，以构建一个不一样的模型。简单来说就是基于通过三角形、线段和点构建更复杂的几何图形

>> **Rasterization** 光栅化**,**图元转像素

图元转化成像素**,**以像素中心点判断像素是否在图元内

**>>Pixel** 像素处理阶段：处理像素，得到位图

像素填充颜色,像素点的集合就是位图

片段着色器:也叫做 Pixel Shader，这个阶段的目的是给每一个像素 Pixel 赋予正确的颜色。颜色的来源就是之前得到的顶点、纹理、光照等信息。由于需要处理纹理、光照等复杂信息，所以 这通常是整个系统的性能瓶颈

测试与混合（Tests and Blending）：也叫做 Merging 阶段，这个阶段主要处理片段的前后位置以及透明度。这个阶段会检测各个着色片段的深度值 z 坐标，从而判断片段的前后位置，以及 是否应该被舍弃。同时也会计算相应的透明度 alpha 值，从而进行片段的混合，得到最终的颜色。

文字渲染

https://juejin.cn/post/6844904162765832206#heading-14

UILabel 通过drawrect: 利用core graphics 进行渲染得到位图

参考coretext 绘制原理

CAtextLayer 原理

https://www.kancloud.cn/manual/ios/97791

5.别的app怎么打开我们自己的app

在自己项目中的 info中配置 URL type的 URL scheme, 拉起项目的APP中的canOpenUrl中配置对应的APP ,通过openURL打开对应的APP,在对应的APP中delegate openURL 代理方法 获取到对应的url

利用通用链接 利用web 拉起对应的APP

6.正弦函数的动画怎么做

https://zhouyangyng.github.io/2017/11/11/iOS%E6%B3%A2%E6%B5%AA%E5%8A%A8%E7%94%BB/

**y=Asin(ωx+φ)+b**

7.ARC release是在什么时候插入的

编译器 在对象生命周期结束的时候自动插入release 例如临时变量 出作用于时 实例变量躲在的

百度

1.swift struct 和 class的区别，struct的方法能改变属性的值吗？我一定要修改怎么办

struct存在栈上， class存在堆上，前者是值引用，后者是类型引用

一定要修改struct里的参数mutating

2.swift 枚举的特点

3.func plus(\_ num: inout Int) {

num + 1

}

num = plus(&num) 有问题吗

有

4.swift的5种访问修饰符（open，public，internal，fileprivate，private）

常用的是哪几种 internal private

5.swift optionnal（可选类型）用过吗？好处？

可选，各种类型都支持nil

6.swift 的闭包逃逸

7.setNeedsLayout() 和layoutIfNeeded()之间的区别

layoutIfNeeded()是同步的，调用后立刻执行，但是setNeedsLayout是异步的，不会立刻执行，比如修改了某一个控件的height，会等着下一个刷新到来时高才会改变

参考音娱时光 15

8.layoutSubviews什么时候调用

参考音娱时光 12

9.KVO，KVC setValue: forkey:会触发KVO吗？

会

KVC 的setValue: forkey 底层会调用willchangevalueforkey didchangevalueforkey (重写这两个方法就能验证)

10.分类有load方法吗？有，调用几次？分类和类中load方法调用的顺序，如果分类中重写了类的方法，如果调用这么方法是怎么调用的，为什么，如果两个分类中都重写了某一个方法是怎么调用的。initialize子类和父类的调用顺序，只调用一次还是可以调用多次？

分类有load方法 ,调用一次,先调用类load,在调用分类的load,(优先遍历类中load方法,之后在遍历分类load).分类重写类的方法,只会调用分类的方法 分类重写方法会造成原有方法列表地址后移,新的方法插入地址列表前面,被优先调用 load方法走的是函数地址直接调用

如果两个分类都重写了某个方法,优先调用后编译的分类,因为重写的方法会被插入到最前面

initialize 走的是消息转发,类第一次加载时会调用initialize,子类优先调用,子类没写,调用父类 子类调用一次,父类可能调用多次, initialize也是与编译顺序有关,编译晚的优先调用

1.调用方式 2.调用时刻 3.调用顺序

\*<https://www.jianshu.com/p/5e5a26c1ae15>\*

11.分类的底层实现，分类怎么添加实例变量吗？为什么可以添加

分类底层是category\_t结构体对象,内部有实例,类方法列表,属性,协议列表,在运行时会将分类的数据合并到本类

分类不能添加成员变量,但是分类借助关联对象,添加实例变量功能

12.讲一下runloop

参考网易问题3

13.swift的基本数据类型和OC的基本数据类型区别是什么（说是会遵守什么协议）

14.讲一下swift的内存管理，他跟OC有什么区别

15.setValue forkey 和setObject forkey的区别，setValue value可以为空吗

value forkey value 可为空，key是字符串， object forkey object不可为空，key可以是遵守NSCopying协议的任意类型

\*<https://www.jianshu.com/p/0860454e4c8e>\*

https://blog.csdn.net/Mo\_Mo123/article/details/64905947

setValue:forKey:是KVC提供的方法，可对绝大部分对象进行操作（包括自定义对象），-setObject: forKey是NSMutableDictionary提供的方法，只能操作可变字典；

-setValue:forKey:当value为nil时，对于字典相当于删除key－value对，-setObject: forKey当object为nil时将直接导致程序crash，因此调用之前需要判空；

16.NSDictionary的底层实现，底层是hash表，那什么时候会触发isEqual呢，hash表中对象存的是指针，但是如果存的是NSString，超过32位怎么存的？

\*<https://juejin.cn/post/6844903608954126344>\*

\*<https://juejin.cn/post/6844903678214668295>\*

http://xuaninitial.com/2019/12/12/NSDictionary%E5%BA%95%E5%B1%82%E5%AE%9E%E7%8E%B0/

底层有两个keys和values指针数组, 获取到的key通过hash函数获取到hash值,在通过hash值对keys 或者 values数组长度取模,得到索引index, 就能找到对应的key和value了,前提是keys和values数组长度一致,即使扩容,长度也是一致的

字符串太大,会取部分的hash值,比如 头,中间,尾部 部分字符串的hash值,如果相同,就判定为是同一个字符串

17.String swift和OC的区别

\*<https://blog.csdn.net/amberoot/article/details/88847596>\*

18.VC的生命周期，view的生命周期

\*<https://www.jianshu.com/p/8ac0ebf7b125>\*

VC initialize init loadview viewdidload viewwillappear viewwilllayoutsubviews viewdidlayoutsubviews viewdidappear viewwilldisappear viewdiddisappear viewwillunload viewdidunload dealloc

View init willmovetosuperview didmovetosuperview willmovetowindow didmovetowindow dealloc

19.用过webkit吗

https://www.jianshu.com/p/51b8301e7ea4

https://juejin.cn/post/6850418117500370958

20.讲一个RN的底层实现

21.针对某一种崩溃你怎么解决（这个没有get到他的点）

22.APP的性能优化你怎么做的

在appDidFinishLanch做的，把一些不必要的初始化放在后面

https://juejin.cn/post/6844904131941892110#heading-49

23.讲一下循环引用的\_\_weak和\_\_strong是为了解决什么问题的

\_\_weak和\_\_strong 是为了防止对象使用时,对象在其他地方被释放,造成 野指针的情况,调用对应的方法可能crash

24.青蛙跳台阶

斐波那契数列

25.怎么判断一个链表有环

运用快慢指针

26.怎么判断一个点在三角形上

海伦公式 p = (a + b + c) / 2;

S = 开根号(p(p - a)(p - b)(p - c))

比较这个点与三角形连接 三个三角形的面积与 原有三角形面积的大小

计算面积

27.现在有个项目需要启动你会怎么做

配合别人 还是负责 需求分析 项目拆分 需求分工 估时 第一版

28.你感觉的swift相对于OC的好处是？

29.swift和OC方法调用的区别

OC调用方法会走消息传递和消息转发，但是swift的话 它有有三种形式，直接派发，函数表派发和消息机制派发

30.卡顿和crash捕获到的堆栈信息 ，他是怎么实现的

卡顿通过借助CAdisplayLink 获取每秒打点多少次 就能得到FPS帧率(利用timestamp与当前帧的时间差,并累计每次调用次数,最后次数/时间就得到FPS) 可以借助runloop 通过while循环记录beforesource 和 afterwaiting 的次数来判断卡顿情况

crash

https://opensource.sensorsdata.cn/opensource/%E5%A6%82%E4%BD%95%E6%89%BE%E5%88%B0%E7%A8%8B%E5%BA%8F%E5%B4%A9%E6%BA%83%E7%9A%84-%E5%87%B6%E6%89%8B-%EF%BC%9F/ 借鉴神策异常处理方案

exception

通过NSSetUncaughtExceptionHandler 注册处理函数,在处理函数中通过exception 的callStackSymbols 拿到异常堆栈 ,进行上报

signal

设置signal 类型 添加捕获函数,捕获函数中将异常数据自定义为exception,调用exception流程

美团

1.属性的默认修饰符有哪些

参考网易问题1

2.weak的底层实现原理，一个对象被释放后，如果没有弱指针引用它还会走hash表查吗？应该不会，那怎么知道的没有没有弱指针指向呢？

参考网易问题1

不走了

ISA指针中ISA\_BITFIELD weak\_reference 标志位,可以知道当前对象用有没有弱引用指向

看源码就可以了

3.TCP的三次握手，为什么不是两次或是四次呢？TCP的拥塞机制？

https://www.jianshu.com/p/f0522796bda6

两次握手 可能 出现已经失效的请求 传给服务器,服务器打开了连接 而客户端已经启用超时的请求,就会忽略掉服务端建立的链接

四次握手 浪费资源,服务端SYN ACK 能够在同一个连接中发送,却分成了两次,造成时间和资源的浪费

拥塞控制:TCP让每一个发送方根据所感知到的网络拥塞程度来限制其能向连接发送流量的速率。

利用超时和冗余ACK 来判定网络拥塞

利用慢启动和拥塞避免来 限制连接发送流量的速率

https://www.jianshu.com/p/b9b5c1091d70

4.关于数据库，什么是事务？如果数据量特别的大，怎么提高查找效率？用索引，索引的优缺点？

这种把多条语句作为一个整体进行操作的功能，被称为数据库事务

https://www.liaoxuefeng.com/wiki/1177760294764384/1179611198786848

事务:逻辑单元执行一系列的操作

https://www.liaoxuefeng.com/wiki/1177760294764384/1179611198786848

合理建立索引

https://www.jianshu.com/p/3fc000589e2e

索引:提高了查询速率 降低了update 和insert 效率

5.什么样的block引用self的时候需要外面用\_\_weak修饰，里面用\_\_block修饰，如果block里面有一个self强引用tableview, 如果block里面用\_tableview 会引起循环引用吗

block被强引用的情况下,捕获实例变量,也会造成循环引用

6.你用过MRC吗？ARC的引用计数retainCount记录在哪？什么时候retainCount + 1，什么时候-1

手动引用计数

需要看isa指针中nonpointer 是否开启指针优化

开启: 引用计数存在isa指针的extra\_rc 中

没开启:引用计数存在SideTable 的refcnts引用计数表中

调用retain时,会使retaincount + 1 调用 alloc new copy strong对象指针引用

调用release 会使retaincount - 1 ,引用对象生命结束(出作用域,被释放)

7.讲一下runloop

网易问题3

8.自动释放池什么时候销毁，项目中用过自动释放池干什么，autoreleasePool底层实现原理？

网易问题3(14) 网易问题4

>>14.autoreleasepool与runloop关系

runloop创建时(CFRunLoopObserverCreate),callout参数函数设置为\_wrapRunLoopWithAutoreleasePoolHandler,在runloop的两个observe中

entry:order设置为最高序列-2147483647, \_wrapRunLoopWithAutoreleasePoolHandler 内部调用releasepoolpush

beforetime | exit : \_wrapRunLoopWithAutoreleasePoolHandler:BeforeWaiting(准备进入休眠) 时调用\_objc\_autoreleasePoolPop() 和 \_objc\_autoreleasePoolPush() 释放旧的池并创建新池；Exit(即将退出Loop) 时调用 \_objc\_autoreleasePoolPop() 来释放自动释放池。这个 Observer 的 order 是 2147483647，优先级最低，保证其释放池子发生在其他所有回调之后

9.说一下APP从一启动到加载出界面的这个过程

字节问题2

10.动态库和静态库的区别

静态库 是.o文件的合集 是跟随主工程合并到一起

动态库 是.o文件合集生成可执行文件 与主工程分开

静态库: 链接时完整地拷贝至可执行文件中, 被多个依赖多次使用就会有多份冗余拷贝.

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.(这个优点是针对系统动态库来说的, 比如 UIKit.framework)

https://zhuanlan.zhihu.com/p/346683326

https://freelf.me/%E5%88%9D%E8%AF%86iOS%E4%B8%AD%E7%9A%84%E5%8A%A8%E6%80%81%E5%BA%93%E5%92%8C%E9%9D%99%E6%80%81%E5%BA%93

11.timer的循环引用

Timer对target 内部是强引用 如果对timer强引用就会造成循环引用

timer的循环引用

scheduledTimerWithTimeInterval:block 在block内部将弱引用self dealloc内释放timer

手动释放

https://juejin.cn/post/6844903968250789896

12.项目中遇到的难点，怎么解决的

13.主要负责了项目中的哪些模块（具体说说）

14.App优化是怎么做的

百度问题22

15.用到了flutter和RN，说一下他们的区别

16.分类的方法调用

百度问题10

17.消息传递的流程

底部 自己总结问题1

18.介绍一下响应链

音娱时光问题20

19. cocoapods用过吗，install和update的区别

https://segmentfault.com/a/1190000015785680

如果没有podfile.lock文件，install和update是没有区别的

主要针对没有规定tag和commit的一些第三方，update会把这些第三方更新到最新，但是install会安装podfile.lock里面缓存的版本

20.找两个链表的交点，如果有的话

Leedcode 160

```

func getIntersectionNode(\_ headA: ListNode?, \_ headB: ListNode?) -> ListNode? {

if headA == nil && headB == nil {

return nil

}

var p = headA

var q = headB

while p !== q {

p = p == nil ? headB : p?.next

q = q == nil ? headA : q?.next

}

return p

}

```

21.删除链表的第k项

```

func deleteKthList(\_ node: ListNode?, \_ k: Int) -> ListNode? {

var first = node

for \_ in 1..<k-1 {

first = first?.next

}

if first == nil {

return node

}

first?.next = first?.next?.next

return node

}

```

22.用过多线程吗？GCD和NSOperation的区别，NSOperation底层是怎么设置依赖关系的

\*<https://www.jianshu.com/p/00fb6fae0dc0>\*

https://cloud.tencent.com/developer/article/1692252

• GCD为C语言API；NSOperation为OC语言，且面向对象。因此GCD相比执行速度更快些。

• GCD队列为FIFO，且设置任务之间的依赖比较复杂（可以使用信号量等方法）；NSOperation的队列可以设置任务的设置依赖，执行优先级等。

• GCD设置最大并发数比较复杂；而NSOperation可以很方便设置。

• NSOperation队列可以取消、暂停、恢复。

• NSOperation支持KVO，可以监测任务是否在执行、是否结束、是否取消等。

CGD:原有的API是没有这个方法的,只有线程未执行时的dispatch\_resume 和 dispatch\_suspend ,这两个函数对已经执行的代码不起作用,可以通过自己定义一个全部变量flag,在想要暂停该线程是,设置flag为YES,并在要被暂停的线程中做判断,执行return操作

NSOperation: suspended只能暂停当前线程的下一个操作,不能暂定已经执行的当前线程

跨队列依赖能不能生效

依赖关系不局限于相同queue中的operation对象，NSOperation对象会管理自己的依赖，因此完全可以对不同queue中的operation对象设置依赖。

GCD如何获取所处的队列

 dispatch\_queue\_t currentQueue = dispatch\_get\_current\_queue();

OperationCancel实现原理

https://www.cnblogs.com/fengmin/p/6108165.html

• addOperation:(NSOperation \*)op 方法中会判断 op 是否存在 且 没有结束，此时添加self 为观察者观察 “isReady” 状态，并将op加到一个内部的队列中（internal->operation）, 查看isReady状态为YES，通知观察者“isReady”状态已改变；

• 观察者 判断 ”isFinished” 状态就将该操作（op）从internal->operation 队列中移除，并更改operationCount、operations；观察者观察“isReady”状态就将该操作（op）加入internal->waiting队列中，等待执行，并最后调用\_excute方法，执行internal->waiting队列中操作。

• \_excute方法判断op（即self）没有被挂起 且 最大并发数max > internal->excuting中线程数 且 internal->waiting队列count 大于0 就 开始从internal->waiting队列中取操作执行，并观察“isFinished”属性。如果该操作（op）是并发的，就调用start方法开启线程；否则改操作就是串行的，将该操作（op）从internal->waiting队列中取出，添加到internal->starting队列中(internal-starting队列存储马上要执行的op)，创建新的线程去执行。

• NSOperation的start方法通过标识isAsynchous来判断是否需要创建一个新的线程，isAsynchrous为YES说明当前线程已经为异步线程，直接在该线程中执行操作，否则开启新线程执行该操作（op）。如果isCancel为YES，那么就不调用main方法直接才操作（op）。

Nsoperation start mian

https://nshipster.cn/nsoperation/

https://www.cnblogs.com/Keys/p/6170542.html

https://iandrew.space/2018-04-28/%E5%A4%9A%E7%BA%BF%E7%A8%8B-NSOperation,NSOperationQueue

https://www.jianshu.com/p/f1742d7cd5e5

Start 处理并发操作下,设置operation的各种status isready isexcuse isfinish 在各种status中手动设置KVO,作为operation的起点,即使走完start函数,isfinish还未置yes,说明在该方法中可以处理自己的逻辑并手动处理status

Main 处理非并发操作,现在和start一致了,main 作为operation的完成状态,晚于start,main函数执行完,会将operation从queue中移除

所以一般在start中处理自己的逻辑

Operation 实现KVO

https://www.quanguanzhou.top/post/41141.html

在start中处理自己逻辑,并手动给状态赋值,在状态的set方法中进行手动调用KVO,queue完成监听,进行对operation的移除操作

23.通过动态方法去调用一个方法和直接调用方法性能会差很多吗？

动态方法调用

Performselector objc\_msgsend IMP函数指针调用

和直接调用一致,底层走的是同一套流程

objc\_msg\_lookup

24.螺旋打印二维数字

leedcode 29

26.菲波那切青蛙跳台阶

百度24题

27.讲项目中遇到的难点

参考12题

唱吧

#define const static extern 区别，#define赋值是在什么时候发挥作用的

#define 宏定义 宏是由预处理器对宏进行替代,不会进行参数检查,仅仅只是对符号表进行替换

inline 而内联函数是通过编译器控制来实现的,会进行参数检查,就是函数中的执行代码不能太多了，如果，内联函数的函数体过大，一般的编译器会放弃内联方式，而采用普通的方式调用函数。这样，内联函数就和普通函数执行效率一样了

const 修饰符 所修饰对象或者指针不可更改

static 静态修饰符 变量存储在静态区

extern 对全局变量的引用,先在本文件中查找,找不到再去全局文件查找

https://www.jianshu.com/p/3b187a500d64

https://www.jianshu.com/p/123570248502

assign修饰NSString会有什么影响

可能造成野指针

NSString \*a = @"abc";

a 存在哪？@"abc"存在哪

栈 堆

button用weak修饰会怎么样

有可能会被释放 但不会有野指针

数据库里面的结构设计

https://codingdict.com/article/22299

OC中存数据的除了NSArray NSDictionary，还有什么

https://www.jianshu.com/p/140363987721

https://www.jianshu.com/p/140363987721

NSset

去重数组

**NSPointerArray** 指针数组**,**可以指定存储对象的引用方式**(**强引用 **||** 弱引用**),**我可以存入**nil,**只能通过**index** 操作对想**,**不能通过内部对象进行操作

**https://juejin.cn/post/6844904019006078984**

**NSHashTable** 和 **NSOrderedSet**

**https://zhangbuhuai.com/post/nshashtable-and-nsmaptable.html**

爱奇艺

OC和swift的区别

职级

1.侧重于讲项目

2.项目中遇到的比较大的挑战

3.你认为你的优势是

4.查看大图crash了，如果让你设计sdwebimage 你会怎么设计

https://juejin.cn/post/6844903597549813767

Crash是因为将大图全部加载进了内存,导致OOM

1.缩放磁盘图片比例进行压缩

2.切个成多个小图

3.修改库 并新增压缩方法

https://www.jianshu.com/p/9626ed7d1eab

5.sdwebimage的缓存策略，内存满了怎么删除的

下载图片 缓存到 内存 磁盘

http://cloverkim.com/SDWebImage-source-code-analysis-1.html

SDimagecache clearmemory —>SDMemoryCache —>removeAllObjects—>weakMapTable—>removeAllObjects

6.iOS如何加载超大图

将大图切割成多个小的位图 ,在利用CATiledLayer和drawLayer: incontext 分片加载和显示

https://zsisme.gitbooks.io/ios-/content/chapter6/catiledLayer.html

https://developer.apple.com/library/archive/samplecode/LargeImageDownsizing/Introduction/Intro.html

7.解决**SDWebImage**图片不同但**URL**相同时的图片缓存问题

**https://blog.csdn.net/q86486107/article/details/75039723**

百度贴吧

1.问flutter和SDK比较多（flutter的状态管理，flutter有什么能分享的经验，如果业务方和SDK方都引用了同一个第三方库怎么控制版本）

2.消息发送（从类里面找，类是什么，原类是什么，类里面都存了什么，原类里面存了什么）

参考自己总结问题

3.你感觉你比较擅长什么

4.你感觉老照片修复是怎么做的

5.数据库的底层实现是什么

参考唱吧问题

小红书

1.找到链表的倒数第k个节点

2.swift里\_\_weak 和 unownered 区别

3.讲一下autorelease

autorelease 会将修饰的对象放入当前runloop 中的自动释放池中

参考自己问题autoreleasepool 添加对象

4.weak的原理

网易问题1

5.categery为什么没法添加实例变量

categery底层是categery\_t结构体对象,结构体内不没有ivar\_list 变量,自由property\_list变量,所以只能添加属性,不能添加实例变量

6.swift怎么布局的，简单动画怎么做的

7.对一个类进行copy怎么做

https://www.jianshu.com/p/43007ee178d9

8.协议的extension里面的方法的应用场景， class extension 里面的private 的方法，class里面可以调用到吗

9.GCD 的信号量和dispatch\_group

dispatch\_semaphore\_t 线程阻塞 线程同步 资源访问

dispatch\_group 线程调度先后

10.你们用的RXswift，那你们的架构设计是？讲一下MVVM

贝壳

1.怎么判断图片的内存大，怎么算的内存（W \* H \* 4像素）/ 1024 \* 1024，图片在渲染的时候内存暴增OOM的原因是什么？下采样？

参考爱奇艺问题

2.内存管理（unsafe，unretain了解吗， \_\_weak, \_\_strong的底层实现是？）

参考网易问题1,3

3.runloop，开启常驻线程的场景（监控卡顿）

网易问题3

timer

子线程保活

4.线程池

https://www.jianshu.com/p/ddcd28e90da8

http://shevakuilin.com/ios-multithreading/

5.列表的优化

参考各种优化策略

6.讲一下flutter

7，讲一下某个app的代码结构

keep

1.冷启动的过程

参考字节问题2

2.CAlayer和UIVIew的关系，CALayer是通过什么来给UIVIew渲染的（代理）

layer不能相应点击事件,UIview作为layer层的代理,可以响应点击事件

CAlayer 是UIVIew 的属性,UIView展示的视图是由CAlayer进行绘制,并显示到屏幕上

UIVIew是CAlayer的代理,也可以说是CAlayer的框架和载体

CAlayer 主要执行图像绘制和显示

UIVIew 在承载CAlayer之外,还响应事件点击和处理

https://luochenxun.com/ios-calayer-overview/

3.离屏渲染是什么

音娱时光 17

4.链表的交点

美团20题

5.事件响应的流程，手势事件传递

音娱时光 20

6.

重点讲项目中遇到的问题，做了哪些东西

7.仿写一个sdwebimage 会考虑哪些事情

缓存

同一url的多次下载

资源多线程访问

大图加载

代码结构

8.讲最近一家公司的所有项目

9.flutter和RN的区别

10.项目中遇到的难点（Kim）

11.讲他们做什么的

天眼查

1.消息发送和消息转发

参考自己发现问题

2.冷启动的过程

参考字节问题2

3.做过哪些优化

4.内存管理

参考网易问题1,2

5.自动释放池

参考网易问题4

6.runloop

参考网易问题3

7.

崩溃日志和卡顿监控怎么做的

8.rn和flutter的区别，rn的瓶颈在哪

9.分类的实现

百度问题11

10.

算法接雨水

11.

讲项目

12.

遇到的难点怎么解决的，遇到的挑战

13.

算法：两个数组合并，去掉重复的元素，并排序

14.

判断二叉树是否对称

完美世界

@autoreleasepool {

a = @{};

}

NSLog(@"a+++++++++++: %@", a);

id b = nil;

@autoreleasepool {

b = @{};

}

NSLog(@"b+++++++++++%@:", b);

self.c = nil;

@autoreleasepool {

self.c = @{};

}

NSLog(@"1c+++++++++++%@:", self.c);

self.c = nil;

[self b];

NSLog(@"2c+++++++++++%@:", self.c);

滴滴

1.

讲你在快手两年多做的东西（一直讲）

2.

根据项目问图片优化

3.

常用的设计模式

结束

另一个部门

反转链表（自己没搞明白）

跟谁学

1.自动释放池

参考网易问题4

2.属性修饰符

参考网易问题1

3.

锁

4.

GCD

5.

讲项目，

6.

找下一份工作关注的三个点

7.

在线教育和线下教育的区别

8.

讲他们的优势

猿辅导

1.

两个字符串相加，只会是正整数

2.

如果实现从0-14依次打印，需要开启三个线程，GCD

3.属性修饰符

网易问题1

4.ARC内存管理

网易问题2

5.

重构选swift的原因

6.

flutter的底层实现

7.

RN和flutter的区别

8.

放弃RN的原因

第四范式（外包）

1.

最小字母的删除（写完问是否可优化，怎么优化）

2.自动释放池的原理

参考网易问题4

3.UI事件传递和响应链

音娱时光20

4.OC内存管理

参考网易问题2

5.属性修饰符

参考网易问题1

6.

讲项目，

7.

离职原因

8.

做了哪些优化

9.

崩溃怎么上传的

10.

怎么查看崩溃

11.

对外包有了解吗

其他

git rebase 和 git merge 的区别

RXSwift的底层实现

写rn遇到的坑

后面如果rn和flutter

一面远程视频面, 套路很正常，自我介绍，iOS理解，Swift理解，算法，你问我。

1.简单介绍一下你自己；

2. UIButton，是怎么得到点击的    答：event，runloop，UIResponder    可是具体UIResponder是怎么得到点击的呢？    答：不清楚    那我这么说，如果不改这个Button的frame，你怎么才能扩大它的接触范围？    答：不清楚    好了。是hittest。

3. class和struct的区别

4. 说一说iOS的内存管理

5. weak是怎么实现的

6. Array是value types，它有一个copy on write的行为，具体怎么实现的？   答：那当有修改时，才会copy，没有修改，就只是引用   那它具体是怎么判断该不该copy呢？   答：按占用内存大小？   那具体是怎么实现的呢？   答：不清楚   好了。Array是一个struct，这个struct里面维护了容量，reference，当修改的时候，通过references的   引用数量，就可以知道是不是该copy。

7. 两个view，怎么找到它们的共同的父view？最近的那个。   答： 要用superview，也就是要按照链表来理解。 （用到了Set）UIView怎么放进Set呢？   答：description？Hashable？不清楚它是不是Hashable。   哦，NSObject都有hash。

8.你有什么问题要问我的？

（用到了Set）你怎么确定UIView可以被放进Set呢？

——————————————————

自己总结的问题

runtime

0.objc\_class 内部构造 1

Isa : 当前对象所属的类对象

Superclass: 当前对象父类

Cache : \_buckets列表 bucket:sel IMP

bits : 方法列表,属性,协议等数据合集 与FAST\_DATA\_MASK 做&操作,生成对应的数据class\_rw\_t

class\_rw\_t : 包含方法列表,协议,属性列表,以及class\_ro\_t对象

class\_ro\_t: 编译时确定的法列表,协议,成员变量列表

1.方法调用流程 1

https://www.jianshu.com/p/d628f1b9e51e

消息查找

objc\_msgsendsuper

id objc\_msgSendSuper(struct objc\_super \*super, SEL op, ...)

struct objc\_super {

id receiver;

Class cls; // the class to search

}

当前接受receiver对象 从 cls(父类)中开始查找对应的方法

objc\_msgsend

查找流程

1.obj->isa对应的类对象中缓存cache\_列表中查找,找到,直接调用

2.缓存找不到,去方法列表中二分查找,找到后调用 并 缓存到cache列表中

3.当前类对象中找不到,去父类缓存中找,找到直接调用

4.父类缓存找不到,找父类方法列表,若找到直接调用 并缓存到objc->isa对应的类缓存里 父类缓存不存储

5.父类还是找不到,依次执行3,4,流程

6.还是找不到,进入动态解析,消息转发流程

动态解析

resolveInstancemethod

resolveClassMethod

返回yes 代表解析成功

消息转发

https://www.jianshu.com/p/198f031f44ea

forwardingTargetForSelector

寻找能处理该sel的对象即forwardingTarget

methodSingnatureForSelector

注意:将转发目标对象操作交给我们处理,需要指定对应的target,不够灵活

给该sel签名,为后续包装invocation做准备

forwardInvocation

注意:将转发操作完全交给我们处理,该处理最灵活

拿到包装好的invocation

自定义invocation taget 参数 等等

//转发失败

doesNotRecognizeSelector 报错

2.load,initialize 执行 1

https://juejin.cn/post/6844903602524274696

Load : 优先调用类的load 父类—>子类

initialize : callInitialize 走的消息转发

3.分类加载流程 1

https://juejin.cn/post/6844903602524274696

分类数据依次加进本类 列表中,原有数据地址后移,分类数据放在最前面,方法调用时,优先调用分类方法 memmove : 内存移动 memcpy:内存拷贝

4.执行顺序 \_objc\_init

map\_image 类的加载,分类的加载 初始化 数据合并

load\_images load方法加载流程

5.weak表 1

6.引用计数 1

7.自动释放池底层 1

ARC 模式下,自动释放对象才会被自动加进自动释放池,使用类方法（工厂方法）实例化的对象才是自动释放的对象，才能被添加到自动释放池，而使用new、alloc、copy关键字生成的对象和retain了的对象，不会被添加到自动释放池中

Autoreleasepoolpage

最大size 4096字节,自身参数占56字节 剩余 分配给自动释放池的对象地址

Page是双链表形式 每页page 存满(full)之后,会生成新的page

push

每一个自动释放池 都会对应一个pool\_boundary,并记录pool\_boundary对应内存地址 , push对象依次放在pool\_boundary地址后面

pop

倒序从page查找,找到对应的pool\_boundary,对涉及到的对象进行release

autorelease

将当前对象依次入栈

https://www.jianshu.com/p/2d9071465a73

autoreleasepool 与runlLoop关系

observeEntry时,会调用autoreleasepoolpush

observeWait 时,调用autoreleasepoolPop,autoreleasepoolpush

observeExit时,调用autoreleasepoolPop

8.对象释放过程 1

9.关联对象 1

https://juejin.cn/post/6844903605347057672

10.APP点击启动流程 1

https://juejin.cn/post/6844903968837992461

冷启动

热启动

读取对应的进程,直接唤醒

————————————————

LLVM clang dyld objc

IOKit —>进程—>应用

事件捕捉

电量

内核 BSD

自己的扩展

dyld 源码

符号表

**OOM**

iOS内核区

源码阅读顺序

LLVM ——clang —dyld—-objc4—gnu和变色龙

springboard —>进程—>应用程序

**sizeof**与**strlen**的区别

**sizeof** 是运算符编译前确定了空间大小

**strlen** 是函数运行时才能确定空间大小

**https://developer.aliyun.com/article/3770**

**4**个字节**int**取值范围是**-2^31** 到**2^31 - 1**

https://blog.csdn.net/qq\_20255275/article/details/99703848

计算机运算法:

正数:取原码  
负数:取补码

NSTimer GCDtime displaylink 硬件定时器

URI 与URL 的区别

https://www.zhihu.com/question/21950864

URI 是在「某一规则」下标识出一个资源的字符串，通过地址或者通过号码都是可行的规则，其中通过地址规则实现的 URI 可以被称作 URL ，URL 是 URI 的一种实现，所以URI 作为更宽泛的定义是包含了 URL 的，就像三角形包含等边三角形一样.

计算机图形渲染的流程

Arm64 汇编 fishhook

cocoapod 底层实现

OC代码存取到mach-o的\_\_DATA数据段文件中,替换load方法

http://dreamtracer.top/ji-yumacho-de-qi-dong-xiang-guan-li-2/

字典的key 值 只要是准守NSCoping 协议 的非空对象都可以

https://www.jianshu.com/p/66174593b836

GCD源码

AFN源码

————————————————

自我介绍

项目了解

直播间方案

如何优化

还存在的问题

组件化方案对比 各种组件化的差异性

组件化方案二进制

二进制方法了解哪些

组件化导致项目编译慢

监控系统级崩溃

如何处理crash

如何监控项目里的崩溃

哪些crash捕捉不到

你遇到过哪些崩溃

有没有写过一些预防崩溃的文件

你的技术栈

底层询问

runtime load 分类 编译顺序

set up objc

Initializer load加载

A分类 viewdidload 编译在前

B分类 + load{ 交换viewdidload 为xx\_viewdidlaod}

主类 调用viewdidload{}

在当前线程空闲的时候去调用一个任务

runloop

有哪些状态

每个状态处理的任务

循环引用理解

Block的循环引用造成的问题

如何解决

\_\_weak修饰之后 可能会出现的问题

在内部用\_\_strong修饰之后 可能会出现什么问题 还会出现循环引用吗 里面的self与外面的self是同一个self吗 指针地址一样吗

怎么去监控循环引用

facebook 的retainCycle 底层实现原理

方法查找流程

多线程 串行 并发

同步 异步

任务先后顺序

系统捕获异常的类或者方法

NSSetUncaughtExceptionHandler

@try @catch

自我介绍

项目中你觉得最难的点

为什么这样做，好处在哪

对公共组件的理解，如何下沉

基础

分类实现

KVO实现

自己实现一个通知

weak实现

内存管理的理解

Block的理解和变量捕获

runloop 的底层实现

子线程保活runloop

子线程发通知

同步异步的理解

算法

实现一个O（1）级别的字符串翻转，字符串有可能是N长度