在主线程中[NSThread exit]会杀掉主线程!但是APP不会挂掉

优先级priority只能保证CPU调度的可能性会高,并不是高的都执行完了再执行低的

@synchronized (self.lockObjc) { 加锁的代码 } // lockObjc是非空NSObject全局变量，耗费大量资源，不建议用

atomic原子性操作，为setter方法加锁，nonatomic不为setter方法加锁

dispatch\_barrier\_async：虽然是异步，在前面的任务执行结束后它才执行，它后面的任务等它执行完成后才会执行，queue不能是全局的并发队列

同步：一个线程中，异步：可以开启新的线程

dispatch\_get\_global\_queue获取全局的并发队列，无需手动创建

延时执行代码的3种方法: 1. performSelector: afterDelay: 2. dispatch\_after(异步) 3. [NSTimer scheduledTimer repeates:NO]

快速迭代遍历dispatch\_apply(10, queue, ^(size\_t index){ 执行10代码,index顺序不确定 })

队列组：等2个异步操作执行完后,回到主线程 dispatch\_group\_t group=dispatch\_group\_create(); dispatch\_group\_async(group, …); dispatch\_group\_notify(group, …)

UIScrollView().minimumZoomScale //UIScrollView的最小缩放属性 并实现viewForZoomingInScrollView

继续loop： [NSRunLoop currentRunLoop] runMode

dispatch\_queue\_create("tanzhouios", DISPATCH\_QUEUE\_CONCURRENT)队列中的线程(任务)并行，NULL的时候串行

NSOperation:

将NSOperation添加到NSOperationQueue中,系统会自动异步执行NSOperation中的操作

最大并发数: maxConcurrentOperationCount

依赖:[operation addDependency:operationA]; //A执行完后,再执行B。可以在不同queue的NSOperation之间创建依赖关系

执行完后的操作operation.completionBlock

死锁dispatch\_sync(dispatch\_get\_main\_queue(), ^{NSLog(@"…");}); //不会创建新的线程，于是sync等待主队列，主队列又等待sync

GCD与NSOperation：

GCD特有的：一次执行；延迟执行；调度组(op中实现有点麻烦)

Op封装的(GCD实现麻烦的)：最大并发线程；队列的暂停/继续；取消所有操作；制定操作之间的依赖

加密: 目前用的HMAC: 登录时，通过用户id获取key，key和明文密码加密。服务器端的密文有时间限制。

指纹识别：#import <LocalAuthentication/LocalAuthentication.h>

LAContext canEvaluatePolicy evaluatePolicy

解压SSZipArchive unzipFileAtPath

NSURLSessionDownloadTask() cancelByProducingResumeData得到的data是断点续传后增加的data，但length跟bytesWritten不一致?

当调用某个类没有的方法，都会进入+(BOOL)resolveInstanceMethod

归档: NSKeyedArchiver archiveRootObject 解档unarchiveObjectWithFile

一种取值方式[p valueForKey:@"\_age"]

归档有非id类型，比如int，最好这样做： if ([[NSString stringWithUTF8String:ivar\_getTypeEncoding(ivar)] isEqual: @"i"]){NSNumber \*value = (NSNumber \*)[self valueForKey:key]; [coder encodeInt:value.intValue forKey:key]; }

socket:

int clientSocket = socket(协议域AF\_INET, socket类型SOCK\_STREAM, 0)

struct sockaddr\_in serverAddr; serverAddr.sin\_family=AF\_INET;

serverAddr.sin\_port=htons(12345); serverAddr.sin\_addr.s\_addr=inet\_addr(“127.0.0.1”)

connect(clientSocket, (const struct sockaddr \*)&serverAddr, sizeof(serverAddr))

ssize\_t sendLen=send(clientSocket, msg.UTF8String, strlen(msg.UTF8String), 0)

uint8\_t buffer[1024]; recv(clientSocket, buffer, sizeof(buffer), 0);

instruments看内存泄漏: control+cmd+i打开instruments->allocations->在下面看某个类的persistent

property(strong): 赋值后指向的内存地址不变，引用计数加1。 weak指向的内存地址不变，引用计数不变。copy开辟了新内存，将内容赋值一份，并指向。A

assign同weak，指向内存且引用计数不变，但当引用计数为0时, assign不会进行数据抹除操作，只是值释放，形成野指针，没给新值之前调用它就可能崩溃。

ARC之前属性构造器的关键字是retain,copy,assign，strong和weak是ARC带出来的关键字。

将NSMutableArray: self.strongAtrr=arr，数组对其中的元素是强引用(浅拷贝)，strongArr又对arr强引用，所以每个元素的引用计数是3。当后来arr添加元素，只有数组对元素的强引用，所以新元素的引用计数是2

当property()后接不可变NSString：strong, copy都不会创建新对象，copy会将引用计数加1。 局部变量修改值后会创建新对象并指向，其他property修饰的地址和值都不会变。 原因：不可变量的值不会改变，既然都不会改变，所以没必要重新开辟一个内存出来让aCopyStr指向他，直接指向原来值位置就可以了

当NSString长度小于10时不再遵循引用计数规则，Tagged Pointer技术对其进行了优化。基本意思就是默认会将一些长度小于10的字符串直接保存在指针上面，下次创建相同值的时候直接用同一份拷贝，这样既减少了一次指针到值的访问，又减少了一份内存的占用

swift与OC的区别(虽然这个问题看起来SB, 但面试可能问到, 又太大了不好现场一点点想，所以记下来)：

swift: 1.元组 2. 类型安全的语言可选值as 3.类型别名:typealias AudioSample=UInt16 4.override

5.无主引用uowned(适用环境: 一个属性可以是nil，另外一个属性不允许是nil，并有可能产生强引用环)

7.新增了两种访问权限(open，public(其他module中不可以被override和继承

)，internal默认，fileprivate(同一个文件的不同类还是可以访问)，private)

8. Map、FlatMap(执行结果append: contentsOf:)、Filter、Reduce(用于累积运算)函数式编程

9.范围运算符: a…b a..<b 10. let var 11不再是非0就是真 12.switch可接字符串，不用break 13. 下标subscript