系统运维：

1. Jvm调优原则？

答案：多数的java应用不需要在服务器上进行GC调优，多数导致GC问题的java应用，都不是因为我们参数设置错误，而是代码问题，在应用上线之前，先考虑将机器的JVM参数设置到最优（最合适），减少创建对象的数量，减少使用全局变量和大对象，GC优化是到最后不得已才采用的手段，在实际应用中，分析GC情况优化代码比优化GC参数要多得多。

1. Jvm调优目的？

答案：将转移到老年代的对象数量降低到最小，减少GC的执行时间。

1. JVM怎么调优？

答案：1.打开PrintGC，查看gc日志（-XX：+PrintGC -XX：+PrintGCDetails（打印gc详情） -XX：+PrintGCTimeStamps（打印gc停顿耗时） -XX：+PrintGCDateStamps（打印GC的时候添加事件标志）），输出的log里头99%都是GC（Allocation Failure）造成的young gc，allocation failure表示向young generation（Eden）给新对象申请空间，但是young generation（Eden）区剩余的空间不够所需的大小，所以导致的minor gc，再看有没有发生full/major gc 2.检查gc日志和系统日志(也可以用GC日志可视化工具：GC easy：上传gc的log文件就可以， GCViewer) 3.调整堆大小，调整GC策略（如果不会调内存的话，可直接将-xms和-xmx调成一样即可），GC调优策略有：（1）将新对象预留在新生代，由于full gc的成本远高于minor gc,因此尽可能将对象分配在新生代是明智的做法，实际项目中根据GC日志分析新生代空间大小分配是否合理，适当通过“-xmn”命令调节新生代大小，最大限度降低新对象直接进入老年代的情况。（2）大对象进入老年代，虽然大部分情况下，将对象分配在新生代是合理的，但是对于大对象这种做法却值得商榷，大对象如果首次在新生代分配可能会出现空间不足导致很多年龄不够的小对象被分配在老年代，破坏新生代的对象结构，可能会出现频繁的full gc.因此，对于大对象，可以设置直接进入老年代（当然短命的大对象对于垃圾回收来说简直就是噩梦）。-XX：PretenureSizeThreshold 可以设置直接进入老年代的对象大小。（3）合理设置进入老年代对象的年龄，-XX：MaxTenuringThreshold设置对象进入老年代的年龄大小，减少老年代的内存占用，降低full gc发生的概率。（4）设置稳定的堆大小，堆大小设置有两个参数：-xms：初始化堆大小， -xmx：最大堆大小。（5）如果满足下面的指标，则一般不需要进行GC优化：minor gc执行时间不到50ms, minor gc执行不频繁，约10秒一次， full gc执行时间不到1秒，full gc执行频率不算频繁，不低于10分钟。

1. java常用排查工具？

答案：top:linux自带的命令，乐意查看系统资源消耗情况，可以看看CPU，内存，SWAP，IO的消耗情况，需要特别注意的有几个值：ni：这个值如果特别高，表示线程上下文切换开销比较大，看是不是开了太多的线程导致的。Res:表示了进程实际占用的内存，swap:内存不足就会占用swap空间，这个时候一般应用的性能会急剧降低，需要特别关注。jstack,jmap,jvisualvm：这三个是jvm 自带的一些工具，必须熟练掌握。

Jstack:jstack命令主要用于生成虚拟机当前时刻的线程快照。线程快照就是当前虚拟机内每一条线程正在执行的方法堆栈的集合，生成快照的目的主要是定位线程长时间出现停顿的原因，如线程死锁，死循环，请求外部资源导致的长时间等待等都是导致线程长时间停顿的原因。线程出现停顿的时候通过jstack来查看各个线程的调用堆栈，就可以知道没有响应的线程到底在后台做些什么事情，或者在等待些什么资源。通过jps得到代码的id之后可以用jstack id(刚刚得到的)。（2）jmap:命令用于生成堆转储快照，jmap的作用并不仅仅是为了获取dump文件，他还可以查询finalizer执行队列，java堆和永久代的详细信息，如空间使用率，当前使用的是哪种收集器等，将指定的应用程序的快照输出到桌面，然后，可以通过jhat，visual VM等工具分析该堆文件，看看有哪些大对象，或者哪些类的实例特别多。Jhat分析headdump文件：他会建立一个HTTP/HTML服务器，让用户可以在浏览器上查看分析结果。访问<http://localhost:7000/> （3）jvisualvm:提供在java虚拟机上运行的java应用程序的详细信息，在visualvm的图形用户界面中，您可以很方便、快捷地查看多个java应用程序的相关信息。功能有：1.显示虚拟机进程以及进程的配置、环境信息（类似jps、jinfo）。2.监视应用程序的CPU、GC、堆、方法区及线程信息（jstat、jstack）。3.dump生成以及分析堆转储快照（jmap、jhat）。4.方法级的程序运行性能分析，找到被调用最多，运行时间最长的方法。5.离线程序快照：收集程序的运行时配置，线程dump、内存dump等信息建立一个快照，可以将快照发送开发者出进行bug反馈。

还有jps，jinfo、jstat，jps：类似于Unix的ps命令，用户查看所有的java进程的启动类，传入参数和java虚拟机参数等信息。显示虚拟机执行主类名称以及这些进程的本地虚拟机唯一ID。Jps –q:只输出进程的本地虚拟机唯一ID，jps –l:输出主类的全名，如果进程执行的是jar包，输出jar路径。Jps –v:输出虚拟机进程启动时JVM参数。 Jps –m:输出传递给java进程的main()函数的参数。Jinfo:实时查看和调整虚拟机各项参数。Jstat:监视虚拟机各种运行状态信息，他可以显示本地或者远程虚拟机的类信息，内存、垃圾收集、JIT编译等运行数据，在没有GUI，只是纯文本控制台环境的服务器上，他将是运行期间定位虚拟机性能问题的首选工具。

1. 系统日志中找出error日志

答案：grep命令，grep命令是一款强大的文本搜索工具，他能使用特定模式匹配（包括正则表达式）搜索文本，并默认输出匹配行，命令格式：grep string filename是从filename中找string，这里从系统日志文件中找error日志：grep error \*\*\*.log，例：find . –name “.log” | grep –i error | grep –vi “info” 是使用find –name来列出所有log文件，然后重定向给grep，然后使用grep –i error来查找所有包含error的行，使用grep –vi来查找不包含info的行。grep过滤多个条件是使用grep –E命令，如grep –E ‘113|abc’ filename，找出文件filename中包含123或者abc的行，中间用“|”管道符分开。

1. 系统空间不够，如何找出大日志文件？

答案：（1）linux根目录满了，使用du –sh \*来查找跟目录下每个文件夹的大小，然后进入空间占用比较大的文件夹，然后再使用du –sh \* | sort -n命令查找大文件，sort还可以让文件夹下的文件按大小排序。（2）find / –type f –size + 100M 查找大于指定大小的文件， 文件名选项是find命令最常用的选项，find ~ -name “\*.txt” –print,不管当前路径，在自己的根目录home下查找文件名符合“\*.txt”的文件，使用~代表home目录。当前目录及子目录用.代表。

项目实践：

1. 线上出现CPU满载，内存OOM如何处理，如何解决？

答案：首先通过top命令查看耗费CPU的进程号，举例3997，然后可以通过top –H –p 3997去查看持续耗费CPU的线程号是多少，然后可以通过jstack 3997命令打印出当前时刻虚拟机的线程快照，就可以看到各个线程的调用堆栈信息，通过刚刚得到的线程号，可以定位到长时间停顿的线程在后台到底在做什么。Jmap能将指定的应用程序的堆转储到桌面（dump文件），然后通过jhat分析，有没有大文件或者是哪些类的实例比较多等等，然后分析代码哪些对象不应该出现过大或者实例过多，再综合jvm，GC参数，GC日志，调整堆大小，调整GC策略等，避免完整内存加载。

1. 新技术做技术选型？

答案：业务，开源，复杂度，部署资源，社区活跃，语言（平台），文档。

1. 数据库被击穿，如何处理？

答案：缓存击穿，是在高并发系统中，大量的请求同时查询一个key(热点key)时，此时这个key正好失效了，就会导致大量的请求都打到数据库上面，使数据库压力过大，甚至使数据库服务被压死，这种现象称为缓存击穿。1.分库分表：要考虑错开热点数据 2.错开热点数据：使用cache-hash算法，做缓存的随机均匀分布，错开热点数据集中在一个单点服务器上。 3.外、内限流控制：常见的限流方法有滑动窗口，令牌桶算法，和漏桶算法。或者直接使用队列、加锁等来做限流。 4.一、二级缓存，热点缓存：一级缓存失效时间短，二级缓存失效时间长，请求优先从一级缓存获取数据，如果一级缓存未命中则加锁，只有1个线程获取到锁，这个线程再从数据库中读取数据并将数据再更新到一级和二级缓存中，而其他线程依旧从二级缓存获取数据并返回。这种方式，主要是通过避免缓存同时失效并结合锁机制实现，当数据更新时，只能淘汰一级缓存。二级缓存中可能会存在脏数据，需要业务能够容忍这种短时间的不一致。 5.缓存更新策略、过期策略：按照缓存过期时间后台定时刷新数据到缓存中。或者是将缓存key的过期时间（绝对时间）同数据一起保存到缓存中去，然后看过期时间减去当前时间是否小于一定的值，若小于，则主动更新缓存，这两个方案都能保证缓存中的数据始终是最新的。6.可以用互斥锁来实现，因为是多个线程同时去查询数据库的这一条数据造成的，所以可以在第一个查询数据的请求上使用一个互斥锁。（单机用synchronized，lock，分布式环境用Redis分布式锁（setnx），zookeeper分布式锁等）

1. 100万政务数据导出，如何处理？

答案：数据量少的话都用POI导出数据，数据量大的话：1.分段、分服务器：针对海量数据，在mysql执行批量插入时，mysql客户端默认是16M，服务器端是4M，并且还要控制io流，以防内存溢出，所以需要对数据进行分批分段处理。 2.邮件异步，页面异步，数据预热等响应：使用线程池异步导出，自定义ThreadPoolExecutor和BlockingQueue队列来结合处理，需要的子线程数量多，但数量不确定，子线程有自己的优先级，根据优先级来确定执行的先后顺序，监听线程池的开始，结束，关闭等状态， 3.考虑数据脱敏，水印等（安全）：常见的脱敏处理方法有：替换，：把某一个属性统一替换掉，重排，：按照一定顺序打乱，加密，：用某种加密算法，截断，：舍弃必要信息来保证数据的模糊性，掩码，：123456->1\*\*\*\*6保留了部分信息，并且保证了信息的长度不变性，如火车票的身份信息，日期偏移取整：舍弃精度来保证原始数据的安全性，一般此种方法可以保护数据的时间分布密度。

1. 项目新模块需要开发紧急上线，如何处理？

答案：人员（需求、测试、研发） 时间 技术选型（尽快上线，后期优化）。

JVM相关：

1. 字节码和class文件是什么？dex文件是什么？和class文件的区别是什么？

答案：在java中，JVM可以理解的代码就叫做字节码（扩展名为.class的文件），他不面向任何特定的处理器，只面向虚拟机，dex文件能够被DVM或者Art虚拟机执行并且加载的文件格式，dex文件是Android系统的可执行文件，包含应用程序的全部操作指令以及运行时数据，dex文件是应用安装时生成的虚拟机可执行二进制文件，如果应用还存在，删除了下次手机开机时还会再次生成，卸载软件时会同时删除dex文件；区别：本质上是一样的，都是二进制流文件格式，dex文件是从class文件演变而来的，class文件存在冗余信息，dex文件则去掉了冗余，并把所有的class文件整合到dex文件中，减少了IO操作，提高了类的查找速度。Dvm是基于寄存器的虚拟机，而jvm是基于虚拟机栈的虚拟机，寄存器存取速度比栈快得多，dvm可以根据硬件实现最大的优化，比较适合移动设备。

1. Jvm运行时的数据区有哪几部分？java对象实例主要的存放区域是哪一块？有什么特征？（GC回收，分代算法 新生代和老年代 逻辑连续OOM等）

答案：有堆，方法区（元空间），虚拟机栈，本地方法栈，程序计数器（唯一一个不会出现outofmemory的内存区域），直接内存6个部分。堆内存区域存放java对象实例，几乎所有的对象实例以及数组都在这里分配内存，但jdk1.8之后如果某些方法中的对象引用没有被返回或未被外面使用，那么对象可以直接在栈上分配内存。Java堆也是垃圾收集器管理的主要区域，因此有称为GC堆，细分为新生代内存（Eden空间，From Survivor，To Survivor为了更好的回收内存，或者更快的分配内存），老生代内存，永久代内存（jdk1.8之后叫元空间，使用直接内存，与永久代不同的是，如果不指定大小的话，随着更多的类的创建，虚拟机会耗尽所有可用的系统内存，整个永久代有一个JVM本身设置的固定大小上限，无法进行调整，而元空间使用的是直接内存，受本机可用内存的限制，虽然元空间仍旧可能溢出，但是比原来出现的几率更小），可以通过-Xms和-Xmx来调整堆内存的大小，可以用JConsole查看堆内存的使用情况，可以使用jmap来获得heap dump，用jhat来分析heap dump，栈空间是用来存储调用栈和局部变量的，java垃圾回收器是用来将死掉的对象所占用的内存回收回来，在释放到java堆空间中。

1. 何谓双亲委派模型？java虚拟机的类加载器有哪几类？

答案：如果一个类加载器收到了类加载请求，系统会首先判断当前类是否被加载过们已经被加载过的类会直接返回，否则才会尝试加载，加载的时候，他并不会自己先去加载，而是把这个请求委托给父类的加载器去执行，如果父类加载器还存在其父类加载器，则进一步向上委托，直到到达顶层的启动类加载器（BootstrapClassLoader），如果父类加载器可以完成类加载任务，就成功返回，倘若父类加载器无法完成此加载任务，子加载器才会尝试自己去加载，这就是双亲委派模型。Java虚拟机中有三个类加载器：启动类加载器（BootstrapClassLoader：加载java\_home/lib下的jar包和类），扩展类加载器（ExtensionClassLoader：加载java\_home/lib/ext下的jar包和类），应用程序类加载器(AppClassLoader：加载当前classpath下的jar包和类)

1. Java对象的四种引用？java对象的生命周期？java中如何判断对象是否存活？（根搜索GCRoots 引用链）？

答案：强引用：必不可少的东西，垃圾回收器绝不会回收他，当内存空间不足，宁愿跑出outofmemory也不会回收具有强引用的对象、软引用：可有可无的生活用品，如果内存空间够，就不会回收他，如果内存不够了，就会回收这些软引用的对象，软引用可用来实现内存敏感的告诉缓存，可以和一个引用队列联合使用，如果软引用所引用的对象被垃圾回收，java虚拟机可以将这个软引用加入到与之关联的引用队列中，软引用的优点：可以加速JVM对垃圾内存的回收速度，可以维护系统的运行安全，防止内存溢出等问题的产生、弱引用：可有可无的对象，只要垃圾回收器发现它，不管内存够不够，都会回收他的内存，不过由于垃圾回收器是一个优先级很低的线程，所以不一定很快就会发现那些只具有弱引用的对象、虚引用：虚引用并不会决定对象的生命周期，如果一个对象仅持有虚引用，那么他就和没有任何引用一样，在任何时候都可能被回收，虚引用主要用来跟踪对象被垃圾回收的活动，虚引用必须和引用队列联合使用，当垃圾回收器准备回收一个对象时，如果发现他还有虚引用，就会在回收对象的内存之前，把这个虚引用加入到与之关联的引用队列中，程序可以通过判断引用队列中是否加入了虚引用，来了解被引用的对象是否将要被垃圾回收，程序如果发现某个虚引用已经被加入到引用队列中，那么就可以在所引用的对象内存被回收之前才去必要的行动。 Java对象的生命周期：创建（加载，连接（验证，准备，解析），初始化），使用，卸载。加载过程中：1.通过全类名获取定义此类的二进制字节流，2.将字节流所代表的静态存储结构转换为方法区的运行时数据结构。3.在内存中生成一个代表该类的class对象，作为方法区这些数据的访问入口。验证：文件格式验证，元数据验证，字节码验证，符号引用验证。准备：准备阶段是正式为类变量分配内存并设置类变量初始值的阶段（在方法区中分配），注意两点：1.这时候进行分配内存的仅包括类变量（static），不包括实例变量，实例变量会在对象实例化时随着对象一起分配在java堆中。2.这里所设置的初始值一般就是0值（初始化阶段才会赋值），但如果是final修饰的变量，那么这时候就直接赋值了。解析：解析阶段是虚拟机将常量池内的符号引用替换为直接引用的过程。主要针对类或接口、字段、类方法、接口方法、方法类型、方法句柄和调用限定符7类符号引用。初始化：初始化是类加载的最后一步，也是真正执行类中定义的java程序代码（字节码），初始化阶段是执行类构造器<clinit>()方法的过程。<clinit>()带锁线程安全，所以在多线程环境下进行类初始化可能会引起死锁，并且这种死锁很难被发现。卸载：卸载类需要满足3个条件：1.该类的所有实例对象都已被GC，也就是说堆不存在该类的实例对象。2.该类没有在其他任何地方被引用。3.该类的类加载器的实例已被GC。 如何判断对象是否存活？1.引用计数法：给对象中添加一个引用计数器，每当有一个地方引用它，计数器就加1，当引用失效计数器减1，任何时候计数器为0的对象就是不可能再被使用的，虽然实现简单，但是很难解决对象之间相互循环引用的问题。 2.可达性分析算法：基本思想就是通过一系列称为“GCRoots”的对象作为起点，从这些节点开始向下搜索，节点所走过的路径称为引用链，当一个对象到GCRoots没有任何引用链相连的话，则证明此对象是不可用的。可作为GCRoots的对象包括几种：虚拟机栈（栈帧中的本地变量表）中引用的对象，本地方法栈（Native方法）中的引用，方法区中类静态属性引用的对象，方法区中常量引用的对象。

1. 分代回收算法的大致过程？

答案：新生代：每次收集都会有大量对象死去，选用复制算法，只需要付出少量对象的复制成本就可以完成每次垃圾回收。复制算法：为了解决效率问题，复制算法出现了，他可以将内存分为大小相同的两块，每次使用其中的一块，当这一块的内存使用完后，就将还存活的对象复制到另一块去，然后再把使用的空间一次清理掉，这样就使每次的内存回收是对内存区间的一半进行回收。老年代：老年代的对象存活率高，而且没有额外的空间对他进行分配担保，所以选择标记清除，标记整理算法。标记清除算法：分为标记和清除两个阶段，首先标记出所有不需要回收的对象，在标记完成后统一回收掉所有没有被标记的对象，这种算法很明显有两个问题：1.效率问题2.空间问题（标记清除会产生大量的不连续的内存碎片）。标记整理算法：根据老年代的特点提出的一种标记算法，标记过程仍然与标记清除算法一样，但后续步骤不是直接对可回收的对象进行回收，而是让所有存活的对象向一端移动，然后直接清理掉边界以外的内存。

1. Synchronized关键字的作用（对于普通同步方法，静态同步方法，同步方法块而言，锁对象分别是什么）？synchronized是重入锁还是非重入锁？公平锁还是非公平锁？如何理解synchronized？（隐式，锁的可见性和互斥性）

答案：修饰实例方法，锁对象是当前实例，进入同步代码之前要获得当前对象实例的锁，修饰静态方法：锁对象是当前类，会作用于类的所有对象实例，因为静态成员不属于任何一个实例对象，是类成员，不管new了多少个对象，就只有一份。所以如果一个线程A调用一个实例的非静态同步方法，而线程B调用这个实例对象所属类的静态synchronized方法，是允许的，不会发生互斥现象。因为访问静态synchronized方法占用的锁是当前类的锁，而非静态synchronized方法占用的是当前实例对象的锁。修饰静态代码块,指定加锁对象，对给定对象加锁，进入代码块之前要获得给定对象的锁。synchronized(this)锁住的是当前对象，synchronized(\*\*\*.class)锁住的是当前类，锁住所有实例。Synchronized是可重入锁，非公平锁。如何理解synchronized？他修饰同步代码块（class），其关键就是必须要对对象的监视器monitor进行获取，当线程获取monitor后才能继续往下执行，否则就只能等待，而这个获取的过程是互斥的，即同一时刻只能有一个线程能够获取到monitor，在执行完同步代码块之后如果紧接着取执行一个静态同步方法，而这个方法锁的对象依然是这个类对象，那么这个正在执行的线程还需要获取该锁吗，答案是不必的，执行静态同步方法的时候就只有一条monitorexit指令，并没有monitorenter获取锁的指令，这就是锁的可重入性，即在同一锁程中，线程不需要再次获取同一把锁，每个对象拥有一个锁的计数器，当线程获取该对象锁后，计数器会加一，释放锁后就会将计数器减一。任意一个对象都拥有自己的监视器，当这个对象由同步代码块或者这个对象的同步方法调用时，执行方法的线程必须先获取该对象的监视器才能进入同步代码块或者同步方法中，否则将会被阻塞在同步代码块和同步方法的入口处，进入blocked状态。Synchronized为隐式锁，因为通过对象内部的监视器锁实现的，然后监视器锁本身又是依赖于底层的操作系统的Mutex Lock来实现的，然后操作系统实现线程间的切换就需要从用户态转换到内核态，成本非常高，耗时长，所以synchronized锁慢。Synchronized方法加锁：方法标志中存在标识ACC\_SYNCHRONIZED,该标识表示线程去获取对象监视器monitor锁。Synchronized的可见性必须满足两个条件：1.加锁时，必须清空当前线程自主内存里的共享变量，从主内存中重新读取一次共享变量。2.解锁前，必须把自主内存里的共享变量同步到主内存中。Synchronized关键字解决的是多个线程之间访问资源的同步性。Synchronized锁的互斥性：1.先获得互斥锁，2.清空自主内存3.从主内存中拷贝最新的共享变量到自主内存中4.执行代码5.将自主内存中的共享变量刷新到主内存中6.释放互斥锁。多线程访问synchronized关键字可能会发生阻塞。

1. Volatile的作用？volatile变量有什么特征？从JVM的角度，是如何保证volatile变量的可见性的？

答案：volatile能禁止指令重排，能保证变量在多个线程之间的可见性。特征：当一个线程对共享变量进行了修改，那么另外的线程都是立即就能看到修改后的最新值。如何保证可见性的？现在的java内存模型下，线程可以把变量保存到本地内存（如机器的寄存器）中，而不是直接在主存中进行读写，这就造成一个线程在主存中修改了一个变量的值，而另外一个线程继续使用他在寄存器中的变量的拷贝，造成数据不一致。Volatile是每次使用这个变量都到主存中进行读取。

1. 重入锁的含义？重入锁的公平性指的是什么？

答案：自己可以再次获取自己的内部锁，

1. 手写java单例模式 懒汉&静态内部类
2. Java线程的状态有哪些？手写代码，打印出java线程的6种状态？手写代码打印出两个线程交替打印1~10
3. 线程池有哪些重要参数？

JDK源码：

1. ArrayList和LinkedList的区别和应用？

答案：初始申请容量，多读场景，多写场景，随机读写场景

1. HashTable和ConcurrentHashMap区别？

答案：ConcurrentHashMap读不锁，锁分段

1. HashMap中key使用自定义复杂对象有什么要注意？

答案：重写equals（）和hashcode()

1. Synchronized是可重入锁吗？

答案：理解可重入锁的概念

1. Synchronized如何实现？

答案：轻量->自旋， 重量->内核级锁

1. Volatile作用和应用场景？

答案：并发类组件，标记变量用于并发标识

1. 如何在并发环境累加某类数据？

答案：Atomic类

1. 需要捕获任何异常该如何做？

答案：catch throwable

1. 线程常见状态，出现死锁的最可能是哪些状态？

答案：6种，block和wait较可能出问题

1. 用一个线程池来记录大量日志（数据可丢）

答案：有限/无限队列，超出队列做丢弃处理（自定义线程池）

第三方源码：

1. 为什么用spring bean？

答案：单例，单例管理

1. Spring bean 如何实现？

答案：实例化后置入ConcurrentHashMap

1. Spring jdbcTemplate是否线程安全？

答案：是线程安全，否则出现数据错乱

1. Mybatis大数据量插入？

答案：先分段，防止内存溢出，线程阻塞 1.insert value多条 2.多条insert循环 3.开启batch session

1. Spring AOP如何实现？

答案：JDK动态代理（有接口），CGLIB动态代理（无接口）

1. Spring MVC的控制器是单例还是多例，线程是否安全？

答案：一般是单例，基于方法，线程安全

1. Mybatis的${}和#{}的区别？

答案：#{}是预编译处理，${}是字符串替换

1. 时间复杂度o(1)，epoll的两种工作方式：1.水平触发（LT）2.边缘触发（ET）

答案：LT模式：若就绪的事件一次没有处理完要做的事件，就会一直去处理，即就会将没有处理完的时间继续放回到就绪队列中（即那个内核中的链表），一直进行处理。ET模式：就绪的事件只能处理一次，若没有处理完会在下次的其他事件就绪时再进行处理，而若以后再也没有就绪的事件，那么剩余的那部分数据也会随之而丢失。由此可见，ET模式的效率比LT模式的效率要高很多。

Redis：

1. Redis对象，字符串，链表，哈希，集合，有序集合及其底层实现（String，list/ziplist,dict/ziplist,intset/dict,ziplist/skiplist+dict）
2. Redis serverloop功能：清理过期键，清理失效客户端，持久化，主从或集群同步
3. RDB AOF持久化 条件
4. 最大内存及淘汰策略：volatile lru,volatile lfu, volatile-random,volatile-ttl,allkeys-lru, allkeys-lfu,allkeys-random,noeviction
5. Redis主从里的PYSNC过程：主判断RUN\_ID和offset与缓冲区关系决定全复制还是部分复制 ，命令传播+心跳检测
6. Redis什么时候拒绝客户端命令：达到最大内存，上次写识别且on stop write on bgsave,事件过程中，鉴权失败；数据正载入
7. 集群哨兵选举流程，raft算法
8. 集群数据库怎么分配，槽分派slot机制
9. 雪崩就是指缓存中大批量热点数据过期后系统涌入大量查询请求，因为大部分数据在Redis层已经失效，请求渗透到数据库层，大批量请求犹如洪水一般涌入，引起数据库压力造成查询堵塞甚至宕机，穿透：多次请求，且大量请求缓存和数据库都查不到，一般是黑客单点攻击，击穿：某个key非常热点，访问非常频繁，处于集中式高并发访问的情况，当这个key在失效的瞬间，大量的请求就击穿了缓存，直接请求数据库，就像是在一道屏障上凿开了一个洞。
10. 集群里主机挂了怎么选举和故障转移，跟哨兵里的选举和故障转移有什么区别
11. Max-memory达到后一定互拒绝写命令吗
12. Dict的rehash intset的升级 ziplist的连续扩散

JVM类：

1. JVM内存堆大小如何调整？

答案：Xms,Xmx

1. 一般会配置什么JVM参数呢？

答案：Xmx，Xms,Xmn,Xss,printGC

1. CMS GC和parallel GC如何选择？

答案：CMS偏低延迟，Parallel偏高吞吐

1. Xss是什么，有什么用？

答案：线程内存大小，栈内存，栈内OOM相关

1. 为什么要尽量避免Full GC？

答案：STW，影响应用延迟

1. 理想情况下可以做到零full gc吗？

答案：是可以的，每次young gc不残留内存引用

基础组件：

1. 一般数据库隔离级别？mysql innodb默认隔离级别如何体现？

答案：RC ，RR，同一事务两次查询结果是否一致的

1. 为什么innodb用b+树

答案：树高度低，适合硬盘查找

1. 一般innodb索引使用什么结构，为什么？

答案：b-Tree ,Hash索引不支持部分场景

1. Innodb主键方式：自增，UUID，不重复随机字符串？

答案：自增->uuid->不重复随机字符串，innodb树的构成方式，排序比较

1. Mysql主从如何同步？

答案：1.语句复制 2.行复制 3.事务复制

1. 为什么Redis sharding客户端一般使用一致性哈希？

答案：服务迁移，增删机器

1. 部署一套Redis高可用高性能环境，如何选择？

答案：Redis集群，主从（RDB+AOF）

1. 怎么传输对象到Redis，考虑点，关注点在哪里？

答案：序列化二进制

1. RabbitMQ和kafka如何选择？

答案：吞吐量，数据量（kafka）,消息可靠性（RabbitMQ）

微服务（spring cloud），注册中心

中间件（mysql，Redis，mq，nginx）

网络（http，tcp,rpc,netty,协议）

架构设计，模块划分

Jvm

Java基础（jdk源码，线程，线程安全，设计模式）

大数据（spark，hive，storm）

1. 一个线程读，一个线程写，不加锁怎么实现？

答案：copyOnWrite,看一下

1. Wait()和sleep（）的区别？

答案：（1）wait()一定要先获得锁，然后才能调用wait(),sleep（）不需要事先获得锁；（2）wait()会释放锁，等待notify()或者notifyAll()通知，sleep（）不会释放锁，到了时间之后会自动转成runnable