1. jdk动态代理与cglib代理的区别？

Jdk动态代理只能对实现了接口的类进行代理，而不能针对类；cglib代理针对类实现代理，对指定的类生成子类；

Jdk动态代理是利用反射机制生成一个实现代理接口的匿名类，而cglib代理是利用asm开源包，对代理对象类的class文件加载进来，通过修改其字节码生成子类来处理

1. jvm的调优命令有哪些？

Jps、jstas、jmap、jinfo

1. java会存在内存泄漏嘛？

会，长生命周期的对象持有短生命周期的对象的引用就很可能发生内存泄漏，尽管短生命周期对对象已经不在需要，但是因为长生命周期对象持有它的引用而导致不能回收，这就是java中内存泄漏的场景

1. 什么是双亲委派模式？如何打破？

双亲委派模式：如果一个类加载器需要加载一个类，那么首先它会把这个类请求委派给父类加载器去完成，每一层都是如此，一直递归到顶层，当父类加载器无法完成这个请求时，子类才会去尝试加载。

如何打破：写一个类加载器，继承classloader类，再重写loadclass和findclass方法

1. 为什么jdk8将永久代替换成元空间？

永久代在方法区中容易出现内存溢出，永久代会为GC带来不必要的复杂度，并且回收效率偏低。

1. string类型是如何实现不可变的？

首先，string是用final修饰的char数组，用final修饰就意味着必须在构造时为其赋值，且赋值后引用不能再改变，其次是string并没有提供接口来改变其value的值，value的值就不能获取。然而并不是真正的不能获取，可通过反射获取其内部属性。

1. hashCode()与equals()？

HashCode（）作用是获取哈希码，从而确定数据在哈希表中的索引位置，这只在哈希容器内才有意义。equals判断两个对象是否相等。Equals相等的话，则hashcode也相等。

1. 重写equals时为什么也要重写hashcode？

默认从object继承过来的hashcode是基于对象的id实现的；重写euqals是为了两个对象能够按我们自己的想法来判断是否相等，如果不重写hashcode方法，则会出现具有相同含义的不同对象的情况。

1. 如何判断一个常量是废弃常量？

只要这个常量没有被任何对象引用，则是废弃的。

10、如何判断一个类是无用的类？

1. 这个类的所有实例都已经被回收
2. 加载该类的calssloader已经被回收
3. 该类对应的calss对象没有在任何地方被引用，无法在任何地方通过反射的方法访问该类的方法

12、什么是happen-before原则？

首先happen-before不是意味着在什么什么之前发生，而是结果可见性；例若A happen-before B，那么A的结果对B是可见的，通过这些规则就可以保证程序能够按照我们预想的方案运转。

可以将这个原则分为两部分理解，单线程与多线程下的HB。单线程下通过语义分析数据依赖关系，编译器和处理器可以合理的优化我们的代码；但是在多线程的环境下不同线程之前的依赖关系由我们定义，处理器和编译器都无法通过分析感知，HB原则定义了某些特定场景下多线程间的数据依赖关系。

总的来说，HB是对单线程环境下的指令重排以及多线程环境下的线程间数据的一致性进行约束。

1. 单线程HB原则：同一个线程中，书写在前面的操作HB后面的操作
2. 锁的HB原则：同一个锁的unlock操作HB此锁的lock操作
3. Volatile的HB原则：对一个volatile变量的写操作HB对此变量的任意操作
4. HB的传递性原则：若A hb B，且 B hb C，则A hb C
5. 线程启动的HB原则：同一个线程的start方法HB此线程的其他方法
6. 线程中断的HB原则：对线程interrupt方法的调用HB被中断线程的检测到中断发送的代码
7. 线程终结的HB原则：线程中所有的操作都HB线程的终止操作
8. 对象创建的HB原则：一个对象的初始化完成先于它的finalize方法调用

13、什么是内存屏障？为什么需要内存屏障？

内存屏障：内存屏障是一条指令，该指令可以对编译器和处理器的指令重排做出限制。比如，一条内存屏障指令可以禁止编译器和处理器将其后面的指令移到内存屏障指令之前。

为什么需要内存屏障：编译器和处理器指令重排只能保证在单线程执行下逻辑正确，在多个线程同时读写多个变量的情况下，若不对指令重排做出一定限制，指令重排后每次的执行顺序都可能不一样，就会产生不同的结果。

14、jvm中为什么要分为eden和survivor？

为了避免每次minor gc 后存活下来的对象都进入老年代，通过给存活下来的对象设置年龄阈值，只有超过才能进入老年代，减少老年的的gc次数。

15、什么情况下会发生栈内存溢出？

线程请求的栈深度大于jvm所允许的最大栈深度，一般方法递归调用容易产生此问题。

若栈可动态扩展，则会申请内存，若申请不到则会出现内存溢出（线程过多）。

16、有哪些类加载器，以及类加载过程？

类加载器：bootstarp classloader、extention classloader、app classcolder

类加载过程：（1）加载：根据路径找到相应的class文件

（2）验证：检查class文件的正确性

（3）准备：给类中的静态变量分配内存空间

（4）解析：虚拟机将常量池中的符号引用替换成直接引用的过程

（5）初始化：对静态变量和静态代码块执行初始化工作

17、G1、ZGC、CMS垃圾收集器的过程？

G1：初始标记，并发标记，最终标记，筛选回收

ZGC：并发标记，并发预备重分配，并发重分配，并发重映射

CMS：初始标记，重新标记，并发清除

18、哪些可以作为GC ROOT？

虚拟机栈（栈帧中的本地变量表）中引用的对象。

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。

本地方法栈中JNI（即一般说的Native方法）引用的对象。

19、高并发、任务执行时间短的业务怎样使用线程池？并发不高、任务执行时间长的业务怎样使用线程池？并发高、业务执行时间长的业务怎样使用线程池？

（1）高并发、任务执行时间短：线程池中线程数可以设置为CPU核数+1，减少线程上下文切换

（2）并发不高、任务执行时间长：

（a）假如业务时间长是集中在IO操作上，也就是IO密集型任务，因为IO操作并不占用CPU，所以不要让所有的CPU闲下来，可以加大线程池中的线程数目，让CPU处理更多的业务

（b）假如业务时间长是集中在计算操作上，也就是计算密集型任务，这个就会耗费CPU资源了，只有把线程数目设置成与CPU核数差不多的，减少上下文的切换

（3）并发高、业务执行时间长：解决这种类型任务的关键不在于线程池而在于整体架构的设 计，看看这些业务里面某些数据是否能做缓存、或增加服务器。最后，业务执行时间长的问题，也可能需要分析一 下，看看能不能使用中间件对任务进行拆分和解耦。

20、常见的集中线程池及应用场景

（1）FixedThreadPool：可重用固定线程数的线程池。（适用于负载比较重的服务器） FixedThreadPool使用无界队列LinkedBlockingQueue作为线程池的工作队列 该线程池中的线程数量始终不变。当有一个新的任务提交时，线程池中若有空闲线程，则立即 执行。若没有，则新的任务会被暂存在一个任务队列中，待有线程空闲时，便处理在任务队列 中的任务。

（2）SingleThreadExecutor：只会创建一个线程执行任务。（适用于需要保证顺序执行各个任 务；并且在任意时间点，没有多线程活动的场景。） SingleThreadExecutorl也使用无界队列LinkedBlockingQueue作为工作队列 若多余一个任务被提交到该线程池，任务会被保存在一个任务队列中，待线程空闲，按先入先 出的顺序执行队列中的任务。

（3）CachedThreadPool：是一个会根据需要调整线程数量的线程池。（大小无界，适用于执行很 多的短期异步任务的小程序，或负载较轻的服务器） CachedThreadPool使用没有容量的SynchronousQueue作为线程池的工作队列，但 CachedThreadPool的maximumPool是无界的。 线程池的线程数量不确定，但若有空闲线程可以复用，则会优先使用可复用的线程。若所有线 程均在工作，又有新的任务提交，则会创建新的线程处理任务。所有线程在当前任务执行完毕 后，将返回线程池进行复用。

（4）ScheduledThreadPool：继承自ThreadPoolExecutor。它主要用来在给定的延迟之后运行 任务，或者定期执行任务。使用DelayQueue作为任务队列。

21、什么是AQS以及其支持几种同步方式？

AQS：AQS是AbustactQueuedSynchronizer的简称，它是一个Java提高的底层同步工具类，用一 个int类型的变量表示同步状态，并提供了一系列的CAS操作来管理这个同步状态。AQS是一个用来构建锁和同步器的框架，使用AQS能简单且高效地构造出应用广泛的大量的同 步器，比如我们提到的ReentrantLock，Semaphore，其他的诸如 ReentrantReadWriteLock，SynchronousQueue，FutureTask等等皆是基于AQS的。

同步方式：独占式、共享式

22、设计模式六大原则

（1）开闭原则：对扩展开放，对修改关闭

（2）里氏替换原则：任何基类可以出现的地方，子类一定可以出现

（3）依赖倒置原则：针对接口编程，依赖于抽象而不依赖于具体

（4）接口隔离原则：使用多个隔离的接口，比使用单个接口要好。降低类之间的耦合度的

（5）迪米特法则（最少知道原则）：一个实体应当尽量少的与其他实体之间发生相互作用，使得 系统功能模块相对独立

（6）合成复用原则：尽量使用合成/聚合的方式，而不是使用继承