**JAVA内存模型**

1. 每个线程都有自己的本地内存空间，多个线程共享主内存，本地内存和主内存数据一致性通过lock，unlock，read，load，assign，use，store，write保证；
2. 变量的创建必须在主存中，变量的修改必须在本地内存；
3. volatile变量，确保内存可见性，确保指令不会被重排；
4. synchronized关键字，确保临界资源排他访问，指令重排，内存可见性（代码块结束的时候，刷新cache line，回写主存）；
5. 内存屏障，确保屏障前后代码不被重排，确保屏障之后的代码，在执行前，从主存加载数据；
6. 原始类型double&long，64位，非原子操作，以32位为单位；
7. before-happen原则，定义两个操作的偏序关系，前序操作产生的影响会被后续操作感知到；

**JAVA内存结构**

1. JAVA内存模型5大区域：方法区（线程共享），堆区（线程共享），程序计数器（线程独享），JVM栈（线程独享）和本地方法栈（线程独享）；
2. 方法区：存放类信息，静态变量，常量和运行时常量池；
3. 程序计数器：每个线程独有，保存代码执行的行号；
4. JVM栈：局部变量表（局部变量，方法参数地址），操作数栈，方法返回地址，动态链接；
5. 本地方法栈：为native方式使用；
6. 堆：共享内存，动态分配，GC主要工作的区域，老年代的回收主要工作在方法区这里；

**JAVA垃圾回收**

1. JAVA内存模型五个区域，其中JVM栈，本地方法区和程序计数器都是随着线程而生线程而灭，所以不在GC的范围内。GC的主要范围包括堆和方法区；
2. 堆对应的是年轻代和老年代，方法区对应的是持久代；
3. 如何界定一个对象是可以被回收的，对象不再被引用就可以被回收；
4. 对象不再被引用，有两种判定方法，一种是引用计数法，另一种是可达性分析法；
5. 引用计数法，每个对象创建出来之后，会有一个引用计数器，如果被其他对象引用，计数器加1，不再被其他对象引用，计数器减1，当计数器为0的时候，表示可以被回收了；
6. 引用计数法，存在循环引用的风险，为了规避这个风险，引入弱引用；
7. 可达性分析算法，我们会定义一个对象为GC Root，判断该对象是否被引用，只需要看从该对象有没有一条路径可达GC Root；
8. 哪些对象可以作为GC Root：JVM栈所引用的对象，方法区中静态变量和常量，本地方法栈中所引用的对象；
9. 要判断一个对象死亡，需要经历两个步骤，第一个步骤是可达性分析，第二个步骤是对象的finalize方法中，该对象可能被恢复引用不；
10. 堆内存年轻代分为Eden区，From Survivor区和To Survivor区，他们的比例是8:1:1，可使用的内存是90%，因为有一个10%的区是用来复制对象使用的，在非GC情况下是闲置的；
11. 垃圾回收算法，主要有三种，复制算法，标记清除算法，标记压缩算法；
12. 复制算法，用于年轻代的内存回收，因为，年轻代的对象生命周期较为短暂，短时间内会有大量的可回收的对象存在，采用该算法，可以更加高效，内存碎片也会较少；
13. 标记清除算法，用于老年代的垃圾回收，相对于标记压缩算法，效率会高一些，但是内存碎片会比较多；
14. 标记压缩算法，用于老年代的垃圾回收，内存利用率较高，性能较弱；
15. 年轻代垃圾回收器：Serial，ParNew和Parallel Scavenge;
16. 老年代垃圾回收器：Serial Old，Parallel Old和CMS；
17. 单线程串行回收器组合：Serial + Serial Old；
18. 多线程并行回收器组合：ParNew + CMS + Serial Old和Parallel Scavenge + Parallel Old；
19. CMS回收器，CPU高度敏感型，采用标记清除算法，产生内存碎片，如果内存用尽，是没办法进行CMS的，这个时候需要STW，采用Serial Old进行内存回收，也就是所谓的FGC；
20. CMS回收过程，经历几个阶段：
    1. 初始标记：GC Root直接关联的对象，需要STW；
    2. 并发标记：做GC Root Tracing，多线程并行；
    3. 重新标记：对上一个阶段产生的新的可回收对象进行标记，需要STW；
    4. 并发清除：并行多线程进行对象清理；
21. 浮动垃圾，我们在并发清理阶段，应为业务线程和GC线程并行执行，导致有些对象从不可达变为可达，在重新标记阶段，通过STW，对上一个阶段不可达对象重新扫描，把上一阶段重新变为可达的对象去掉标记；但是并不对并发清理阶段，漏标记的不可达对象进行额外处理（产生的新的垃圾对象），这部分对象就是所谓的浮动垃圾，需要下次GC才能处理；
22. G1垃圾回收器，将堆空间分成若干个Region，每个Region的大小从1M-32M，垃圾回收的最小内存单元是Region；
23. G1垃圾回收器的Region有四种角色，E，S，O和H；角色会发生变化；
24. G1垃圾回收器，在Region之间采用的是复制算法；
25. 每个Region由Card Page组成，每个Card Page大小为512 Byte；Card Table是一个字节数组，数组的每个元素为1 Byte，另外一个Region有引用该Region的Card Page的首地址对应的索引编号，address >> 9；
26. Card Table是Remember Set的一种实现；

**JAVA基础知识**

1. ConcurrentHashMap数据结构，在1.8之前，采用的是分段锁，为了提升并发性，每个分段有一个锁保证线程安全，每个分段由若干个HashEntry组成，每个HashEntry由一个链表组成，采用链地址法解决冲突问题；在1.8以及之后，去掉了分段锁，采用Node+CAS的方式保证线程安全，进一步提升并行性，另外当单链表的节点数大于8的时候，会转换成红黑树，提升查找效率；
2. ArrayList和LinkedList比较，前者为线性表，后者为链表，前者适合查找，时间复杂度为常数，后者适合插入和删除操作；
3. Vector，线程安全的数组，和ArrayList类似，ArrayList线程不安全；
4. HashTable和HashMap，前者线程安全，需要锁表；后者线程不安全；
5. HashMap线程不安全，如果多线程put，get线程可能会导致进入死循环；多线程put非空元素，get得到空值；多线程put，导致元素丢失；
6. String，StringBuilder和StringBuffer的区别，String是不可变的，意味着每次会生成一个新的对象；StringBuilder是线程不安全的，StringBuffer是线程安全的；
7. Vector和ArrayList的区别，Vector是线程安全的，ArrayList是线程不安全的，默认情况下，Vector在存储不够的时候是扩展1倍，ArrayList是扩展50%+1个；
8. HashMap和HashTable的区别，前者是线程不安全的，后者是锁表，线程安全的；前者是继承于AbstractMap，后者继承自Dictionary；前者可以有一个NULL的Key，后者不允许；
9. JAVA线程池，固定数量线程的线程池，无界队列线程池，单个线程的线程池；线程池满的时候，会采取几种抛弃策略，抛出异常，直接忽略，任务提交者线程执行任务，抛弃任务队列中最老的任务；
10. 可以向线程池提交两种任务，一种是Runnable一种是Callable，方法签名不同，后者可以抛出异常和获取返回值；
11. LinkedHashMap是带一个双向链表的HashMap，可以用它实现LRU，最近访问的节点，修改该节点双向链表的指针，将其至于链表头部，淘汰采用从链表末尾开始删除；
12. Object对象的wait/notify方法，该对象的两个操作必须在同步块中，比如synchronized(obj){}，调用对象wait方法，让对象放弃所持有的对象锁，并让线程处于等待状态；调用notify方法，之前wait的线程重新被唤醒，可以重新竞争对象锁；
13. Java原始数据类型的大小分别是，byte是1个字节，short是2个字节，int是4个字节，float是4个字节，double和long是8个字节；
14. 一个对象实例化的过程包括：父类的静态变量，父类的静态代码块，子类的静态变量，子类的静态代码块，父类的非静态变量，父类构造函数，子类的非静态变量，子类的构造函数；
15. 对HashMap的加载因子的理解，用于描述哈希表的散列程度，负载因子越大，表示散列程度越低，查找效率越低；如果负载因子越小，说明散列程度越高，空间换时间，查找效率越高；capacity表示桶数组的大小，initalCapacity表示初始容量，大小为最接近initialCapacity大小的2的幂次方；负载因子默认为0.75，如果超过，需要resize，默认为2倍；在扩容期间，需要rehash，链表的元素会颠倒位置；
16. Class.forName和ClassLoader区别在于，前者会执行静态代码块，后者不会；
17. 动态代理，可以通过JDK动态代理实现，也可以通过CGLIB来实现，区别在于，前者在创建对象期间效率比较高，后者在方法执行期间效率比较高；另外，前者必须要代理类和被代理类实现相同的接口，后者可以不用实现接口；前者采用反射实现，后者采用字节码技术实现ASM，通过创建一个继承类的子类对象，通过字节码技术修改子类来实现；当一个方法没有被aop事务包裹，那么该方法调用一个被事务包裹的aop方法时，该事务失效；
18. JAVA中，private只能在本类可见，protected只能在本类，本包和子类可见，default只能在本包可见，public所有可见；
19. 深拷贝和浅拷贝的区别，后者拷贝的是基础数据类型，引用数据类型只是拷贝了引用；
20. 错误和异常的区别，前者是由虚拟机创建和抛出的，比如栈溢出异常和OOM；异常包括受检查异常和运行时异常，前者，编译器要求必须在方法抛出申明的任何可能的受检查异常，比如FileNotFoundException和EOFException；后者是程序在运行过程中产生的异常，比如数组越界，算数运算，ClassNotFoundException，空指针异常和参数非法异常；
21. JAVA泛型解决的问题，是为了解决向下转型的问题，向下转型不能在编译期间发现问题，只有在运行时可能发生错误；
22. HashSet，实现了Set接口，同时，组合了一个HashMap，所以在保证去重的同时，也可以保证查找效率；
23. JAVA的类加载器，整个加载过程包括加载，链接和初始化三个过程；其中链接包括校验，准备和解析三个阶段；其中校验阶段是指对字节码流进行检查，加载文件格式进行检查等；准备阶段是为静态变量分配内存空间；解析阶段是指将二进制文件符号引用转换为直接引用；
24. JAVA的类加载器包括：根类加载器（JDK核心类库），扩展类加载器（lib/ext），系统类加载器（classpath）以及用户类加载器；
25. 所谓的双亲委派模式，当收到加载类的指令后，本类先委托父类去加载该类，如果父类还存在上级，一直递归上去，当父类都加载不了，再由本类去加载，如果还是加载失败，抛出ClassNotFoundException；

**数据库基础**

1. MYSQL的事务四个特性：原子性，一致性，隔离性和持久性；
2. MYSQL并发问题：
   1. 脏读：一个事务读取了另一个未提交的事务数据；
   2. 不可重复读：一个事务多次读取，读取到的数据是不一样的，针对update操作；
   3. 幻读：一个事务多次读取，读取到的数据是不一样的，针对insert/delete操作；
3. 事务隔离级别：
   1. Read-Uncommitted
   2. Read-Committed：解决脏读；
   3. Repeatable Read：解决脏读和不可重复读；
   4. Serialize：解决脏读，不可重复读和幻读；
4. MYSQL采用的是聚簇索引，通过主键建立的一颗B+树，叶子节点是数据记录；
5. MYSQL的innodeDB引擎，辅助索引通过B+树检索到主键，如果是覆盖索引的话，只需要查询一次，如果是非·覆盖索引，需要在主索引通过主键再次检索一次；对于MYSIM采用的是非聚簇索引，所以和数据分离的方式；
6. MYSQL索引使用：
   1. 索引采用最左匹配原则；
   2. 索引列上做函数运算，索引失效；
   3. 存储引擎，索引条件右边的列是不会使用的，比如where id=1 and age > 5 and sex=1，sex字段不会被使用，age着重排序；
   4. 尽量使用覆盖索引，减少一次检索；
   5. 不等于导致索引失效；
   6. is null和is not null导致索引失效；
   7. like如果%最前面导致索引失效；
   8. 字符串不加单引号导致索引失效；
   9. or导致索引失效；

**其他基础**

1. 一致性Hash算法要解决的问题，在集群几点扩容和缩容期间，尽可能的减少失效数据回源，导致对底层系统和存储的压力；
2. 一致性HASH算法的原理：构造一个2的32次方的整数环，每一个整数代表一个节点，先计算key的HASH值，然后在顺时针方向找到距离这个HASH值最近的节点，完成映射；
3. 操作系统死锁的四个必要条件，护持条件，请求保持，不剥夺条件，循环等待；
4. 继承是is-a的关系，聚合是has-a的关系，接口实现是like-a的关系，组合是only-has-a的关系；
5. IO模型，IO包含两个阶段，数据等待和数据拷贝；
   1. 阻塞模型：在等待阶段和拷贝阶段，应用进程都处于阻塞状态；
   2. 非阻塞模型：在等待阶段为非阻塞，系统调用返回EWOULDBLOCK错误码，在拷贝阶段处于阻塞状态；
   3. 信号驱动IO：数据等待阶段处于非阻塞，不需要轮训查看数据是否准备好，只需要向内核注册一个信号，等数据准备好之后，通过信号的方式通知用户进程；
   4. 多路复用：数据等待阶段，select阻塞，支持多文件描述符，当有一个文件描述符准备好，select就返回；
   5. AIO：向内核注册一个信号，在等待阶段和拷贝阶段，都是非阻塞的，当数据拷贝完成，通过信号通知用户进程；