20181027考题

1. GC算法：

标记清除，标记可以回收的对象，不会整理内存，出现空闲内存空间，删除较快，但是有内存碎片，可能引起大对象分配不了

标记整理，内存整理，存活对象重新排列到内存靠前的位置，剩余内存作为可用

年轻代，伊甸，s0,s1，标记整理，回收90%对象，

Old，标记清除，常驻对象较少，相对对象较大，标记整理会带来内存拷贝引起的性能问题。

Min GC

Major GC在old，

手动关闭system.gc，不会立即执行，但是会执行一次fullGC，导致

1.8JVM默认用G1，优化点 分代之间没有物理隔离，region 默认64M，可回收，打分计算，GC只回收价值最高的回收，region排优先级，提高GC效率，缩短stw时间，精确估算出一次gc时间。

CMS，垃圾回收，两个标记步骤是stw，其他清除步骤是和用户线程并发，可配置多，

Panew，young区收集器，GC时会一直stw。

【答案】：

* 标记-清除算法：分为“标记”和“清除”两个阶段，首先标记出所有需要回收的对象，在标记完成后统一回遍历正在回收的分代，收掉所有被标记的对象。它的主要缺点有两个：一个是效率问题，清除过程的效率不高；另外一个是空间问题，标记清除之后会产生大量不连续的内存碎片，空间碎片太多可能会导致，当程序在以后的运行过程中需要分配较大对象时无法找到足够的连续内存而不得不提前触发另一次垃圾收集动作。
* 复制算法：将可用内存按容量划分为大小相等的两块，每次只使用其中的一块。当这一块的内存用完了，就将还存活着的对象复制到另外一块上面，然后再把已使用过的内存空间一次清理掉。每次都是对其中的一块进行内存回收，内存分配时也就不用考虑内存碎片等复杂情况，只要移动堆顶指针，按顺序分配内存即可，实现简单，运行高效。只是这种算法的代价是将内存缩小为原来的一半，持续复制长生存期的对象则导致效率降低。
* 标记-整理算法：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对象进行清理，而是让所有存活的对象都向一端移动，然后直接清理掉端边界以外的内存。能避免标记-清除算法清除效率和内存碎片的问题。
* 分代收集算法：根据对象存活周期的不同将内存划分为几块，一般把java堆分为新生代和老年代，根据各个年代的特点财通最适当的收集算法。新生代使用复制算法，老年代使用标记-清除或者标记-整理算法。

1. 工作中有没有遇到过OOM，怎么排查的？

没遇到，但是帮别人分析过。堆栈看哪个区溢出？Jstate，jcutil，P区 O区 如果是100%，则表明溢出

Jmap，看对象分布情况，对象是否大，实例数多，hashmap占用内存多，是否过多hashmap，缓存过大，保存大对象。

配置出错，rmq初始化 5G堆外内存，2G direct mem

JVM堆内存配置有问题，因为消费者缓存多条数据

OOM之前会出现多次fullgc

Yong区 不会出现OOM，提前晋升到老年代。

【答案】：

当JVM因为没有足够的内存来为对象分配空间并且垃圾回收器也已经没有空间可回收时，就会抛出这个OOM。原因主要是：1）分配的少了：比如虚拟机本身可使用的内存（一般通过启动时的VM参数指定）太少。2）应用用的太多，并且用完没释放，浪费了。此时就会造成内存泄露或者内存溢出。

内存泄露：申请使用完的内存没有释放，导致虚拟机不能再次使用该内存，此时这段内存就泄露了，因为申请者不用了，而又不能被虚拟机分配给别人用。

内存溢出：申请的内存超出了JVM能提供的内存大小，此时称之为溢出。

按照JVM规范，除了程序计数器不会抛出OOM外，其他各个内存区域都可能会抛出OOM。

最常见的OOM情况有以下三种：

java.lang.OutOfMemoryError: Java heap space ------>java堆内存溢出，此种情况最常见，一般由于内存泄露或者堆的大小设置不当引起。对于内存泄露，需要通过内存监控软件查找程序中的泄露代码，而堆大小可以通过虚拟机参数-Xms,-Xmx等修改。

java.lang.OutOfMemoryError: PermGen space ------>java永久代溢出，即方法区溢出了，一般出现于大量Class或者jsp页面，或者采用cglib等反射机制的情况，因为上述情况会产生大量的Class信息存储于方法区。此种情况可以通过更改方法区的大小来解决，使用类似-XX:PermSize=64m -XX:MaxPermSize=256m的形式修改。另外，过多的常量尤其是字符串也会导致方法区溢出。这个错误只会出现在1.8之前，从1.8开始PermGen被Metaspace替换，Metaspace可动态扩容、缩容，使用的是用户本地内存而不占用堆，所以可以避免产生元数据区内存溢出。

java.lang.StackOverflowError ------> 不会抛OOM error，但也是比较常见的Java内存溢出。JAVA虚拟机栈溢出，一般是由于程序中存在深度递归调用造成的，栈大小设置太小也会出现此种溢出。可以通过虚拟机参数-Xss来设置栈的大小，默认为1M。

要dump堆的内存镜像，可以采用如下两种方式：

设置JVM参数-XX:+HeapDumpOnOutOfMemoryError，设定当发生OOM时自动dump出堆信息。使用JDK自带的jmap命令。"jmap -dump:format=b,file=heap.bin <pid>"   其中pid可以通过jps获取。

dump堆内存信息后，需要对dump出的文件进行分析，从而找到OOM的原因。常用的工具有：mat: eclipse memory analyzer, Debuggers, profilers, heap dump analyzers等工具

1. Java的基本数据结构哪些是线程安全的？automic\*，currentHshMap，copyOnWrite\*， currentskipMap，vetor，blockingQueue？Stack？list?set?threadLocal

线程不安全的数据结构怎么保证数据一致可靠？

原子操作，读，修改，写入；先检查后修改；主动加锁，同步，串行去做。

使用并发容器，替换。

全局变量和类变量被多线程修改，尽量使用局部变量。

【答案】：

Java基本数据结构中hashtable和vector是线程安全的。

方法1：ArrayList，HashSet,LinkedList,HashMap等等非线程安全的类，通过[工具类Collections](http://how2j.cn/k/collection/collection-collections/369.html)转换为线程安全的，例如Collections.synchronizedList。

方法2：同步代码块/同步方法（synchronized）

方法3：重入锁，Lock锁机制， 通过创建Lock对象，采用lock()加锁，unlock()解锁，来保护指定的代码块

方法4：使用Threadlocal 类来包装共享变量，做到每个线程有自己的copy。

方法5：非阻塞同步，CAS是实现非阻塞同步的计算机指令，它有三个操作数：内存位置，旧的预期值，新值，在执行CAS操作时，当且仅当内存地址的值符合旧的预期值的时候，才会用新值来更新内存地址的值，否则就不执行更新。

方法6：使用ConcurrentLinkedQueue，ConcurrentHashMap，ConcurrentLinkedHashMap，ConcurrentSkipListMap替代。