 -Xms：最小堆大小 -Xmx：最大堆大小 -Xminf and -Xmaxf：GC（垃圾回收）之后可用空间的最小值最大值 -Xmine and -Xmaxe：堆增长的最小最大值 -Xmint and -Xmaxt：垃圾回收占时间整个运行时间的比例，默认是5%。如果回收时间小于5%，那么它就缩减堆，反之增大。 一般来说只要对Xms和Xmx设置合理，后面的三对不用特别设置。可以看看http://publib.boulder.ibm.com/infocenter/javasdk/v5r0/index.jsp上heap expasion和heap shrinkage两章的说明，除非有下文的情况： 如果使用大小可变的堆（比如，-Xms 和 -Xmx 不同），应用程序可能遇到这样的情况，不断出现分配失败而堆没有扩展。这就是堆失效，是由于堆的大小刚刚能够避免扩展但又不足以解决以后的分配失败而造成的。通常，垃圾收集周期释放的空间不仅可以满足当前的分配失败，而且还有很多可供以后的分配请求使用的空间。但是，如果堆处于失效状态，那么每个垃圾收集周期释放的空间刚刚能够满足当前的分配失败。结果，下一次分配请求时，又会进入垃圾收集周期，依此类推。大量生存时间很短的对象也可能造成这种现象。避免这种循环的一种办法是增加 -Xminf 和 -Xmaxf 的值。比方说，如果使用 -Xminf.5，堆将增长到至少有 50% 的自由空间。同样，增加 -Xmaxf 也是很合理。如果 -Xminf等于0.5，-Xmaxf 为默认值 0.6，因为 JVM 要把自由空间比例保持在 50% 和 60% 之间，所以就会出现太多的扩展和收缩。两者相差 0.3 是一个不错的

选择，这样 -Xmaxf.8 可以很好地匹配 -Xminf.5。 如果记录表明，需要多次扩展才能达到稳定的堆大小，但可以更改 -Xmine，根据应用程序的行为来设置扩展大小的最小值。目标是获得足够的可用空间，不仅能满足当前的请求，而且能满足以后的很多请求，从而避免过多的垃圾收集周期。-Xmine、-Xmaxf 和 -Xminf 为控制应用程序的内存使用特性提供了很大的灵活性。 摘自Java性能优化的策略和常见方法
所以在应用正式上线的头一段时间，最好把GC日志打开，观察一下堆（heap）的增长（expasion）和收缩（shrinkage）。最佳的情况就是，每次回收后可用的堆大小占整个堆的50%左右。如果回收后才腾出30%不到的可用空间，那就该再调整一下上述的参数了。下图看起来直观一点，使用-verbose:size参数可以查看当前的默认值。 那为何不把Xms和Xmx设置成一样大，就像SUN的JDK所推荐的那样？ Using the same values is not usually a good idea, because it delays the start of garbage collection until the heap is full. The first time that the Garbage Collector runs, therefore, becomes a very expensive operation. Also, the heap is more likely to be fragmented and require a heap compaction. Again this is a very expensive operation. …… If the Garbage Collector cannot find enough garbage, it runs compaction. If the Garbage Collector finds enough garbage, or any of the other conditions for heap expansion are met , the Garbage Collector expands the heap. 因为IBM JDK采用的是标记（mark）-扫描（sweep）-标记-……-扫描-紧凑排列（compact），如果还不能提供足够的空间，扩展堆（expasion）。依次循环，直到达到最大堆大小。每次扩展前，那些长存的对象就被调

整到堆的底部，每次扩展后需要再动的量就很少。所以如果把Xms 设置成和Xmx一样，那么扫描和紧凑排列这么一个充满内存碎片的大堆的开销将大大高于从小扩展堆的开销。
The overheads of expanding the heap are almost trivial compared to the cost of collecting and compacting a very large fragmented heap.
这是由于IBM的GC特点造成的，而SUN的JDK采用的是分代回收的策略，所以就没有这种情况，反而会受益于堆大小一致。不过这么说起来，用了-Xgcpolicy:gencon，就应该把最小堆最大堆设置成一样咯。 在 Gencon回收策略下，可以通过-Xmn来设置婴儿区域（Nursery或者叫young）的大小，通过-Xmo来设置长存区（tenured或者 old）的大小。注意-Xmn不能和-Xmns/-Xmnx参数一起使用，-Xmo不能和-Xmos/-Xmox一起使用，否则会报错。前者就是把年轻代和长存代的大小固定了，而后两者就是设定两个部分最大最小的范围，默认情况下： -Xmns是-Xms的25%或者64M（在JDK 5.0中默认是25%） -Xmnx是-Xmx的25%或者64M（同上） -Xmos是-Xmx减去-Xmns的大小 -Xmox是和-Xmx一样大 可见默认的年轻代太小了，生产环境中有必要改大一点。因为年轻代使用的是复制策略，所以回收速度相当快（minor gc），而长存代使用的是和optavgpause 策略相似的方式进行并发标志、扫描策略，回收速度比较慢（major gc）。理想情况是minor gc和major gc的比值在1：10左右。（可以通过GC日志查看回收的区域） gencon中年老期限（Tenure age）和倾斜比率（Tilt ratio）这两个参数是JVM自己动态调整的。 针对固定对象问题（Pinned Objects），使用-Xk -Xp参数设定kCluster和pCluster，Avoiding Java heap fragmentation with Java SDK

V1.4.2. 一些不常用的参数： -Xloainitial 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com