<http://blog.csdn.net/zhangliao613/article/details/78560630>

感谢有奉献精神的人

转自：http://blog.csdn.net/a503921892/article/details/39048889

年轻代 年老代概念 http://jefferent.iteye.com/blog/1123677

JVM的堆的内存， 是通过下面面两个参数控制的   
  
-Xms 最小堆的大小， 也就是当你的虚拟机启动后， 就会分配这么大的堆内存给你   
-Xmx 是最大堆的大小   
  
当最小堆占满后，会尝试进行GC，如果GC之后还不能得到足够的内存(GC未必会收集到所有当前可用内存)，分配新的对象，那么就会扩展堆，如果-Xmx设置的太小，扩展堆就会失败，导致OutOfMemoryError错误提示。  
  
实际上，细节不止于此， 堆还会被分成几个不同的区域，分别应用不同的GC算法

-------------------------------

google到了一篇博文，贴出来，漫漫看：<http://unixboy.javaeye.com/blog/174173>

### [JVM调优总结 -Xms -Xmx -Xmn -Xss](http://unixboy.javaeye.com/blog/174173)

1. 堆大小设置  
   JVM 中最大堆大小有三方面限制：相关操作系统的数据模型（32-bt还是64-bit）限制；系统的可用虚拟内存限制；系统的可用物理内存限制。32位系统下，一般限制在1.5G~2G；64为操作系统对内存无限制。我在Windows Server 2003 系统，3.5G物理内存，JDK5.0下测试，最大可设置为1478m。  
   典型设置：
   * java -Xmx3550m -Xms3550m -Xmn2g -Xss128k  
     -Xmx3550m：设置JVM最大可用内存为3550M。  
     -Xms3550m：设置JVM促使内存为3550m。此值可以设置与-Xmx相同，以避免每次垃圾回收完成后JVM重新分配内存。  
     -Xmn2g：设置年轻代大小为2G。整个JVM内存大小=年轻代大小 + 年老代大小 + 持久代大小。持久代一般固定大小为64m，所以增大年轻代后，将会减小年老代大小。此值对系统性能影响较大，Sun官方推荐配置为整个堆的3/8。  
     -Xss128k： 设置每个线程的堆栈大小。JDK5.0以后每个线程堆栈大小为1M，以前每个线程堆栈大小为256K。更具应用的线程所需内存大小进行调整。在相同物理内 存下，减小这个值能生成更多的线程。但是操作系统对一个进程内的线程数还是有限制的，不能无限生成，经验值在3000~5000左右。
   * java -Xmx3550m -Xms3550m -Xss128k -XX:NewRatio=4 -XX:SurvivorRatio=4 -XX:MaxPermSize=16m -XX:MaxTenuringThreshold=0  
     -XX:NewRatio=4:设置年轻代（包括Eden和两个Survivor区）与年老代的比值（除去持久代）。设置为4，则年轻代与年老代所占比值为1：4，年轻代占整个堆栈的1/5  
     -XX:SurvivorRatio=4：设置年轻代中Eden区与Survivor区的大小比值。设置为4，则两个Survivor区与一个Eden区的比值为2:4，一个Survivor区占整个年轻代的1/6  
     -XX:MaxPermSize=16m:设置持久代大小为16m。  
     -XX:MaxTenuringThreshold=0：设置垃圾最大年龄。如果设置为0的话，则年轻代对象不经过Survivor区，直接进入年老代。对于年老代比较多的应用，可以提高效率。如果将此值设置为一个较大值，则年轻代对象会在Survivor区进行多次复制，这样可以增加对象再年轻代的存活时间，增加在年轻代即被回收的概论。
2. 回收器选择  
   JVM给了三种选择：串行收集器、并行收集器、并发收集器，但是串行收集器只适用于小数据量的情况，所以这里的选择主要针对并行收集器和并发收集器。默认情况下，JDK5.0以前都是使用串行收集器，如果想使用其他收集器需要在启动时加入相应参数。JDK5.0以后，JVM会根据当前[系统配置](http://java.sun.com/j2se/1.5.0/docs/guide/vm/server-class.html)进行判断。
   * 吞吐量优先的并行收集器  
     如上文所述，并行收集器主要以到达一定的吞吐量为目标，适用于科学技术和后台处理等。  
     典型配置：
     + java -Xmx3800m -Xms3800m -Xmn2g -Xss128k -XX:+UseParallelGC -XX:ParallelGCThreads=20  
       -XX:+UseParallelGC：选择垃圾收集器为并行收集器。此配置仅对年轻代有效。即上述配置下，年轻代使用并发收集，而年老代仍旧使用串行收集。  
       -XX:ParallelGCThreads=20：配置并行收集器的线程数，即：同时多少个线程一起进行垃圾回收。此值最好配置与处理器数目相等。
     + java -Xmx3550m -Xms3550m -Xmn2g -Xss128k -XX:+UseParallelGC -XX:ParallelGCThreads=20 -XX:+UseParallelOldGC  
       -XX:+UseParallelOldGC：配置年老代垃圾收集方式为并行收集。JDK6.0支持对年老代并行收集。
     + java -Xmx3550m -Xms3550m -Xmn2g -Xss128k -XX:+UseParallelGC  -XX:MaxGCPauseMillis=100  
       -XX:MaxGCPauseMillis=100:设置每次年轻代垃圾回收的最长时间，如果无法满足此时间，JVM会自动调整年轻代大小，以满足此值。
     + java -Xmx3550m -Xms3550m -Xmn2g -Xss128k -XX:+UseParallelGC  -XX:MaxGCPauseMillis=100 -XX:+UseAdaptiveSizePolicy  
       -XX:+UseAdaptiveSizePolicy：设置此选项后，并行收集器会自动选择年轻代区大小和相应的Survivor区比例，以达到目标系统规定的最低相应时间或者收集频率等，此值建议使用并行收集器时，一直打开。
   * 响应时间优先的并发收集器  
     如上文所述，并发收集器主要是保证系统的响应时间，减少垃圾收集时的停顿时间。适用于应用服务器、电信领域等。  
     典型配置：
     + java -Xmx3550m -Xms3550m -Xmn2g -Xss128k -XX:ParallelGCThreads=20 -XX:+UseConcMarkSweepGC -XX:+UseParNewGC  
       -XX:+UseConcMarkSweepGC：设置年老代为并发收集。测试中配置这个以后，-XX:NewRatio=4的配置失效了，原因不明。所以，此时年轻代大小最好用-Xmn设置。  
       -XX:+UseParNewGC:设置年轻代为并行收集。可与CMS收集同时使用。JDK5.0以上，JVM会根据系统配置自行设置，所以无需再设置此值。
     + java -Xmx3550m -Xms3550m -Xmn2g -Xss128k -XX:+UseConcMarkSweepGC -XX:CMSFullGCsBeforeCompaction=5 -XX:+UseCMSCompactAtFullCollection  
       -XX:CMSFullGCsBeforeCompaction：由于并发收集器不对内存空间进行压缩、整理，所以运行一段时间以后会产生“碎片”，使得运行效率降低。此值设置运行多少次GC以后对内存空间进行压缩、整理。  
       -XX:+UseCMSCompactAtFullCollection：打开对年老代的压缩。可能会影响性能，但是可以消除碎片
3. 辅助信息  
   JVM提供了大量命令行参数，打印信息，供调试使用。主要有以下一些：
   * -XX:+PrintGC  
     输出形式：[GC 118250K->113543K(130112K), 0.0094143 secs]

                [Full GC 121376K->10414K(130112K), 0.0650971 secs]

* + -XX:+PrintGCDetails  
    输出形式：[GC [DefNew: 8614K->781K(9088K), 0.0123035 secs] 118250K->113543K(130112K), 0.0124633 secs]

                [GC [DefNew: 8614K->8614K(9088K), 0.0000665 secs][Tenured: 112761K->10414K(121024K), 0.0433488 secs] 121376K->10414K(130112K), 0.0436268 secs]

* + -XX:+PrintGCTimeStamps -XX:+PrintGC：PrintGCTimeStamps可与上面两个混合使用  
    输出形式：11.851: [GC 98328K->93620K(130112K), 0.0082960 secs]
  + -XX:+PrintGCApplicationConcurrentTime:打印每次垃圾回收前，程序未中断的执行时间。可与上面混合使用  
    输出形式：Application time: 0.5291524 seconds
  + -XX:+PrintGCApplicationStoppedTime：打印垃圾回收期间程序暂停的时间。可与上面混合使用  
    输出形式：Total time for which application threads were stopped: 0.0468229 seconds
  + -XX:PrintHeapAtGC:打印GC前后的详细堆栈信息  
    输出形式：  
    34.702: [GC {Heap before gc invocations=7:  
     def new generation   total 55296K, used 52568K [0x1ebd0000, 0x227d0000, 0x227d0000)  
    eden space 49152K,  99% used [0x1ebd0000, 0x21bce430, 0x21bd0000)  
    from space 6144K,  55% used [0x221d0000, 0x22527e10, 0x227d0000)  
      to   space 6144K,   0% used [0x21bd0000, 0x21bd0000, 0x221d0000)  
     tenured generation   total 69632K, used 2696K [0x227d0000, 0x26bd0000, 0x26bd0000)  
    the space 69632K,   3% used [0x227d0000, 0x22a720f8, 0x22a72200, 0x26bd0000)  
     compacting perm gen  total 8192K, used 2898K [0x26bd0000, 0x273d0000, 0x2abd0000)  
       the space 8192K,  35% used [0x26bd0000, 0x26ea4ba8, 0x26ea4c00, 0x273d0000)  
        ro space 8192K,  66% used [0x2abd0000, 0x2b12bcc0, 0x2b12be00, 0x2b3d0000)  
        rw space 12288K,  46% used [0x2b3d0000, 0x2b972060, 0x2b972200, 0x2bfd0000)  
    34.735: [DefNew: 52568K->3433K(55296K), 0.0072126 secs] 55264K->6615K(124928K)Heap after gc invocations=8:  
     def new generation   total 55296K, used 3433K [0x1ebd0000, 0x227d0000, 0x227d0000)  
    eden space 49152K,   0% used [0x1ebd0000, 0x1ebd0000, 0x21bd0000)  
      from space 6144K,  55% used [0x21bd0000, 0x21f2a5e8, 0x221d0000)  
      to   space 6144K,   0% used [0x221d0000, 0x221d0000, 0x227d0000)  
     tenured generation   total 69632K, used 3182K [0x227d0000, 0x26bd0000, 0x26bd0000)  
    the space 69632K,   4% used [0x227d0000, 0x22aeb958, 0x22aeba00, 0x26bd0000)  
     compacting perm gen  total 8192K, used 2898K [0x26bd0000, 0x273d0000, 0x2abd0000)  
       the space 8192K,  35% used [0x26bd0000, 0x26ea4ba8, 0x26ea4c00, 0x273d0000)  
        ro space 8192K,  66% used [0x2abd0000, 0x2b12bcc0, 0x2b12be00, 0x2b3d0000)  
        rw space 12288K,  46% used [0x2b3d0000, 0x2b972060, 0x2b972200, 0x2bfd0000)  
    }  
    , 0.0757599 secs]
  + -Xloggc:filename:与上面几个配合使用，把相关日志信息记录到文件以便分析。

1. 常见配置汇总
   * 堆设置
     + -Xms:初始堆大小
     + -Xmx:最大堆大小
     + -XX:NewSize=n:设置年轻代大小
     + -XX:NewRatio=n:设置年轻代和年老代的比值。如:为3，表示年轻代与年老代比值为1：3，年轻代占整个年轻代年老代和的1/4
     + -XX:SurvivorRatio=n:年轻代中Eden区与两个Survivor区的比值。注意Survivor区有两个。如：3，表示Eden：Survivor=3：2，一个Survivor区占整个年轻代的1/5
     + -XX:MaxPermSize=n:设置持久代大小
   * 收集器设置
     + -XX:+UseSerialGC:设置串行收集器
     + -XX:+UseParallelGC:设置并行收集器
     + -XX:+UseParalledlOldGC:设置并行年老代收集器
     + -XX:+UseConcMarkSweepGC:设置并发收集器
   * 垃圾回收统计信息
     + -XX:+PrintGC
     + -XX:+PrintGCDetails
     + -XX:+PrintGCTimeStamps
     + -Xloggc:filename
   * 并行收集器设置
     + -XX:ParallelGCThreads=n:设置并行收集器收集时使用的CPU数。并行收集线程数。
     + -XX:MaxGCPauseMillis=n:设置并行收集最大暂停时间
     + -XX:GCTimeRatio=n:设置垃圾回收时间占程序运行时间的百分比。公式为1/(1+n)
   * 并发收集器设置
     + -XX:+CMSIncrementalMode:设置为增量模式。适用于单CPU情况。
     + -XX:ParallelGCThreads=n:设置并发收集器年轻代收集方式为并行收集时，使用的CPU数。并行收集线程数。

四、调优总结

1. 年轻代大小选择
   * 响应时间优先的应用：尽可能设大，直到接近系统的最低响应时间限制（根据实际情况选择）。在此种情况下，年轻代收集发生的频率也是最小的。同时，减少到达年老代的对象。
   * 吞吐量优先的应用：尽可能的设置大，可能到达Gbit的程度。因为对响应时间没有要求，垃圾收集可以并行进行，一般适合8CPU以上的应用。
2. 年老代大小选择
   * 响应时间优先的应用：年老代使用并发收集器，所以其大小需要小心设置，一般要考虑并发会话率和会话持续时间等一些参数。如果堆设置小了，可以会造成内存碎片、高回收频率以及应用暂停而使用传统的标记清除方式；如果堆大了，则需要较长的收集时间。最优化的方案，一般需要参考以下数据获得：
     + 并发垃圾收集信息
     + 持久代并发收集次数
     + 传统GC信息
     + 花在年轻代和年老代回收上的时间比例

减少年轻代和年老代花费的时间，一般会提高应用的效率

* + 吞吐量优先的应用：一般吞吐量优先的应用都有一个很大的年轻代和一个较小的年老代。原因是，这样可以尽可能回收掉大部分短期对象，减少中期的对象，而年老代尽存放长期存活对象。

1. 较小堆引起的碎片问题  
   因为年老代的并发收集器使用标 记、清除算法，所以不会对堆进行压缩。当收集器回收时，他会把相邻的空间进行合并，这样可以分配给较大的对象。但是，当堆空间较小时，运行一段时间以后， 就会出现“碎片”，如果并发收集器找不到足够的空间，那么并发收集器将会停止，然后使用传统的标记、清除方式进行回收。如果出现“碎片”，可能需要进行如 下配置：
   * -XX:+UseCMSCompactAtFullCollection：使用并发收集器时，开启对年老代的压缩。
   * -XX:CMSFullGCsBeforeCompaction=0：上面配置开启的情况下，这里设置多少次Full GC后，对年老代进行压缩