https://www.cnblogs.com/redcreen/archive/2011/05/04/2037057.html

不管是YGC还是Full GC,GC过程中都会对导致程序运行中中断,正确的选择[不同的GC策略](http://www.cnblogs.com/redcreen/archive/2011/05/04/2037029.html),调整JVM、GC的参数，可以极大的减少由于GC工作，而导致的程序运行中断方面的问题，进而适当的提高Java程序的工作效率。但是调整GC是以个极为复杂的过程，由于各个程序具备不同的特点，如：web和GUI程序就有很大区别（Web可以适当的停顿，但GUI停顿是客户无法接受的），而且由于跑在各个机器上的配置不同（主要cup个数，内存不同），所以使用的GC种类也会不同(如何选择见[GC种类及如何选择](http://www.cnblogs.com/redcreen/archive/2011/05/04/2037029.html))。本文将注重介绍JVM、GC的一些重要参数的设置来提高系统的性能。

       JVM内存组成及GC相关内容请见之前的文章:[JVM内存组成](http://www.cnblogs.com/redcreen/archive/2011/05/04/2036387.html) [GC策略&内存申请](http://www.cnblogs.com/redcreen/archive/2011/05/04/2037056.html)。

**JVM参数的含义** 实例见[实例分析](http://www.cnblogs.com/redcreen/archive/2011/05/05/2038331.html)

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **含义** | **默认值** |  |
| -Xms | 初始堆大小 | 物理内存的1/64(<1GB) | 默认(MinHeapFreeRatio参数可以调整)空余堆内存小于40%时，JVM就会增大堆直到-Xmx的最大限制. |
| -Xmx | 最大堆大小 | 物理内存的1/4(<1GB) | 默认(MaxHeapFreeRatio参数可以调整)空余堆内存大于70%时，JVM会减少堆直到 -Xms的最小限制 |
| -Xmn | 年轻代大小(1.4or lator) |  | **注意**：此处的大小是（eden+ 2 survivor space).与jmap -heap中显示的New gen是不同的。 整个堆大小=年轻代大小 + 年老代大小 + 持久代大小. 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 |
| -XX:NewSize | 设置年轻代大小(for 1.3/1.4) |  |  |
| -XX:MaxNewSize | 年轻代最大值(for 1.3/1.4) |  |  |
| -XX:PermSize | 设置持久代(perm gen)初始值 | 物理内存的1/64 |  |
| -XX:MaxPermSize | 设置持久代最大值 | 物理内存的1/4 |  |
| -Xss | 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 一般小的应用， 如果栈不是很深， 应该是128k够用的 大的应用建议使用256k。这个选项对性能影响比较大，需要严格的测试。（校长） 和threadstacksize选项解释很类似,官方文档似乎没有解释,在论坛中有这样一句话:"” -Xss is translated in a VM flag named ThreadStackSize” 一般设置这个值就可以了。 |
| -*XX:ThreadStackSize* | Thread Stack Size |  | (0 means use default stack size) [Sparc: 512; Solaris x86: 320 (was 256 prior in 5.0 and earlier); Sparc 64 bit: 1024; Linux amd64: 1024 (was 0 in 5.0 and earlier); all others 0.] |
| -XX:NewRatio | 年轻代(包括Eden和两个Survivor区)与年老代的比值(除去持久代) |  | -XX:NewRatio=4表示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 Xms=Xmx并且设置了Xmn的情况下，该参数不需要进行设置。 |
| -XX:SurvivorRatio | Eden区与Survivor区的大小比值 |  | 设置为8,则两个Survivor区与一个Eden区的比值为2:8,一个Survivor区占整个年轻代的1/10 |
| -XX:LargePageSizeInBytes | 内存页的大小不可设置过大， 会影响Perm的大小 |  | =128m |
| -XX:+UseFastAccessorMethods | 原始类型的快速优化 |  |  |
| -XX:+DisableExplicitGC | 关闭System.gc() |  | 这个参数需要严格的测试 |
| -XX:MaxTenuringThreshold | 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率 该参数只有在串行GC时才有效. |
| -XX:+AggressiveOpts | 加快编译 |  |  |
| -XX:+UseBiasedLocking | 锁机制的性能改善 |  |  |
| -Xnoclassgc | 禁用垃圾回收 |  |  |
| -XX:SoftRefLRUPolicyMSPerMB | 每兆堆空闲空间中SoftReference的存活时间 | 1s | softly reachable objects will remain alive for some amount of time after the last time they were referenced. The default value is one second of lifetime per free megabyte in the heap |
| -XX:PretenureSizeThreshold | 对象超过多大是直接在旧生代分配 | 0 | 单位字节 新生代采用Parallel Scavenge GC时无效 另一种直接在旧生代分配的情况是大的数组对象,且数组中无外部引用对象. |
| -XX:TLABWasteTargetPercent | TLAB占eden区的百分比 | 1% |  |
| -XX:+*CollectGen0First* | FullGC时是否先YGC | false |  |

**并行收集器相关参数**

|  |  |  |  |
| --- | --- | --- | --- |
| -XX:+UseParallelGC | Full GC采用parallel MSC (此项待验证) |  | 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.(此项待验证) |
| -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 |
| -XX:ParallelGCThreads | 并行收集器的线程数 |  | 此值最好配置与处理器数目相等 同样适用于CMS |
| -XX:+UseParallelOldGC | 年老代垃圾收集方式为并行收集(Parallel Compacting) |  | 这个是JAVA 6出现的参数选项 |
| -XX:MaxGCPauseMillis | 每次年轻代垃圾回收的最长时间(最大暂停时间) |  | 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. |
| -XX:+UseAdaptiveSizePolicy | 自动选择年轻代区大小和相应的Survivor区比例 |  | 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. |
| -XX:GCTimeRatio | 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) |
| -XX:+*ScavengeBeforeFullGC* | Full GC前调用YGC | true | Do young generation GC prior to a full GC. (Introduced in 1.4.1.) |

**CMS相关参数**

|  |  |  |  |
| --- | --- | --- | --- |
| -XX:+UseConcMarkSweepGC | 使用CMS内存收集 |  | 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:+AggressiveHeap |  |  | 试图是使用大量的物理内存 长时间大内存使用的优化，能检查计算资源（内存， 处理器数量） 至少需要256MB内存 大量的CPU／内存， （在1.4.1在4CPU的机器上已经显示有提升） |
| -XX:CMSFullGCsBeforeCompaction | 多少次后进行内存压缩 |  | 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:+CMSParallelRemarkEnabled | 降低标记停顿 |  |  |
| -XX+UseCMSCompactAtFullCollection | 在FULL GC的时候， 对年老代的压缩 |  | CMS是不会移动内存的， 因此， 这个非常容易产生碎片， 导致内存不够用， 因此， 内存的压缩这个时候就会被启用。 增加这个参数是个好习惯。 可能会影响性能,但是可以消除碎片 |
| -XX:+UseCMSInitiatingOccupancyOnly | 使用手动定义初始化定义开始CMS收集 |  | 禁止hostspot自行触发CMS GC |
| -XX:CMSInitiatingOccupancyFraction=70 | 使用cms作为垃圾回收 使用70％后开始CMS收集 | 92 | 为了保证不出现promotion failed(见下面介绍)错误,该值的设置需要满足以下公式[**CMSInitiatingOccupancyFraction计算公式**](https://www.cnblogs.com/redcreen/archive/2011/05/04/2037057.html#CMSInitiatingOccupancyFraction_value) |
| -XX:CMSInitiatingPermOccupancyFraction | 设置Perm Gen使用到达多少比率时触发 | 92 |  |
| -XX:+CMSIncrementalMode | 设置为增量模式 |  | 用于单CPU情况 |
| -XX:+CMSClassUnloadingEnabled |  |  |  |

**辅助信息**

|  |  |  |  |
| --- | --- | --- | --- |
| -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-XX:+PrintGC -XX:+PrintGCDetails混合使用 输出形式:11.851: [GC 98328K->93620K(130112K), 0.0082960 secs] |
| -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:+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:+PrintHeapAtGC | 打印GC前后的详细堆栈信息 |  |  |
| -Xloggc:filename | 把相关日志信息记录到文件以便分析. 与上面几个配合使用 |  |  |
| -XX:+PrintClassHistogram | garbage collects before printing the histogram. |  |  |
| -XX:+PrintTLAB | 查看TLAB空间的使用情况 |  |  |
| XX:+PrintTenuringDistribution | 查看每次minor GC后新的存活周期的阈值 |  | Desired survivor size 1048576 bytes, new threshold 7 (max 15) new threshold 7即标识新的存活周期的阈值为7。 |

**GC性能方面的考虑**

       对于GC的性能主要有2个方面的指标：吞吐量throughput（工作时间不算gc的时间占总的时间比）和暂停pause（gc发生时app对外显示的无法响应）。

1. Total Heap

       默认情况下，vm会增加/减少heap大小以维持free space在整个vm中占的比例，这个比例由MinHeapFreeRatio和MaxHeapFreeRatio指定。

一般而言，server端的app会有以下规则：

* 对vm分配尽可能多的memory；
* 将Xms和Xmx设为一样的值。如果虚拟机启动时设置使用的内存比较小，这个时候又需要初始化很多对象，虚拟机就必须重复地增加内存。
* 处理器核数增加，内存也跟着增大。

2. The Young Generation

       另外一个对于app流畅性运行影响的因素是young generation的大小。young generation越大，minor collection越少；但是在固定heap size情况下，更大的young generation就意味着小的tenured generation，就意味着更多的major collection(major collection会引发minor collection)。

       NewRatio反映的是young和tenured generation的大小比例。NewSize和MaxNewSize反映的是young generation大小的下限和上限，将这两个值设为一样就固定了young generation的大小（同Xms和Xmx设为一样）。

       如果希望，SurvivorRatio也可以优化survivor的大小，不过这对于性能的影响不是很大。SurvivorRatio是eden和survior大小比例。

一般而言，server端的app会有以下规则：

* 首先决定能分配给vm的最大的heap size，然后设定最佳的young generation的大小；
* 如果heap size固定后，增加young generation的大小意味着减小tenured generation大小。让tenured generation在任何时候够大，能够容纳所有live的data（留10%-20%的空余）。

**经验&&规则**

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.
   * 避免设置过小.当新生代设置过小时会导致:1.YGC次数更加频繁 2.可能导致YGC对象直接进入旧生代,如果此时旧生代满了,会触发FGC.
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。
   * 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.
3. 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
4. 用64位操作系统，Linux下64位的jdk比32位jdk要慢一些，但是吃得内存更多，吞吐量更大
5. XMX和XMS设置一样大，MaxPermSize和MinPermSize设置一样大，这样可以减轻伸缩堆大小带来的压力
6. 使用CMS的好处是用尽量少的新生代，经验值是128M－256M， 然后老生代利用CMS并行收集， 这样能保证系统低延迟的吞吐效率。 实际上cms的收集停顿时间非常的短，2G的内存， 大约20－80ms的应用程序停顿时间
7. 系统停顿的时候可能是GC的问题也可能是程序的问题，多用jmap和jstack查看，或者killall -3 java，然后查看java控制台日志，能看出很多问题。(相关工具的使用方法将在后面的blog中介绍)
8. 仔细了解自己的应用，如果用了缓存，那么年老代应该大一些，缓存的HashMap不应该无限制长，建议采用LRU算法的Map做缓存，LRUMap的最大长度也要根据实际情况设定。
9. 采用并发回收时，年轻代小一点，年老代要大，因为年老大用的是并发回收，即使时间长点也不会影响其他程序继续运行，网站不会停顿
10. JVM参数的设置(特别是 –Xmx –Xms –Xmn -XX:SurvivorRatio  -XX:MaxTenuringThreshold等参数的设置没有一个固定的公式，需要根据PV old区实际数据 YGC次数等多方面来衡量。为了避免promotion faild可能会导致xmn设置偏小，也意味着YGC的次数会增多，处理并发访问的能力下降等问题。每个参数的调整都需要经过详细的性能测试，才能找到特定应用的最佳配置。

**promotion failed:**

垃圾回收时promotion failed是个很头痛的问题，一般可能是两种原因产生，第一个原因是救助空间不够，救助空间里的对象还不应该被移动到年老代，但年轻代又有很多对象需要放入救助空间；第二个原因是年老代没有足够的空间接纳来自年轻代的对象；这两种情况都会转向Full GC，网站停顿时间较长。

解决方方案一：

*第一个原因我的最终解决办法是去掉救助空间，设置-XX:SurvivorRatio=65536 -XX:MaxTenuringThreshold=0即可，第二个原因我的解决办法是设置CMSInitiatingOccupancyFraction为某个值（假设70），这样年老代空间到70%时就开始执行CMS，年老代有足够的空间接纳来自年轻代的对象。*

解决方案一的改进方案：

*又有改进了，上面方法不太好，因为没有用到救助空间，所以年老代容易满，CMS执行会比较频繁。我改善了一下，还是用救助空间，但是把救助空间加大，这样也不会有promotion failed。具体操作上，32位Linux和64位Linux好像不一样，64位系统似乎只要配置MaxTenuringThreshold参数，CMS还是有暂停。为了解决暂停问题和promotion failed问题，最后我设置-XX:SurvivorRatio=1 ，并把MaxTenuringThreshold去掉，这样即没有暂停又不会有promotoin failed，而且更重要的是，年老代和永久代上升非常慢（因为好多对象到不了年老代就被回收了），所以CMS执行频率非常低，好几个小时才执行一次，这样，服务器都不用重启了。*

-Xmx4000M -Xms4000M -Xmn600M -XX:PermSize=500M -XX:MaxPermSize=500M -Xss256K -XX:+DisableExplicitGC -XX:SurvivorRatio=1 -XX:+UseConcMarkSweepGC -XX:+UseParNewGC -XX:+CMSParallelRemarkEnabled -XX:+UseCMSCompactAtFullCollection -XX:CMSFullGCsBeforeCompaction=0 -XX:+CMSClassUnloadingEnabled -XX:LargePageSizeInBytes=128M -XX:+UseFastAccessorMethods -XX:+UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=80 -XX:SoftRefLRUPolicyMSPerMB=0 -XX:+PrintClassHistogram -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+PrintHeapAtGC -Xloggc:log/gc.log

**CMSInitiatingOccupancyFraction值与Xmn的关系公式**

上面介绍了promontion faild产生的原因是EDEN空间不足的情况下将EDEN与From survivor中的存活对象存入To survivor区时,To survivor区的空间不足，再次晋升到old gen区，而old gen区内存也不够的情况下产生了promontion faild从而导致full gc.那可以推断出：eden+from survivor < old gen区剩余内存时，不会出现promontion faild的情况，即：  
(Xmx-Xmn)\*(1-CMSInitiatingOccupancyFraction/100)>=(Xmn-Xmn/(SurvivorRatior+2))  进而推断出：

CMSInitiatingOccupancyFraction <=((Xmx-Xmn)-(Xmn-Xmn/(SurvivorRatior+2)))/(Xmx-Xmn)\*100

例如：

当xmx=128 xmn=36 SurvivorRatior=1时 CMSInitiatingOccupancyFraction<=((128.0-36)-(36-36/(1+2)))/(128-36)\*100 =73.913

当xmx=128 xmn=24 SurvivorRatior=1时 CMSInitiatingOccupancyFraction<=((128.0-24)-(24-24/(1+2)))/(128-24)\*100=84.615…

当xmx=3000 xmn=600 SurvivorRatior=1时  CMSInitiatingOccupancyFraction<=((3000.0-600)-(600-600/(1+2)))/(3000-600)\*100=83.33

CMSInitiatingOccupancyFraction低于70% 需要调整xmn或SurvivorRatior值。

令：

[网上一童鞋](http://bbs.weblogicfans.net/archiver/tid-2835.html)推断出的公式是：:(Xmx-Xmn)\*(100-CMSInitiatingOccupancyFraction)/100>=Xmn 这个公式个人认为不是很严谨，在内存小的时候会影响xmn的计算。

关于实际环境的GC参数配置见:[实例分析](http://www.cnblogs.com/redcreen/archive/2011/05/05/2038331.html)   [监测工具见JVM监测](http://www.cnblogs.com/redcreen/archive/2011/05/09/2040977.html)

参考：

JAVA HOTSPOT VM（<http://www.helloying.com/blog/archives/164>）

[JVM 几个重要的参数](http://www.iteye.com/wiki/jvm/2870-JVM) (校长)

[java jvm 参数 -Xms -Xmx -Xmn -Xss 调优总结](http://hi.baidu.com/sdausea/blog/item/c599ef13fcd3a7dbf6039e12.html)

[Java HotSpot VM Options](http://www.oracle.com/technetwork/java/javase/tech/vmoptions-jsp-140102.html)

<http://bbs.weblogicfans.net/archiver/tid-2835.html>

[Frequently Asked Questions About the Java HotSpot VM](http://www.oracle.com/technetwork/java/hotspotfaq-138619.html)

[Java SE HotSpot at a Glance](http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136373.html)

[Java性能调优笔记](http://blog.csdn.net/yang_net/archive/2010/08/22/5830820.aspx)(内附测试例子 很有用)

[说说MaxTenuringThreshold这个参数](http://blog.bluedavy.com/?p=70)

相关文章推荐:

[GC调优方法总结](http://blog.csdn.net/pigeon21/archive/2011/01/27/6166217.aspx)

[Java 6 JVM参数选项大全（中文版）](http://kenwublog.com/docs/java6-jvm-options-chinese-edition.htm)