链滴

JVM（Java 虚拟机）优化大全和案例实战

作者：zhaoyong
原文链接：https://ld246.com/article/1487818185702
来源网站：链滴

<h2 id="堆内存设置">堆内存设置</h2>
<h2 id="原理">原理</h2>
JVM 堆内存分为 2 块：Permanent Space 和 Heap Space。

Permanent 即 持久代（Permanent Generation），主要存放的是 Java 类定义信息，与垃圾收器要收集的 Java 对象关系不大。
Heap = { Old + NEW = {Eden, from, to} }，Old 即 年老代（Old Generation），New 即 年代（Young Generation）。年老代和年轻代的划分对垃圾收集影响比较大。

<h3 id="年轻代">年轻代</h3>
所有新生成的对象首先都是放在年轻代。年轻代的目标就是尽可能快速的收集掉那些生命周期短对象。年轻代一般分 3 个区，1 个 Eden 区，2 个 Survivor 区（from 和 to）。
大部分对象在 Eden 区中生成。当 Eden 区满时，还存活的对象将被复制到 Survivor 区（两个的一个），当一个 Survivor 区满时，此区的存活对象将被复制到另外一个 Survivor 区，当另一个 Survivor 区也满了的时候，从前一个 Survivor 区复制过来的并且此时还存活的对象，将可能被复制到年代。
2 个 Survivor 区是对称的，没有先后关系，所以同一个 Survivor 区中可能同时存在从 Eden 区制过来对象，和从另一个 Survivor 区复制过来的对象；而复制到年老区的只有从另一个 Survivor 区来的对象。而且，因为需要交换的原因，Survivor 区至少有一个是空的。特殊情况下，根据程序需要，Survivor 区是可以配置为多个的（多于 2 个），这样可以增加对象在年轻中的存在时间，减少被放到年老代的可能。
针对年轻代的垃圾回收即 Young GC。
<h3 id="年老代">年老代</h3>
在年轻代中经历了 N 次（可配置）垃圾回收后仍然存活的对象，就会被复制到年老代中。因此可以认为年老代中存放的都是一些生命周期较长的对象。
针对年老代的垃圾回收即 Full GC。
<h3 id="持久代">持久代</h3>
用于存放静态类型数据，如 Java Class, Method 等。持久代对垃圾回收没有显著影响。但是有些应用可能动态生成或调用一些 Class，如 hibernate CGLib 等，在这种候往往需要设置一个比较大的持久代空间来存放这些运行过程中动态增加的类型。
所以，当一组对象生成时，内存申请过程如下：

JVM 会试图为相关 Java 对象在年轻代的 Eden 区中初始化一块内存区域。
当 Eden 区空间足够时，内存申请结束。否则执行下一步。
JVM 试图释放在 Eden 区中所有不活跃的对象（Young GC）。释放后若 Eden 空间仍然不足以入新对象，JVM 则试图将部分 Eden 区中活跃对象放入 Survivor 区。
Survivor 区被用来作为 Eden 区及年老代的中间交换区域。当年老代空间足够时，Survivor 区存活了一定次数的对象会被移到年老代。
当年老代空间不够时，JVM 会在年老代进行完全的垃圾回收（Full GC）。
Full GC 后，若 Survivor 区及年老代仍然无法存放从 Eden 区复制过来的对象，则会导致 JVM 法在 Eden 区为新生成的对象申请内存，即出现“Out of Memory”。

OOM（“Out of Memory”）异常一般主要有如下 2 种原因：
1. 年老代溢出，表现为：java.lang.OutOfMemoryError:Javaheapspace
这是最常见的情况，产生的原因可能是：设置的内存参数 Xmx 过小或程序的内存泄露及使用不问题。
例如循环上万次的字符串处理、创建上千万个对象、在一段代码内申请上百 M 甚至上 G 的内存还有的时候虽然不会报内存溢出，却会使系统不间断的垃圾回收，也无法处理其它请求。这种情况下了检查程序、打印堆内存等方法排查，还可以借助一些内存分析工具，比如 MAT 就很不错。
2. 持久代溢出，表现为：java.lang.OutOfMemoryError:PermGenspace
通常由于持久代设置过小，动态加载了大量 Java 类而导致溢出，解决办法唯有将参数 -XX:MaxP

rmSize 调大（一般 256m 能满足绝大多数应用程序需求）。将部分 Java 类放到容器共享区（例如 Tmcat share lib）去加载的办法也是一个思路，但前提是容器里部署了多个应用，且这些应用有大量共享类库。
<h2 id="参数说明">参数说明</h2>

-Xmx3550m：设置 JVM 最大堆内存为 3550M。

-Xms3550m：设置 JVM 初始堆内存为 3550M。此值可以设置与-Xmx 同，以避免每次垃圾回收完成后 JVM 重新分配内存。

-Xss128k：设置每个线程的栈大小。JDK5.0 以后每个线程栈大小为 1M，之前每个线程栈大小为 256K。应当根据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 左右需要注意的是：当这个值被设置的较大（例如 >2MB）时将会在很大程度上降低系统的性能。</p

-Xmn2g：设置年轻代大小为 2G。在整个堆内存大小确定的情况下，增大轻代将会减小年老代，反之亦然。此值关系到 JVM 垃圾回收，对系统性能影响较大，官方推荐配置整个堆大小的 3/8。

-XX:NewSize=1024m：设置年轻代初始值为 1024M。

-XX:MaxNewSize=1024m：设置年轻代最大值为 1024M。

-XX:PermSize=256m：设置持久代初始值为 256M。

-XX:MaxPermSize=256m：设置持久代最大值为 256M。

-XX:NewRatio=4：设置年轻代（包括 1 个 Eden 和 2 个 Survivor 区）与年老代的比值。表示轻代比年老代为 1:4。

-XX:SurvivorRatio=4：设置年轻代中 Eden 区与 Survivor 区的比值。表示 2 个 Survivor 区（JVM 堆内存年轻代中默认有 2 个大小相等的 Survivor 区）与 1 个 Eden 区的比为 2:4，即 1 个 Survivor 区占整个年轻代大小的 1/6。

-XX:MaxTenuringThreshold=7：表示一个对象如果在 Survivor 区（救助空间）移动了 7 次还有被垃圾回收就进入年老代。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，对于需要大量常驻内存的应用，这样做可以提高效率。如果将此值设置为一个较大值，则年轻代对会在 Survivor 区进行多次复制，这样可以增加对象在年轻代存活时间，增加对象在年轻代被垃圾回的概率，减少 Full GC 的频率，这样做可以在某种程度上提高服务稳定性。

<h2 id="疑问解答">疑问解答</h2>

-Xmn，-XX:NewSize/-XX:MaxNewSize，-XX:NewRatio 3 组参数都可以影响年轻代的大小，合使用的情况下，优先级是什么？
如下：

高优先级：-XX:NewSize/-XX:MaxNewSize
中优先级：-Xmn（默认等效 -Xmn=-XX:NewSize=-XX:MaxNewSize=?）
低优先级：-XX:NewRatio

推荐使用-Xmn 参数，原因是这个参数简洁，相当于一次设定 NewSize/MaxNewSIze，而且两相等，适用于生产环境。-Xmn 配合 -Xms/-Xmx，即可将堆内存布局完成。
-Xmn 参数是在 JDK 1.4 开始支持。
<h2 id="垃圾回收器选择">垃圾回收器选择</h2>
JVM 给出了 3 种选择：串行收集器、并行收集器、并发收集器。串行收集器只适用于小数据量情况，所以生产环境的选择主要是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。JDK5.0 以后，JVM 会根据当前系统配置进行智能判断。
<h2 id="串行收集器">串行收集器</h2>

-XX:+UseSerialGC：设置串行收集器。

<h2 id="并行收集器-吞吐量优先-">并行收集器（吞吐量优先）</h2>

-XX:+UseParallelGC：设置为并行收集器。此配置仅对年轻代有效。即年轻代使用并行收集，年老代仍使用串行收集。
-XX:ParallelGCThreads=20：配置并行收集器的线程数，即：同时有多少个线程一起进行垃圾收。此值建议配置与 CPU 数目相等。
-XX:+UseParallelOldGC：配置年老代垃圾收集方式为并行收集。JDK6.0 开始支持对年老代并收集。
-XX:MaxGCPauseMillis=100：设置每次年轻代垃圾回收的最长时间（单位毫秒）。如果无法满此时间，JVM 会自动调整年轻代大小，以满足此时间。
-XX:+UseAdaptiveSizePolicy：设置此选项后，并行收集器会自动调整年轻代 Eden 区大小和 Survivor 区大小的比例，以达成目标系统规定的最低响应时间或者收集频率等指标。此参数建议在使用行收集器时，一直打开。

<h2 id="并发收集器-响应时间优先-">并发收集器（响应时间优先）</h2>

-XX:+UseConcMarkSweepGC：即 CMS 收集，设置年老代为并发收集。CMS 收集是 JDK1.4 期版本开始引入的新 GC 算法。它的主要适合场景是对响应时间的重要性需求大于对吞吐量的需求，够承受垃圾回收线程和应用线程共享 CPU 资源，并且应用中存在比较多的长生命周期对象。CMS 收的目标是尽量减少应用的暂停时间，减少 Full GC 发生的几率，利用和应用程序线程并发的垃圾回收程来标记清除年老代内存。

-XX:+UseParNewGC：设置年轻代为并发收集。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，所以无需再设置此参数。

-XX:CMSFullGCsBeforeCompaction=0：由于并发收集器不对内存空间进行压缩和整理，所以行一段时间并行收集以后会产生内存碎片，内存使用效率降低。此参数设置运行 0 次 Full GC 后对内空间进行压缩和整理，即每次 Full GC 后立刻开始压缩和整理内存。

-XX:+UseCMSCompactAtFullCollection：打开内存空间的压缩和整理，在 Full GC 后执行。

能会影响性能，但可以消除内存碎片。

-XX:+CMSIncrementalMode：设置为增量收集模式。一般适用于单 CPU 情况。

-XX:CMSInitiatingOccupancyFraction=70：表示年老代内存空间使用到 70% 时就开始执行 CS 收集，以确保年老代有足够的空间接纳来自年轻代的对象，避免 Full GC 的发生。

<h2 id="其它垃圾回收参数">其它垃圾回收参数</h2>

-XX:+ScavengeBeforeFullGC：年轻代 GC 优于 Full GC 执行。
-XX:-DisableExplicitGC：不响应 System.gc() 代码。
-XX:+UseThreadPriorities：启用本地线程优先级 API。即使 <code>java.lang.Thread.setPriority() </code> 生效，不启用则无效。
-XX:SoftRefLRUPolicyMSPerMB=0：软引用对象在最后一次被访问后能存活 0 毫秒（JVM 默为 1000 毫秒）。
-XX:TargetSurvivorRatio=90：允许 90% 的 Survivor 区被占用（JVM 默认为 50%）。提高对 Survivor 区的使用率。

<h2 id="辅助信息参数设置">辅助信息参数设置</h2>

-XX:-CITime：打印消耗在 JIT 编译的时间。
-XX:ErrorFile=./hs_err_pid.log：保存错误日志或数据到指定文件中。
-XX:HeapDumpPath=./java_pid.hprof：指定 Dump 堆内存时的路径。</i>
-XX:-HeapDumpOnOutOfMemoryError：当首次遭遇内存溢出时 Dump 出此时的内存。
-XX:OnError=";"：出现致命 ERROR 后运行自定义命令。
-XX:OnOutOfMemoryError=";"：当首次遭遇内存溢出时执行自定义命令。
-XX:-PrintClassHistogram：按下 Ctrl+Break 后打印堆内存中类实例的柱状信息，同 JDK 的 jmp -histo 命令。
-XX:-PrintConcurrentLocks：按下 Ctrl+Break 后打印线程栈中并发锁的相关信息，同 JDK 的 jack -l 命令。
-XX:-PrintCompilation：当一个方法被编译时打印相关信息。
-XX:-PrintGC：每次 GC 时打印相关信息。
-XX:-PrintGCDetails：每次 GC 时打印详细信息。
-XX:-PrintGCTimeStamps：打印每次 GC 的时间戳。
-XX:-TraceClassLoading：跟踪类的加载信息。
-XX:-TraceClassLoadingPreorder：跟踪被引用到的所有类的加载信息。
-XX:-TraceClassResolution：跟踪常量池。
-XX:-TraceClassUnloading：跟踪类的卸载信息。

<h2 id="关于参数名称等">关于参数名称等</h2>

标准参数（-），所有 JVM 都必须支持这些参数的功能，而且向后兼容；例如：

-client——设置 JVM 使用 Client 模式，特点是启动速度比较快，但运行性能和内存管理效率不高，通常用于客户端应用程序或开发调试；在 32 位环境下直接运行 Java 程默认启用该模式。

-server——设置 JVM 使 Server 模式，特点是启动速度比较慢，但运行性能和内存管理效率很高，适用于生产环境。在具有 64 位能力的 JDK 环境下默认启用该模式。

非标准参数（-X），默认 JVM 实现这些参数的功能，但是并不保证所有 JVM 实现都满足，且保证向后兼容；

非稳定参数（-XX），此类参数各个 JVM 实现会有所不同，将来可能会不被支持，需要慎重使；

<h2 id="JVM服务参数调优实战">JVM 服务参数调优实战</h2>
<h2 id="大型网站服务器案例">大型网站服务器案例</h2>
承受海量访问的动态 Web 应用
服务器配置：8 CPU, 8G MEM, JDK 1.6.X
参数方案：
-server -Xmx3550m -Xms3550m -Xmn1256m -Xss128k -XX:SurvivorRatio=6 -XX:MaxPerSize=256m -XX:ParallelGCThreads=8 -XX:MaxTenuringThreshold=0 -XX:+UseConcMarkSweeGC
调优说明：

-Xmx 与 -Xms 相同以避免 JVM 反复重新申请内存。-Xmx 的大小约等于系统内存大小的一半即充分利用系统资源，又给予系统安全运行的空间。

-Xmn1256m 设置年轻代大小为 1256MB。此值对系统性能影响较大，Sun 官方推荐配置年轻大小为整个堆的 3/8。

-Xss128k 设置较小的线程栈以支持创建更多的线程，支持海量访问，并提升系统性能。

-XX:SurvivorRatio=6 设置年轻代中 Eden 区与 Survivor 区的比值。系统默认是 8，根据经验置为 6，则 2 个 Survivor 区与 1 个 Eden 区的比值为 2:6，一个 Survivor 区占整个年轻代的 1/8。

-XX:ParallelGCThreads=8 配置并行收集器的线程数，即同时 8 个线程一起进行垃圾回收。此一般配置为与 CPU 数目相等。

-XX:MaxTenuringThreshold=0 设置垃圾最大年龄（在年轻代的存活次数）。如果设置为 0 的，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的存活时间，增加在年轻代即被回收的概率。根据被海量访问的动态 Web 应用之特点，其内存要么缓存起来以减少直接访问 DB，要么被快速回收以支持高并发海量请求，因此其内存对象在年轻代存多次意义不大，可以直接进入年老代，根据实际应用效果，在这里设置此值为 0。

-XX:+UseConcMarkSweepGC 设置年老代为并发收集。CMS（ConcMarkSweepGC）收集的标是尽量减少应用的暂停时间，减少 Full GC 发生的几率，利用和应用程序线程并发的垃圾回收线程

标记清除年老代内存，适用于应用中存在比较多的长生命周期对象的情况。

<h2 id="内部集成构建服务器案例">内部集成构建服务器案例</h2>
高性能数据处理的工具应用
服务器配置：1 CPU, 4G MEM, JDK 1.6.X
参数方案：
-server -XX:PermSize=196m -XX:MaxPermSize=196m -Xmn320m -Xms768m -Xmx1024

调优说明：

-XX:PermSize=196m -XX:MaxPermSize=196m 根据集成构建的特点，大规模的系统编译可能要加载大量的 Java 类到内存中，所以预先分配好大量的持久代内存是高效和必要的。

-Xmn320m 遵循年轻代大小为整个堆的 3/8 原则。

-Xms768m -Xmx1024m 根据系统大致能够承受的堆内存大小设置即可。

在 64 位服务器上运行应用程序，构建执行时，用 jmap -heap 11540 命令观察 JVM 堆内存状如下：
Attaching to process ID 11540, please wait...
Debugger attached successfully.
Server compiler detected.
JVM version is 20.12-b01
using thread-local object allocation.
Parallel GC with 4 thread(s)
Heap Configuration:
MinHeapFreeRatio = 40
MaxHeapFreeRatio = 70
MaxHeapSize = 1073741824 (1024.0MB)
NewSize = 335544320 (320.0MB)
MaxNewSize = 335544320 (320.0MB)
OldSize = 5439488 (5.1875MB)
NewRatio = 2
SurvivorRatio = 8
PermSize = 205520896 (196.0MB)
MaxPermSize = 205520896 (196.0MB)
Heap Usage:
PS Young Generation
Eden Space:
capacity = 255852544 (244.0MB)
used = 101395504 (96.69828796386719MB)
free = 154457040 (147.3017120361328MB)
39.63044588683081% used
From Space:
capacity = 34144256 (32.5625MB)
used = 33993968 (32.41917419433594MB)
free = 150288 (0.1433258056640625MB)
99.55984397492803% used
To Space:

capacity = 39845888 (38.0MB)
used   = 0 (0.0MB)
free   = 39845888 (38.0MB)
0.0% used
PS Old Generation
capacity = 469762048 (448.0MB)
used   = 44347696 (42.29325866699219MB)
free   = 425414352 (405.7067413330078MB)
9.440459523882184% used
PS Perm Generation
capacity = 205520896 (196.0MB)
used   = 85169496 (81.22396087646484MB)
free   = 120351400 (114.77603912353516MB)
41.440796365543285% used
结果是比较健康的。