http://www.oracle.com/technetwork/java/tuning-139912.html

**Java调优白皮书**

**表的内容**

1 [介绍](http://www.oracle.com/technetwork/java/tuning-139912.html#section1)

1.1 [目标](http://www.oracle.com/technetwork/java/tuning-139912.html#section1.1)   
1.2 [这是一个文档](http://www.oracle.com/technetwork/java/tuning-139912.html#section1.2)   
1.3 [如何使用本白皮书](http://www.oracle.com/technetwork/java/tuning-139912.html#section1.3)

2 [最佳实践](http://www.oracle.com/technetwork/java/tuning-139912.html#section2)

2.1 [使用最近Javaa¢释放](http://www.oracle.com/technetwork/java/tuning-139912.html#section2.1)   
2.2 [获得最新的Javaa¢更新发布](http://www.oracle.com/technetwork/java/tuning-139912.html#section2.2)   
2.3 [确保您的操作系统补丁程序是最新的](http://www.oracle.com/technetwork/java/tuning-139912.html#section2.3)   
2.4 [消除差异](http://www.oracle.com/technetwork/java/tuning-139912.html#section2.4)

3 [决策的数据](http://www.oracle.com/technetwork/java/tuning-139912.html#section3)

3.1 [谨防微基准测试!](http://www.oracle.com/technetwork/java/tuning-139912.html#section3.1)   
3.2 [使用统计数据](http://www.oracle.com/technetwork/java/tuning-139912.html#section3.2)   
3.3 [使用一个基准利用](http://www.oracle.com/technetwork/java/tuning-139912.html#section3.3)

4 [调优的想法](http://www.oracle.com/technetwork/java/tuning-139912.html#section4)

4.1 [一般调优指南](http://www.oracle.com/technetwork/java/tuning-139912.html#section4.1)

以下4.4.1一 [了解人体工程学的设置](http://www.oracle.com/technetwork/java/tuning-139912.html#section4.1.1)   
4.1.2一 [堆大小](http://www.oracle.com/technetwork/java/tuning-139912.html#section4.1.2)   
4.1.3一 [垃圾收集器的政策](http://www.oracle.com/technetwork/java/tuning-139912.html#section4.1.3)   
4.1.4一 [其他调优参数](http://www.oracle.com/technetwork/java/tuning-139912.html#section4.1.4)

4.2 [调优示例](http://www.oracle.com/technetwork/java/tuning-139912.html#section4.2)

4.2.1的准备 [调优示例1:调优的吞吐量](http://www.oracle.com/technetwork/java/tuning-139912.html#section4.2.1)   
4.2.2一 [调优示例2:并行年老代收集器](http://www.oracle.com/technetwork/java/tuning-139912.html#section4.2.2)   
4.2.3一 [调优示例3:尝试256 MB页](http://www.oracle.com/technetwork/java/tuning-139912.html#section4.2.3)   
4.2.4一 [调优示例4:尝试- xx:+ AggressiveOpts](http://www.oracle.com/technetwork/java/tuning-139912.html#section4.2.4)   
4.2.5一 [调优示例5:尝试偏向锁](http://www.oracle.com/technetwork/java/tuning-139912.html#section4.2.5)   
4.2.6一 [调优示例6:优化低暂停时间和高吞吐量](http://www.oracle.com/technetwork/java/tuning-139912.html#section4.2.6)   
4.2.7一 [调优示例7:尝试AggressiveOpts低暂停时间和高吞吐量](http://www.oracle.com/technetwork/java/tuning-139912.html#section4.2.7)

5 [监控和分析](http://www.oracle.com/technetwork/java/tuning-139912.html#section5)

5.1 [监控](http://www.oracle.com/technetwork/java/tuning-139912.html#section5.1)   
5.2 [分析](http://www.oracle.com/technetwork/java/tuning-139912.html#section5.2)

6 [编码性能](http://www.oracle.com/technetwork/java/tuning-139912.html#section6)   
7一个 [指针](http://www.oracle.com/technetwork/java/tuning-139912.html#section7)   
8 [反馈和Java性能社区](http://www.oracle.com/technetwork/java/tuning-139912.html#section8)

8.1 [反馈](http://www.oracle.com/technetwork/java/tuning-139912.html#section8.1)   
8.2 [Java性能社区](http://www.oracle.com/technetwork/java/tuning-139912.html#section8.2)

|  |
| --- |
|  |

**1介绍**

这个Java调优白皮书的目的是为Java性能调优信息作为参考,技术和指针。

**1.1目标**

本白皮书的目标是收集最佳实践和“如何”Java性能在一个地方。   
  
这个优化文档最初的目标是优化服务器应用程序在大型,多处理器服务器。 本文档的未来版本将探索类似桌面Java性能的建议。

**1.2生活这是一个文档**

这个白皮书已经写在太阳的最新进展Javaa¢HotSpota¢虚拟机。 因此本文将频繁的基础上演变反映最新的性能特点和新的最佳实践。

**1.3如何使用本白皮书**

本白皮书组织部分从最简单最容易的方法得到更好的Java性能逐步更复杂的优化和设计建议。   
  
开始 [最佳实践](http://www.oracle.com/technetwork/java/tuning-139912.html#section2) 确保你得到最好的Java性能可能之前你做任何调优。 再深入的性能调优的基本理解的正确方法 [决策的数据](http://www.oracle.com/technetwork/java/tuning-139912.html#section3) 。 只有在此基础上直接分析可以安全地继续探索 [调优的想法](http://www.oracle.com/technetwork/java/tuning-139912.html#section4) 为您的应用程序。 为什么会有“调优的想法”,而不是毯子的建议呢? 因为每个应用程序都是不同的,没有一组适合每一个部署环境的建议。 调优的想法部分不仅仅是为了给你的Java命令行选项还背景对他们意味着什么,当他们可能导致增加性能。   
  
甚至在调优过程中,但肯定像你想要性能更上一层楼有必要探索 [监控和分析](http://www.oracle.com/technetwork/java/tuning-139912.html#section5) 您的应用程序。 通过收集详细的数据在实际应用程序的性能可以微调命令行选项和有一个想法,把编码改进工作。 在节 [编码性能](http://www.oracle.com/technetwork/java/tuning-139912.html#section6) 我们将介绍Javaa¢API在设计应用程序时要考虑的性能。   
  
许多其他文件和网站将收集 [指针](http://www.oracle.com/technetwork/java/tuning-139912.html#section7) 部分。 我们鼓励你帮助继续使Java更快的通过 [反馈和Java性能社区](http://www.oracle.com/technetwork/java/tuning-139912.html#section8) 。 您的反馈将有助于使本白皮书更好。

|  |
| --- |
|  |

**2最佳实践**

有许多最佳实践与Java应用程序获得最佳性能。 这里有一些非常基本的实践可以使性能在开始调优的显著差异。

**2.1使用最近Javaa¢释放**

每个主要释放Javaa¢引入新功能,bug修复和性能增强。 在开始调优之前或者考虑应用程序级别变化利用新的语言特性仍然是值得升级到最新主要Javaa¢释放。 信息为什么升级到Java SE 5.0(老虎)是重要的请参见参考资料 [指针](http://www.oracle.com/technetwork/java/tuning-139912.html#section7) 部分。   
  
可以理解,这并不总是可能的某些应用程序,尤其是第三方ISV应用程序可能还没有提供支持的最新版本Javaa¢。 在这种情况下使用最近的 *更新* 版本支持您的应用程序(请鼓励你的ISV支持最新的Javaa¢版本!)。

**获得最新的2.2 Javaa¢更新发布**

对于每个主要Javaa¢释放“火车”(例如J2SE 1.3.1,J2SE 1.4.2,J2SE 5.0)太阳定期发布更新版本。 例如最近的更新版本的J2SE 5.0 *更新6* 或 [Java SE 1.5.0\_06](http://www.oracle.com/technetwork/java/javase/downloads/index-jdk5-jsp-142662.html) 。   
  
更新版本通常包括bug修复和性能改进。 通过部署最新的更新发布Javaa¢你将受益于最新最好的性能改进。

**确保您的操作系统补丁2.3是最新的**

尽管Java是跨平台确实依赖于底层操作系统,因此重要的是,操作系统基础Javaa¢尽可能最新的平台。   
  
在Solaris中有一组推荐当部署Java应用程序的补丁。 让这些Java Solaris补丁,请参见下一节的链接 [Solaris操作系统补丁](http://www.oracle.com/technetwork/java/javase/downloads/index-jdk5-jsp-142662.html#patches) 在Javaa¢下载页面。

**2.4消除差异**

请注意,不同的系统活动和操作系统上运行的其他应用程序可能会引入大量的方差为任何应用程序的性能的测量,包括Java应用程序。 操作系统和其他应用程序的活动可能会引入CPU、内存、磁盘或网络资源争用,可能会干扰你的测量。   
  
你开始测量之前Java性能评估应用程序的行为可能尝试 *变色* 你的性能结果(例如,访问互联网的更新,从用户的主目录,阅读文件等等)。 尽可能通过简化应用程序行为,通过改变一次只有一个变量(即操作系统可调参数,Java命令行选项,程序参数,等等),你的表现独立调查可以跟踪每个变化的影响。

|  |
| --- |
|  |

**3数据的决策**

真的很容易运行一个应用程序一次,一次改变后,得出一些结论,变化的影响。 有时,应用程序运行很长一段时间。 有时启动应用程序可能是复杂的,依赖于多个外部服务。 但你能合理地做出决定从这个测试? 真的不可能知道从数据可以得出一个结论,除非你测量数据定量的力量。 应用科学的方法是很重要的,在设计任何一组实验。 严谨性尤为必要在测量Java应用程序性能的行为因为Javaa¢HotSpota¢虚拟机适应和反应到特定的机器和特定的应用程序正在运行。 和微妙的变化在时间由于其他系统活动可以导致性能和可衡量的差异无关的比较。

**3.1微基准测试的注意!**

Java的优点之一是,它为数据在运行时动态地优化。 我们发现越来越多情况下,Java性能达到或超过了类似静态编译程序的性能。 然而这种适应性的JVM很难衡量Java功能的小片段。   
  
的原因之一,是具有挑战性的衡量Java性能是它会随着时间而改变。 在启动时,JVM通常会花一些时间“热身”。 根据JVM实现,可能会花一些时间在解释模式虽然异形找到“热”的方法。 当一个方法变得足够热时,它可能被编译和优化本机代码。   
  
的确有些优化展开循环,提升机变量之外的循环,甚至完全消除“死代码”。 那么你应该如何处理呢? 你应该定时的机器速度因素差异几个迭代循环,然后运行的迭代的数量会给你类似的总运行时间在每个平台? 如果你这样做你的时机是什么可能发生循环将估计循环时间在解释模式下运行。 当运行测试循环优化和运行更快。 如此之快,事实上,你的总运行时间可能太短,你没有测量内循环只是热身应用程序所需的基础设施。   
  
对于某些应用程序的垃圾收集可以使写作标准。 然而,重要的是要注意,对于一个给定的调优参数GC吞吐量是可预测的。 所以要么避免对象分配在你的内循环(避免激怒GC的)或GC运行足够长的时间来达到稳定状态。 如果你分配对象作为基准的一部分小心堆的大小减少GC的影响和收集到足够的样本,这样你得到公平的平均有多少时间是花在GC。   
  
甚至还有更微妙的陷阱与基准。 如果循环内的工作不是真正的常数为每个迭代? 如果你添加一个字符串,例如,你可能会做一份附加这将增加的工作量每次循环执行。 记得让循环常数的计算和非平凡的。 通过印刷最终答案你会阻止整个循环体被完全消除。   
  
显然运行稳定状态是至关重要的可重复的结果。 考虑运行基准数分钟。 任何应用程序运行不到一分钟可能是由JVM启动时间。 另一个时间的问题,可能发生抖动。 抖动时计时机制是多粗的粒度比基准时间的可变性。 在某些Windows平台上,例如,电话 [System.currentTimeMillis()](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/System.html#currentTimeMillis%28%29) 有一个有效的15毫秒的时间粒度。 简称测试这会变色的结果。 任何新的基准应该利用新的 [system . nanotime()](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/System.html#nanoTime%28%29) 方法应该减少这种抖动。

**3.2使用统计**

实验前,你应该尽量消除尽可能多的应用程序的性能变化。 然而很少有可能消除 *所有* 变化,特别是噪声从异步操作系统服务。 通过重复相同的实验的几个试验和平均的结果有效地集中在信号的噪声。 提高信号的速度的平方根成正比的样本数量(更多见 [减少噪音的影响平均的数据采集系统](http://zone.ni.com/devzone/conceptd.nsf/webmain/D3887A3DF70CBE0F86256A5400681ACD) )。   
  
把Java基准统计方面我们要测试 *基线* 在变化和设置 *标本* 更改后的设置。 例如您可以运行一个基准基线测试10次没有Java命令行选项的命令行和标本测试10次“- server”。 这将给你两个不同的样本数量。 你真的想要回答的问题是“改变Java设置产生影响吗? ,如果是,多大的区别吗?”。   
  
第二个问题,确定改进的比例实际上是简单的问题:   
percentageImprovement = 100.0 x(SpecimenAvg - BaselineAvg)/ BaselineAvg   
  
第一个问题,“改变Java设置产生影响吗? ”,是最重要的问题,但是因为我们真正想知道的是“足够重要的区别是,我们可以得出结论吗?”。 统计术语这个问题可以描述为“做这两个样本的数量反映了相同的底层人口吗?”。 要回答这个问题,我们使用学生们学习任务。 看到 [指针](http://www.oracle.com/technetwork/java/tuning-139912.html#section7) 节的更多背景学生的学习任务。 使用样本的数量,基线人口的平均值和标准偏差和标本人口以及设置所需的风险水平( *α* 我们可以确定 *假定值* 或概率变化在Java环境中是很有意义的。 风险级别( *α* )通常被设置为0.05(或0.01)这意味着五次(一次)的一百年,你将找到一个显著统计学差异意味着即使没有。 一般来说,如果 *假定值* 小于0.05然后我们会说,区别是重要的,因此我们可以得出结论,Java设置的变化产生影响。 更多解释假定值和功率分析请参阅 [假设检验的逻辑](http://davidmlane.com/hyperstat/logic_hypothesis.html) 。

**3.3使用基准利用**

什么是 *利用基准* 吗? 基准利用通常是使用一个脚本程序启动一个给定的基准,捕获输出日志文件,并提取基准分数(和小分)。 经常利用基准有设施预定数量的试验和运行基准测试,在理想的情况下,计算统计数据的结果不同的Java设置(设置变化是否在操作系统层面,Java调优或编码水平差异)。   
  
包括一些基准利用。 即使在这种情况下你想重写利用更多种类的测试参数,获取额外的数据和/或计算额外的数据。 甚至写一个简单的基准装置的优点是,它可以消除单调的收集许多样品,它可以启动应用程序以一致的方式,它能简化计算的过程数据。   
  
你是否使用基准利用它将必须确保当你尝试Java调优更改,你能够回答这个问题:“我收集到足够的样本提供足够的统计学意义,我可以得出结论从结果?”。

|  |
| --- |
|  |

**4一个调优的想法**

现在你已经采取了最简单的步骤 [最佳实践](http://www.oracle.com/technetwork/java/tuning-139912.html#section2) 节,准备调优通过理解的正确方法 [决策的数据](http://www.oracle.com/technetwork/java/tuning-139912.html#section3) 。 本节优化思想包含了各种调优选项建议,你应该在你的Java应用程序。 所有的对比不同的选项应该执行使用上面讨论的统计技术。

**4.1一般调优指南**

下面是一些通用调优指南帮助你分类的各种Java调优您将执行。

**以下4.4.1的了解人体工程学的设置**

之前,你开始优化Java命令行参数要知道太阳的HotSpota¢Java虚拟机整合技术开始调整自己。 这被称为智能调优 [人体工程学](http://www.oracle.com/technetwork/java/ergo5-140223.html) 。 大多数计算机,至少两个CPU和2 GB的物理内存 [*服务器级别的*](http://docs.oracle.com/javase/1.5.0/docs/guide/vm/server-class.html) 这意味着机器默认设置:

* 的 -服务器 编译器
* 的 - xx:+ UseParallelGC 平行(吞吐量)垃圾收集器
* 的 xms 初始堆大小1/64th机器的物理内存
* 的 - xmx 最大堆大小是1/4th机器的物理内存(最多1 GB max)。

请注意,所有使用32位Windows系统 客户端 编译器默认和64位Windows系统满足上述标准将被视为服务器级别的机器。

**4.1.2堆大小**

尽管人体工程学显著提高了“开箱即用”的经验对于许多应用程序,优化调优通常需要更多地关注Java内存区域的大小。   
  
最大堆大小的Java应用程序受到三个因素:过程数据模型(32位或64位)和相关的操作系统限制,虚拟内存系统上可用的数量,和系统上可用的物理内存数量。 一个特定的应用程序的Java堆的大小不能超过甚至达到最大过程的虚拟地址空间数据模型。 为一个32位的过程模型,过程的最大虚拟地址大小通常是4 GB,尽管有些操作系统限制这种2 GB或3 GB。 最大堆大小通常-Xmx3800m(1600)2 GB限制),但实际的限制是依赖于应用程序的。 对于64位的流程模型,最大是无限的。 专用系统为一个Java应用程序,Java堆的大小不应设置为系统上的物理RAM的数量,需要额外的内存操作系统,其他系统进程,甚至其他JVM操作。 承诺太多的系统的物理内存可能会导致虚拟内存分页到磁盘,在垃圾收集操作,很有可能导致重大的性能问题。 系统与多个Java进程,或多个流程,这些流程的Java堆的总和也应该不超过系统中物理RAM的大小。   
  
下一个最重要的Java内存可调的大小如果年轻一代(也称为NewSize)。 一般来说最大的推荐值的年轻一代是最大堆大小的3/8。 注意,吞吐量和低暂停时间收藏家可以超过这个比例。 有关更多信息,请参见讨论的年轻一代的保证 [微调垃圾收集与5.0 Java虚拟机](http://www.oracle.com/technetwork/java/gc-tuning-5-138395.html) 文档。   
  
额外的内存设置,如堆栈大小,将在下面更详细地介绍。

**4.1.3垃圾收集器政策**

的Javaa¢平台提供了一个选择的垃圾收集算法。 每种算法都有各种各样的政策可调参数。 而不是重复的细节 [微调垃圾收集](http://www.oracle.com/technetwork/java/gc-tuning-5-138395.html) 文档在这里我只想说,前两个选择是最常见的大型服务器应用程序:

* 的 - xx:+ UseParallelGC 平行(吞吐量)垃圾收集器,或
* 的 - xx:+ UseConcMarkSweepGC 并发垃圾收集器(低暂停时间)(也称为CMS)
* 的 - xx:+ UseSerialGC 串行垃圾收集器(对于较小的应用程序和系统)

**4.1.4其他调优参数**

某些其他Java调优参数对性能的影响将这里提到很高。 请参见 [指针](http://www.oracle.com/technetwork/java/tuning-139912.html#section7) 部分全面参考Java调优参数。   
  
的 [VM选项](http://www.oracle.com/technetwork/java/javase/tech/vmoptions-jsp-140102.html#largepages) 讨论了Java支持大内存页的页面。 通过适当配置操作系统,然后使用命令行选项 - xx:+ UseLargePages ( *在默认情况下为Solaris* ), - xx:LargePageSizeInBytes 可以得到最好的服务器的内存管理系统的效率。 页面更大的注意,我们可以更好地利用虚拟内存硬件资源(tlb),但这可能导致更大的空间大小为永久性的生成和缓存的代码,进而可以迫使你减少你的Java堆的大小。 这是一个很小的关注2 MB或4 MB页面大小,但256 MB页面大小的一个更有趣的问题。   
  
选择一个特定solaris可调的一个例子 **libumem** 替代堆分配器。 尝试 **libumem** 在Solaris中可以使用以下指令LD\_PRELOAD环境变量:

* 设置 **libumem** 所有子进程的一个给定的壳,设置和导出环境变量   
  LD\_PRELOAD = / usr / lib / libumem.so
* 启动Java应用程序 **libumem** 从 **上海** :   
  LD\_PRELOAD = / usr / lib / libumem。 因此,java *java设置application-args*
* 启动Java应用程序 **libumem** 从 **csh** :   
  env LD\_PRELOAD = / usr / lib / libumem。 因此,java *java设置application-args*

您可以验证 **libumem** 在使用通过验证过程设置 **pmap(1)** 或 **pldd(1)** 。

**4.2调优示例**

下面是一些具体的调优示例,用于实验。 请理解,这些只是例子,最优堆大小和调优参数为您的应用程序在您的硬件可能有所不同。

**4.2.1一个调优示例1:准备调优的吞吐量**

这里是一个例子具体命令行调优一个服务器应用程序运行在系统4 GB的内存和能够同时运行32线程和核心或上下文(CPU)。   
  
**java** -Xmx3800m -Xms3800m -Xmn2g - xss128k - xx:+ UseParallelGC - xx:ParallelGCThreads = 20   
  
评论:

* **-Xmx3800m -Xms3800m**   
  配置一个大型Java堆利用大内存系统。
* **-Xmn2g**   
  配置一个大型堆的年轻一代(这可以并行收集),再次利用大内存系统。 它有助于防止过早短暂的对象提升为旧的一代,垃圾收集是更加昂贵。
* **- xss128k**   
  减少默认最大线程堆栈大小,允许更多的进程的虚拟内存地址空间使用的Java堆。
* **- xx:+ UseParallelGC**   
  选择并行的新一代垃圾收集器Java堆(注:这是一般违约 [服务器级别的](http://docs.oracle.com/javase/1.5.0/docs/guide/vm/server-class.html) 机器)
* **- xx:ParallelGCThreads = 20**   
  减少垃圾回收线程的数量。 默认等于处理器数,这可能会不必要地高32线程能力系统。

**4.2.2一个调优示例2:并行年老代收集器**

类似于示例1我们这里想测试并行年老代收集器的影响。   
  
**java** -Xmx3550m -Xms3550m -Xmn2g - xss128k - xx:+ UseParallelGC - xx:ParallelGCThreads = 20 - xx:+ UseParallelOldGC   
  
评论:

* **-Xmx3550m -Xms3550m**   
  尺寸已经减少了。 ParallelOldGC收集器有额外的本地,非Java堆内存需求,所以Java堆大小可能需要减少运行32位JVM时。
* **- xx:+ UseParallelOldGC**   
  使用并行年老代收集器。 旧一代的某些阶段收集可以并行执行,加快旧一代集合。

**4.2.3调优示例3:尝试256 MB页**

这个调优示例是特定于这些要是系统支持巨大的页面大小256 MB。   
  
**java** -Xmx2506m -Xms2506m -Xmn1536m - xss128k - xx:+ UseParallelGC - xx:ParallelGCThreads = 20 - xx:+ UseParallelOldGC - xx:LargePageSizeInBytes = 256   
  
评论:

* **-Xmx2506m -Xms2506m**   
  尺寸已经减少了因为使用大页面设置导致永久性的生成和代码缓存大小为256 MB,这减少了Java堆内存可用。
* **-Xmn1536m**   
  年轻一代堆大小通常是作为整个Java堆大小的一小部分。 通常我们建议你开始调优与年轻一代的大小1/4th总堆大小。 年轻一代是在这种情况下保持类似的比率降低年轻一代和年老代上浆前面示例中使用的选择使用。
* **- xx:LargePageSizeInBytes = 256**   
  导致Java堆,包括永久的一代,和编译后的代码缓存页面作为一个最小大小256 MB(对于那些支持它的平台)。

**4.2.4调优示例4:尝试- xx:+ AggressiveOpts**

这调优示例例2类似,但添加了AggressiveOpts选项。   
  
**java** -Xmx3550m -Xms3550m -Xmn2g - xss128k - xx:+ UseParallelGC - xx:ParallelGCThreads = 20 - xx:+ UseParallelOldGC - xx:+ AggressiveOpts   
  
评论:

* **-Xmx3550m -Xms3550m**   
  尺寸增加回到示例2的水平,因为我们不再使用巨大的页面。
* **-Xmn2g**   
  尺寸增加回到示例2的水平,因为我们不再使用巨大的页面。
* **- xx:+ AggressiveOpts**   
  打开点性能优化,预计将在默认情况下在以后的版本中。 更改这个标志分组的微小的变化,JVM运行时编译后的代码,而不是不同的性能特性(如BiasedLocking和ParallelOldGC)。 这是一个很好的旗帜尝试JVM工程团队的最新即将发布的性能调整。 注意:此选项 *实验* ! 具体优化通过这个选项可以改变从版本甚至构建。 你应该重新评估的影响这个选项之前部署一个新版本的Java。

**4.2.5调优示例5:尝试偏向锁**

这个优化的例子是建立在例4,偏向锁选项。   
  
**java** -Xmx3550m -Xms3550m -Xmn2g - xss128k - xx:+ UseParallelGC - xx:ParallelGCThreads = 20 - xx:+ UseParallelOldGC - xx:+ AggressiveOpts - xx:+及更高   
  
评论:

* **- xx:+及**   
  使技术改善无争用同步的性能。 “偏见”向线程对象首先获得其监控通过 **monitorenter** 字节码或同步方法调用;该线程执行的后续监控操作相对更快的在多处理器的机器上。 一些应用程序可能会获得大量的无争用同步的加速启用了这个标志;一些应用程序的特定模式锁定可能会看到减速,尽管尝试了的负面影响降到最低。

**4.2.6调优示例6:优化低暂停时间和高吞吐量**

这调优示例与例2类似,但使用并发垃圾收集器(而不是并行吞吐量收集器)。   
  
**java** -Xmx3550m -Xms3550m -Xmn2g - xss128k - xx:ParallelGCThreads = 20 - xx:+ UseConcMarkSweepGC - xx:+ UseParNewGC - xx:SurvivorRatio = 8 - xx:TargetSurvivorRatio = 90 - xx:MaxTenuringThreshold = 31   
  
评论:

* **- xx:+ UseConcMarkSweepGC - xx:+ UseParNewGC**   
  选择马克扫描并发收集器。 这种收集器可能提供更好的应用程序的响应时间属性(即。 、低应用程序暂停时间)。 这是平行的,多并发收集器和可以很好的搭配线程一个大型多处理器系统的能力。
* **- xx:SurvivorRatio = 8**   
  幸存者空间比设置为1:8,导致更大的幸存者空间(比例越小,空间越大)。 大幸存者空间允许短暂的对象较长时间内死于年轻一代。
* **- xx:TargetSurvivorRatio = 90**   
  允许90%的幸存者空间被占据而不是默认的50%,允许更好的幸存者空间内存的利用率。
* **- xx:MaxTenuringThreshold = 31**   
  允许短暂的对象更长的时间死在年轻一代(,因此,避免促销)。 这个设置的结果是次要的GC时间可以增加额外对象复制。 这个值和幸存者空间大小可能需要调整以平衡之间的复制开销幸存者空间和任期的对象要住很长一段时间。 CMS的缺省设置 **SurvivorRatio = 1024** 和 **MaxTenuringThreshold = 0** 导致所有的幸存者被提升的清除。 这可以将大量的压力一个并发线程收集终身的一代。 注意:当使用 - xx:+及 ,这个设置应该是15。

**4.2.7调优示例7:尝试AggressiveOpts低暂停时间和高吞吐量**

这个调优示例是构建在示例6,并添加AggressiveOpts选项。   
  
**java** -Xmx3550m -Xms3550m -Xmn2g - xss128k - xx:ParallelGCThreads = 20 - xx:+ UseConcMarkSweepGC - xx:+ UseParNewGC - xx:SurvivorRatio = 8 - xx:TargetSurvivorRatio = 90 - xx:MaxTenuringThreshold = 31 - xx:+ AggressiveOpts   
  
评论:

* **- xx:+ AggressiveOpts**   
  打开点性能优化,预计将在默认情况下在以后的版本中。 更改这个标志分组的微小的变化,JVM运行时编译后的代码,而不是不同的性能特性(如BiasedLocking和ParallelOldGC)。 这是一个很好的旗帜尝试JVM工程团队的最新即将发布的性能调整。 注意:此选项 *实验* ! 具体优化通过这个选项可以改变从版本甚至构建。 你应该重新评估的影响这个选项之前部署一个新版本的Java。

|  |
| --- |
|  |

**5一个监控和分析**

讨论监控(从正在运行的应用程序)中提取高水平的统计数据或分析(插装应用程序提供详细的性能统计数据)是对象本身的白皮书。 这个Java为目的的调优白皮书这些主题将使用工具为例,介绍了可用于永久免费。

**5.1监测**

的Javaa¢平台提供了大量的内置监测设施。 请参阅文档 [监视和管理的Javaa¢平台](http://docs.oracle.com/javase/1.5.0/docs/guide/management/index.html) 为更多的信息。   
  
最受欢迎的“内置”工具 [JConsole](http://www.oracle.com/technetwork/java/tuning-139912.html) 和 [jvmstat](http://www.oracle.com/technetwork/java/jvmstat-142257.html) 技术。

**5.2分析**

的Javaa¢平台还包括一些分析设施。 最受欢迎的“内置”分析工具 [的 -Xprof 分析器](http://www.javaperformancetuning.com/news/qotm037.shtml) 和 [HPROF分析器](http://www.oracle.com/technetwork/java/tuning-139912.html) (使用HPROF见也 [堆分析工具](https://hat.dev.java.net/) )。   
  
一个分析器 [JFluid技术](http://profiler.netbeans.org/jfluid.html) 已纳入受欢迎 [NetBeans](http://profiler.netbeans.org/index.html) 开发工具。

|  |
| --- |
|  |

**6的编码性能**

本节将介绍编码水平变化,你可以将对性能产生影响。 为了这个初稿Java调优的白皮书的例子的各种编码水平变化,可能会对性能产生影响正在利用新的语言特性,如NIO和并发实用程序。   
  
的 [新的I / O API](http://docs.oracle.com/javase/1.5.0/docs/guide/nio/index.html) (NIO)为操作,比如内存映射文件提供改进的性能和可伸缩的网络操作。 通过使用NIO开发者可以显著改善内存或网络密集型应用程序的性能。 使用NIO的成功故事之一 [灰熊web容器](http://weblogs.java.net/blog/jfarcand/archive/2005/06/grizzly_an_http.html) 这是太阳的一部分吗 [GlassFish](https://glassfish.dev.java.net/) 项目。   
  
另一个例子的新Java语言的设置是影响性能的特性 [并发实用程序](http://docs.oracle.com/javase/1.5.0/docs/guide/concurrency/index.html) 。 越来越多的服务器应用程序将目标平台与多个CPU和多核CPU。 为了最好地利用这些系统必须在设计时就考虑多线程的应用程序。 古典多线程编程是非常复杂而且容易出错由于线程交互的微妙之处如竞态条件。 现在与并发实用程序开发者终于有一套坚实的基石,在该平台上可以构建可伸缩的多线程应用程序,同时避免编写多线程框架的复杂性。

|  |
| --- |
|  |

**7一个指针**

下面是本文中引用的资源。

* 获取和部署Java的最新版本
  + [下载最新版本的Java](http://www.oracle.com/technetwork/java/javase/downloads/index-jdk5-jsp-142662.html)
  + [Java J2SE 5.0的部署](http://docs.oracle.com/javase/1.5.0/docs/guide/deployment/)
  + [Java Web Start J2SE 5.0](http://docs.oracle.com/javase/1.5.0/docs/guide/javaws/)
  + [J2SE 5.0 Java插件](http://docs.oracle.com/javase/1.5.0/docs/guide/plugin/)
* J2SE 5.0平台
  + [J2SE 5.0采用](http://www.oracle.com/technetwork/java/tigeradoption-139250.html)
  + [理由迁移到J2SE 5.0(虎)](http://www.oracle.com/technetwork/articles/javase/migratetotiger-136405.html)
  + [J2SE 5.0支持的系统配置](http://www.oracle.com/technetwork/java/javase/system-configurations-139801.html)
  + [J2SE平台概述](http://www.oracle.com/technetwork/java/javase/tech/index-jsp-140763.html)
  + [J2SE 5.0概述](http://docs.oracle.com/javase/1.5.0/docs/guide/index.html)
  + [J2SE 5.0文档指南](http://docs.oracle.com/javase/1.5.0/docs/)
  + [Java命名约定](http://weblogs.java.net/blog/kgh/archive/2005/06/goodbye_j2se_he_1.html)
* 理解统计数据
  + [维基百科:学生的学习任务](http://en.wikipedia.org/wiki/Student%27s_t-test)
  + [统计每一个作家都应该知道](http://nilesonline.com/stats/) 包括 [学生的T](http://www.robertniles.com/stats/tscore.shtml)
  + [HyperStat在线:入门统计教科书](http://davidmlane.com/hyperstat/index.html)
* Java调优和人体工程学
  + [人体工程学在5.0 Java虚拟机](http://www.oracle.com/technetwork/java/ergo5-140223.html)
  + [更有益的检测](http://docs.oracle.com/javase/1.5.0/docs/guide/vm/server-class.html)
  + [垃圾收集器工效学](http://docs.oracle.com/javase/1.5.0/docs/guide/vm/gc-ergonomics.html)
  + [标准的Java选项](http://docs.oracle.com/javase/1.5.0/docs/tooldocs/solaris/java.html)
  + [Java HotSpot VM选项](http://www.oracle.com/technetwork/java/javase/tech/vmoptions-jsp-140102.html)
  + [乔嘲笑的JVM选项的集合](http://blogs.sun.com/roller/resources/watt/jvm-options-list.html)
* 垃圾收集
  + [微调垃圾收集与5.0 Java虚拟机](http://www.oracle.com/technetwork/java/gc-tuning-5-138395.html)
* 监控、管理和分析
  + [监视和管理的Javaa¢平台](http://docs.oracle.com/javase/1.5.0/docs/guide/management/index.html)
  + [使用JConsole监控应用程序](http://www.oracle.com/technetwork/java/tuning-139912.html)
  + [jvmstat 3.0](http://www.oracle.com/technetwork/java/jvmstat-142257.html) 和 [visualgc——视觉垃圾收集监控工具](http://www.oracle.com/technetwork/java/visualgc-136680.html)
  + [的 -Xprof 分析器](http://www.javaperformancetuning.com/news/qotm037.shtml)
  + 的 [HPROF分析器](http://www.oracle.com/technetwork/java/tuning-139912.html) 和 [堆分析工具](https://hat.dev.java.net/)
  + [的NetBeans分析器](http://profiler.netbeans.org/index.html)
* 合成负载生成
  + [SLAMD分布式负载生成引擎](http://www.slamd.com/)
* 编码性能
  + [J2SE 5.0新特性和增强功能](http://docs.oracle.com/javase/1.5.0/docs/relnotes/features.html)
  + [Java 5.0并发实用程序](http://docs.oracle.com/javase/1.5.0/docs/guide/concurrency/overview.html)
* Java性能文档
  + [Java性能技术主题](http://www.oracle.com/technetwork/java/javase/tech/performance-jsp-141338.html)
  + [J2SE 5.0性能白皮书](http://www.oracle.com/technetwork/java/5-136747.html)
  + [Java性能社区](https://performance.dev.java.net/nonav/home/)

|  |
| --- |
|  |

**8一个反馈和Java性能社区**

作为本白皮书是一个活生生的文档,我们希望通过添加其他最佳实践不断改进和优化思想以及额外的指针。

**8.1一个反馈**

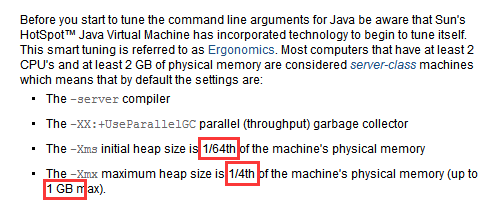
你有一个评论,问题,错误或增强这个白皮书? 请提交你的评论 [一般性能讨论](http://forums.java.net/jive/forum.jspa?forumID=60) 论坛 [java.net](https://performance.dev.java.net/nonav/home/) 并使用“Java调优白皮书”的主题。 讨论后我们已经开始初稿的Wiki页面java.net讨论修正为未来版本本白皮书和想法:PerformanceTuningWhitePaper

**8.2 Java性能社区**

你有性能问题不回答这份文件吗? 愿意贡献代码,帮助运行Java应用程序(利用)和/或计算统计数据? 请张贴您的问题或想法 [一般性能讨论](http://forums.java.net/jive/forum.jspa?forumID=60) 论坛。 论坛的地方 [Java性能社区](https://performance.dev.java.net/nonav/home/) 可以分享想法和最佳实践。

# 总结

## 1.堆内存





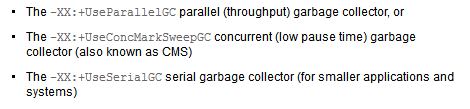
永远记住，jvm是运行在操作系统上的！

## 2.年轻代

Generally speaking the largest recommended value for the young generation is 3/8 of the maximum heap size(年轻代最大建议为堆的3/8=0.375)

## 3.gc收集器分类

串行-并行-并发



-sever-Xmx3800m -Xms3800m

-Xmn2g

-xss128k

-xx:+ UseParallelGC

-xx:ParallelGCThreads = 20

-XX:+UseParallelGC

-XX:+UseParallelOldGC

-XX:+UseParNewGC#当使用CMS收集器时，该标志激活年轻代使用多线程并行执行垃圾回收

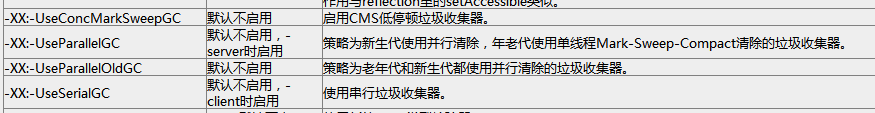
-XX:+UseConcMarkSweepGC

-XX:+UseG1GC

**-XX:SurvivorRatio=8**

**-XX:TargetSurvivorRatio=90 利用率，默认50%**

-XX:MaxTenuringThreshold=31 对象生命数，默认15



对于CMS收集器，年轻代**GC**算法和老年代**GC**算法是不同的，因此年轻代**GC**有两种不同的实现，并且是两个不同的标志。