链滴

Hotspot JVM 1.8 参数查阅手册

作者：xjlnjut730
原文链接：https://ld246.com/article/1573940849546
来源网站：链滴

这里整理下JVM相关的参数信息，本文档使用的是MAC OS平台的Hotspot JVM，版本为1.8.0_181。

JVM运行模式

Hotspot JVM有两种运行模式，分别是server和client。

它们的区别是server的初始堆空间会大一些，默认使用的是并行垃圾回收器，启动慢运行快。client M相对来讲会保守一些，初始堆空间会小一些，使用串行的垃圾回收器，它的目标是为了让JVM的启速度更快，但运行速度会比server模式慢些。

可以使用-server和-client选项来指定JVM的类型。

X选项

java -X可以输出所以的X的选项，下面列出一些常用的选项。

```
-Xmixed          混合模式执行 (默认)
-Xint            仅解释模式执行
-Xloggc:<file>   将 GC 状态记录在文件中 (带时间戳)
-Xms<size>       设置初始 Java 堆大小
-Xmx<size>       设置最大 Java 堆大小
-Xss<size>       设置 Java 线程堆栈大小
```

XX选项

查看所有XX选项的执行语句为：java -XX:+PrintFlagsInitial，对应的配置语法如下：

- 如果是布尔类型的选项，它的格式为-XX:+flag或者-XX:-flag,分别表示开启和关闭选项。
- 针对非布尔类型的选项，它的格式为-XX:flag=value

说明部分是从互联网上搜集到的相关的定义，我整理的差不多的时候才知道，源码里有你想要的一切：

http://hg.openjdk.java.net/jdk8/jdk8/hotspot/file/tip/src/share/vm/runtime/globals.hpp

```
<table style='font-size:10px;'><tr><td style='width:220px;'>参数名</td><td style='width:60
x;'>默认值</td><td>说明</td></tr>
<tr><td> AdaptiveSizeDecrementScaleFactor</td><td>4</td><td>Adaptive size scale dow
 factor for shrinking </td></tr>
<tr><td> AdaptiveSizeMajorGCDecayTimeScale</td><td>10</td><td>Time scale over whi
h major costs decay </td></tr>
<tr><td> AdaptiveSizePausePolicy</td><td>0</td><td>Policy for changing generation size
for pause goals </td></tr>
<tr><td> AdaptiveSizePolicyCollectionCostMargin</td><td>50</td><td>If collection costs
are within margin, reduce both by full delta </td></tr>
<tr><td> AdaptiveSizePolicyInitializingSteps</td><td>20</td><td>Number of steps where
heuristics is used before data is used </td></tr>
<tr><td> AdaptiveSizePolicyOutputInterval</td><td>0</td><td>Collecton interval for print
ng information, zero => never </td></tr>
<tr><td> AdaptiveSizePolicyWeight</td><td>10</td><td>Weight given to exponential res
zing, between 0 and 100 </td></tr>
<tr><td> AdaptiveSizeThroughPutPolicy</td><td>0</td><td>Policy for changeing generat
on size for throughput goals </td></tr>
<tr><td> AdaptiveTimeWeight</td><td>25</td><td>Weight given to time in adaptive pol
```

cy, between 0 and 100 </td></tr>
<tr><td> AdjustConcurrency</td><td>false</td><td>call thr_setconcurrency at thread crea e time to avoid LWP starvation on MP systems (For Solaris Only) </td></tr>
<tr><td> AggressiveOpts</td><td>false</td><td>允许使用积极的性能优化功能，这些功能望在即将发布的版本中成为默认功能。默认情况下，禁用此选项并且不使用实验性能功能。 </td></t >
<tr><td> AliasLevel</td><td>3</td><td>0 for no aliasing, 1 for oop/field/static/array split, 2 for best </td></tr>
<tr><td> AlignVector</td><td>true</td><td>Perform vector store/load alignment in loop </td></tr>
<tr><td> AllocateInstancePrefetchLines</td><td>1</td><td>设置在实例分配指针之前预取行数。 </td></tr>
<tr><td> AllocatePrefetchDistance</td><td>-1</td><td>设置对象分配的预取距离的大小（字节为单位）。将从最后分配的对象的地址开始预取将要使用新对象的值写入的内存。每个Java线程有自己的分配点。负值表示基于平台选择预取距离。正值是预取的字节数。附加字母k或K表示千字节m或M指示兆字节，g或G指示千兆字节。 </td></tr>
<tr><td> AllocatePrefetchInstr</td><td>0</td><td>将预取指令设置为在分配指针之前预取。 </td></tr>
<tr><td> AllocatePrefetchLines</td><td>3</td><td>使用编译代码中生成的预取指令设置在后一次对象分配后要加载的高速缓存行数。如果最后分配的对象是实例，则默认值为1;如果是数组，默认值为3。 </td></tr>
<tr><td> AllocatePrefetchStepSize</td><td>16</td><td>设置顺序预取指令的步长（以字节单位）。附加字母k或K表示千字节，m或M指示兆字节，g或G指示千兆字节。 </td></tr>
<tr><td> AllocatePrefetchStyle</td><td>1</td><td>为预取指令设置生成的代码样式。0：不生成预取指令。1：每次分配后执行预取指令。2：使用线程局部分配块（TLAB）水印指针来确定何执行预取指令。3：在SPARC上使用BIS指令进行分配预取。 </td></tr>
<tr><td> AllowJNIEnvProxy</td><td>false</td><td>Allow JNIEnv proxies for jdbx </td></ r>
<tr><td> AllowNonVirtualCalls</td><td>false</td><td>Obey the ACC_SUPER flag and all w invokenonvirtual calls </td></tr>
<tr><td> AllowParallelDefineClass</td><td>false</td><td>Allow parallel defineClass requ sts for class loaders registering as parallel capable </td></tr>
<tr><td> AllowUserSignalHandlers</td><td>false</td><td>允许为java进程安装信号处理器。 </td></tr>
<tr><td> AlwaysActAsServerClassMachine</td><td>false</td><td>Always act like a server class machine </td></tr>
<tr><td> AlwaysCompileLoopMethods</td><td>false</td><td>when using recompilation, never interpret methods containing loops </td></tr>
<tr><td> AlwaysLockClassLoader</td><td>false</td><td>Require the VM to acquire the cl ss loader lock before calling loadClass() even for class loaders registering as parallel capable /td></tr>
<tr><td> AlwaysPreTouch</td><td>false</td><td>在JVM初始化期间允许触摸Java堆上的每页面。这会在进入main()方法之前将所有页面放入内存中。该选项可用于测试以模拟长时间运行的系，其中所有虚拟内存都映射到物理内存。 </td></tr>
<tr><td> AlwaysRestoreFPU</td><td>false</td><td>Restore the FPU control word after e ery JNI call (expensive) </td></tr>
<tr><td> AlwaysTenure</td><td>false</td><td>Always tenure objects in eden. (ParallelGC only) </td></tr>
<tr><td> AssertOnSuspendWaitFailure</td><td>false</td><td>Assert/Guarantee on exter al suspend wait failure </td></tr>
<tr><td> AssumeMP</td><td>false</td><td> </td></tr>
<tr><td> AutoBoxCacheMax</td><td>128</td><td>自动装箱拆箱缓存的最大值 </td></tr>
<tr><td> AutoGCSelectPauseMillis</td><td>5000</td><td>Automatic GC selection pause hreshhold in ms </td></tr>

<tr><td> BCEATraceLevel</td><td>0</td><td>How much tracing to do of bytecode escap
analysis estimates </td></tr>
<tr><td> BackEdgeThreshold</td><td>100000</td><td>统计一个方法中循环体代码执行的
数。 </td></tr>
<tr><td> BackgroundCompilation</td><td>true</td><td>启用后台编译。 </td></tr>
<tr><td> BaseFootPrintEstimate</td><td>2^28</td><td>Estimate of footprint other than
ava Heap </td></tr>
<tr><td> BiasedLockingBulkRebiasThreshold</td><td>20</td><td>偏向锁总撤销次数阈值
某一类锁的对象的总撤销次数超过该值，JVM会宣布该类的偏向锁失效。 </td></tr>
<tr><td> BiasedLockingBulkRevokeThreshold</td><td>40</td><td>偏向锁总撤销次数阈值
如果总撤销数超过该值，JVM会撤销该类实例的偏向锁，并且在之后的加锁过程中直接为该类实例设
轻量级锁。 </td></tr>
<tr><td> BiasedLockingDecayTime</td><td>25000</td><td>开启一次新的批量重偏向距离
次批量重偏向的后的延迟时间。就是开启批量重偏向后，经过了一段较长的时间（>=BiasedLocking
ecayTime），撤销计数器才超过阈值，那我们会重置计数器。 </td></tr>
<tr><td> BiasedLockingStartupDelay</td><td>4000</td><td>JVM启用后启用偏向锁的延迟
间。 </td></tr>
<tr><td> BindGCTaskThreadsToCPUs</td><td>false</td><td>绑定GC线程到个别的CPU核
仅在Solaris支持。 </td></tr>
<tr><td> BlockLayoutByFrequency</td><td>true</td><td>从热路径移动不频繁的执行分支。
</td></tr>
<tr><td> BlockLayoutMinDiamondPercentage</td><td>20</td><td>Miniumum %% of a s
ccessor (predecessor) for which block layout a will allow a fork (join) in a single chain </td></
r>
<tr><td> BlockLayoutRotateLoops</td><td>true</td><td>Allow back branches to be fall t
roughs in the block layour </td></tr>
<tr><td> BranchOnRegister</td><td>false</td><td>Use Sparc V9 branch-on-register opc
des </td></tr>
<tr><td> BytecodeVerificationLocal</td><td>false</td><td>Enables the Java bytecode veri
ier for local classes </td></tr>
<tr><td> BytecodeVerificationRemote</td><td>true</td><td>Enables the Java bytecode v
rifier for remote classes </td></tr>
<tr><td> C1OptimizeVirtualCallProfiling</td><td>true</td><td>Use CHA and exact type r
sults at call sites when updating MDOs </td></tr>
<tr><td> C1ProfileBranches</td><td>true</td><td>Profile branches when generating cod
for updating MDOs </td></tr>
<tr><td> C1ProfileCalls</td><td>true</td><td>Profile calls when generating code for upd
ting MDOs </td></tr>
<tr><td> C1ProfileCheckcasts</td><td>true</td><td>rofile checkcasts when generating c
de for updating MDOs </td></tr>
<tr><td> C1ProfileInlinedCalls</td><td>true</td><td>Profile inlined calls when generating
code for updating MDOs </td></tr>
<tr><td> C1ProfileVirtualCalls</td><td>true</td><td>Profile virtual calls when generating
ode for updating MDOs </td></tr>
<tr><td> C1UpdateMethodData</td><td>true</td><td>Update methodDataOops in Tier1
generated code </td></tr>
<tr><td> CICompilerCount</td><td>2</td><td>设置用于编译的编译器线程数。 </td></tr>
<tr><td> CICompilerCountPerCPU</td><td>false</td><td>单个CPU的编译器线程数。 </td
</tr>
<tr><td> CITime</td><td>false</td><td>打印消耗在JIT编译的时间 </td></tr>
<tr><td> CMSAbortSemantics</td><td>false</td><td>Whether abort-on-overflow semant
cs is implemented </td></tr>
<tr><td> CMSAbortablePrecleanMinWorkPerIteration</td><td>100</td><td>(Temporary,
ubject to experimentation)" Nominal minimum work per abortable preclean iteration </td></

r>
<tr><td> CMSAbortablePrecleanWaitMillis</td><td>100</td><td>(Temporary, subject to xperimentation)" Time that we sleep between iterations when not given" enough work per ite ation </td></tr>
<tr><td> CMSBitMapYieldQuantum</td><td>10485760</td><td>Bitmap operations shou d process at most this many bits between yields </td></tr>
<tr><td> CMSBootstrapOccupancy</td><td>50</td><td>老年代CMS GC时空间占比阈值。果预测CMS GC完成所需要的时间大于预计的老年代将要填满的时间，则进行GC。这些判断是需要基历史的CMS GC统计指标，然而，第一次CMS GC时，统计数据还没有形成，这时会跟据老年代的使占比来判断是否要进行GC。 </td></tr>
<tr><td> CMSClassUnloadingEnabled</td><td>true</td><td>使用并发标记清除（CMS）圾收集器时启用类卸载。 </td></tr>
<tr><td> CMSClassUnloadingMaxInterval</td><td>0</td><td>卸载类判定阈值，CMSClass nloadingEnabled启用时，若上次卸载类之后发生gc的次数大于该值，则进行卸载类。 </td></tr>
<tr><td> CMSCleanOnEnter</td><td>true</td><td>Clean-on-enter optimization for reduc ng number of dirty cards </td></tr>
<tr><td> CMSCompactWhenClearAllSoftRefs</td><td>true</td><td>回收软引用时进行CM压缩。 </td></tr>
<tr><td> CMSConcMarkMultiple</td><td>32</td><td>Size (in cards) of CMS concurrent T marking task </td></tr>
<tr><td> CMSConcurrentMTEnabled</td><td>true</td><td>当该标志被启用时，并发的CM阶段将以多线程执行(因此，多个GC线程会与所有的应用程序线程并行工作)。 </td></tr>
<tr><td> CMSCoordinatorYieldSleepCount</td><td>10</td><td>number of times the coo dinator GC thread will sleep while yielding before giving up and resuming GC </td></tr>
<tr><td> CMSDumpAtPromotionFailure</td><td>false</td><td>在晋升失败时dump信息。 </td></tr>
<tr><td> CMSEdenChunksRecordAlways</td><td>true</td><td>preclean阶段不对eden进抽样，而是在ParNew运行时抽样。 </td></tr>
<tr><td> CMSExpAvgFactor</td><td>50</td><td>设置在计算并发收集统计信息的指数平均时用于加权当前样本的时间百分比（0到100）。 </td></tr>
<tr><td> CMSExtrapolateSweep</td><td>false</td><td>CMS: cushion for block demand uring sweep </td></tr>
<tr><td> CMSFullGCsBeforeCompaction</td><td>0</td><td>设置执行多少次FullGC后才进一次内存碎片整理。默认每次都进行内存碎片整理。 </td></tr>
<tr><td> CMSIncrementalDutyCycle</td><td>10</td><td>每次增量回收垃圾的占总垃圾回任务的比例。此选项在JDK 8中已弃用。 </td></tr>
<tr><td> CMSIncrementalDutyCycleMin</td><td>0</td><td>每次增量回收垃圾的占总垃圾收任务的最小比例。此选项在JDK 8中已弃用。 </td></tr>
<tr><td> CMSIncrementalMode</td><td>false</td><td>开始CMS增量模式(i-cms)。在增量式下，CMS收集器在并发阶段，不会独占整个周期，而会周期性的暂停，唤醒应用线程。收集器把并阶段工作，划分为片段，安排在次级(minor)回收之间运行。这对需要低延时，运行在少量CPU服务上的应用很有用。此选项在JDK 8中已弃用。 </td></tr>
<tr><td> CMSIncrementalOffset</td><td>0</td><td>设置增量模式占空比在次要集合之间的间段内向右移动的时间百分比（0到100）。在弃用选项后，此选项在JDK 8中已弃用。 </td></tr>
<tr><td> CMSIncrementalPacing</td><td>true</td><td>根据应用程序的行为自动调整每次行的垃圾回收任务的数量，此选项在JDK 8中已弃用。 </td></tr>
<tr><td> CMSIncrementalSafetyFactor</td><td>10</td><td>设置计算占空比时用于添加保性的时间百分比（0到100）。在弃用选项后，此选项在JDK 8中已弃用。 </td></tr>
<tr><td> CMSIndexedFreeListReplenish</td><td>4</td><td>Replenish and indexed free li t with this number of chunks </td></tr>
<tr><td> CMSInitiatingOccupancyFraction</td><td>-1</td><td>指定在老年代用掉多少内后开始进行垃圾回收。与吞吐量优先的回收器不同的是，吞吐量优先的回收器在老生代内存用尽了以才开始进行收集，这对CMS来讲是不行的，因为吞吐量优先的垃圾回收器运行的时候会停止所有用户程，所以不会产生新的对象，而CMS运行的时候，用户线程还有可能产生新的对象，所以不能等到内

用光后才开始运行。比如-XX:CMSInitiatingOccupancyFraction=75表示老生代用掉75%后开始回垃圾。 </td></tr>
<tr><td> CMSIsTooFullPercentage</td><td>98</td><td>卸载类判定阈值，开启CMSClassUnoadingEnabled时，若老年代内存使用率高于该值，则进行类卸载。 </td></tr>
<tr><td> CMSLargeCoalSurplusPercent</td><td>0.95</td><td>CMS: the factor by which t inflate estimated demand of large block sizes to prevent coalescing with an adjoining block /td></tr>
<tr><td> CMSLargeSplitSurplusPercent</td><td>1.00</td><td>CMS: the factor by which t inflate estimated demand of large block sizes to prevent splitting to supply demand for smaler blocks </td></tr>
<tr><td> CMSLoopWarn</td><td>false</td><td>Warn in case of excessive CMS looping <td></tr>
<tr><td> CMSMaxAbortablePrecleanLoops</td><td>0</td><td>预清理中最多循环的次数循环超过该值，则退出循环。若为0，则无循环次数的限制。 </td></tr>
<tr><td> CMSMaxAbortablePrecleanTime</td><td>5000</td><td>预清理阶段最长时间，果达到该值，中止预清理阶段，进入remark。单位为毫秒。 </td></tr>
<tr><td> CMSOldPLABMax</td><td>1024</td><td>Max size of CMS gen promotion lab cches per worker per blksize </td></tr>
<tr><td> CMSOldPLABMin</td><td>16</td><td>Min size of CMS gen promotion lab cachs per worker per blksize </td></tr>
<tr><td> CMSOldPLABNumRefills</td><td>4</td><td>Nominal number of refills of CMSen promotion lab cache per worker per block size </td></tr>
<tr><td> CMSOldPLABReactivityFactor</td><td>2</td><td>The gain in the feedback loop or on-the-fly PLAB resizing during a scavenge </td></tr>
<tr><td> CMSOldPLABResizeQuicker</td><td>false</td><td>Whether to react on-the-fly uring a scavenge to a sudden change in block demand rate </td></tr>
<tr><td> CMSOldPLABToleranceFactor</td><td>4</td><td>The tolerance of the phase-chnge detector for on-the-fly PLAB resizing during a scavenge </td></tr>
<tr><td> CMSPLABRecordAlways</td><td>true</td><td>Whether to always record survivr space PLAB bdries" (effective only if CMSParallelSurvivorRemarkEnabled) </td></tr>
<tr><td> CMSParPromoteBlocksToClaim</td><td>16</td><td>Number of blocks to attept to claim when refilling CMS LAB for parallel GC. </td></tr>
<tr><td> CMSParallelInitialMarkEnabled</td><td>true</td><td>初始化标记阶段启用并行理。 </td></tr>
<tr><td> CMSParallelRemarkEnabled</td><td>true</td><td>标记标记阶段启用并行处理。/td></tr>
<tr><td> CMSParallelSurvivorRemarkEnabled</td><td>true</td><td>Whether parallel reark of survivor space" enabled (effective only if CMSParallelRemarkEnabled) </td></tr>
<tr><td> CMSPrecleanDenominator</td><td>3</td><td>CMSPrecleanNumerator:CMSPreleanDenominator yields convergence ratio </td></tr>
<tr><td> CMSPrecleanIter</td><td>3</td><td>Maximum number of precleaning iteration passes </td></tr>
<tr><td> CMSPrecleanNumerator</td><td>2</td><td>CMSPrecleanNumerator:CMSPrclanDenominator yields convergence ratio </td></tr>
<tr><td> CMSPrecleanRefLists1</td><td>true</td><td>Preclean ref lists during (initial) prclean phase </td></tr>
<tr><td> CMSPrecleanRefLists2</td><td>false</td><td>Preclean ref lists during abortable preclean phase </td></tr>
<tr><td> CMSPrecleanSurvivors1</td><td>false</td><td>Preclean survivors during (initial) preclean phase </td></tr>
<tr><td> CMSPrecleanSurvivors2</td><td>true</td><td>Preclean survivors during abortale preclean phase </td></tr>
<tr><td> CMSPrecleanThreshold</td><td>1000</td><td>Don't re-iterate if #dirty cards les than this </td></tr>

<tr><td> CMSPrecleaningEnabled</td><td>true</td><td>CMS是否需要进行Pre cleaning个阶段。 </td></tr>
<tr><td> CMSPrintChunksInDump</td><td>false</td><td>In a dump enabled by CMSDupAtPromotionFailure, include more detailed information about the free chunks. </td></tr>
<tr><td> CMSPrintEdenSurvivorChunks</td><td>false</td><td> </td></tr>
<tr><td> CMSPrintObjectsInDump</td><td>false</td><td>In a dump enabled by CMSDupAtPromotionFailure, include more detailed information about the allocated objects. </td></r>
<tr><td> CMSRemarkVerifyVariant</td><td>1</td><td>Choose variant (1,2) of verification following remark </td></tr>
<tr><td> CMSReplenishIntermediate</td><td>true</td><td>Replenish all intermediate fre-list caches </td></tr>
<tr><td> CMSRescanMultiple</td><td>32</td><td>Size (in cards) of CMS parallel rescan tsk </td></tr>
<tr><td> CMSSamplingGrain</td><td>16384</td><td>The minimum distance between een samples for CMS (see above) </td></tr>
<tr><td> CMSScavengeBeforeRemark</td><td>false</td><td>CMS在remark前强制进行一MinorGC </td></tr>
<tr><td> CMSScheduleRemarkEdenPenetration</td><td>50</td><td>CMS预清理后，eden space的使用率如果低于该值，则中断预清理，进入remark阶段。 </td></tr>
<tr><td> CMSScheduleRemarkEdenSizeThreshold</td><td>2097152</td><td>设置CMS新标记的Eden区大小阈值。低于此值时不启动重新标记阶段，因为回收预期微不足道。 </td></tr>
<tr><td> CMSScheduleRemarkSamplingRatio</td><td>5</td><td>Start sampling Eden to at least before yg occupancy reaches" 1/ of the size at which we plan to schedule remark </t></tr>
<tr><td> CMSSmallCoalSurplusPercent</td><td>1.05</td><td>CMS: the factor by which t inflate estimated demand of small block sizes to prevent coalescing with an adjoining block /td></tr>
<tr><td> CMSSmallSplitSurplusPercent</td><td>1.10</td><td>CMS: the factor by which t inflate estimated demand of large block sizes to prevent coalescing with an adjoining block /td></tr>
<tr><td> CMSSplitIndexedFreeListBlocks</td><td>true</td><td>When satisfying batched demand, split blocks from the IndexedFreeList whose size is a multiple of requested size </td </tr>
<tr><td> CMSTriggerInterval</td><td>-1</td><td>CMS回收的间隔。如果=0表示一直触发。 </td></tr>
<tr><td> CMSTriggerRatio</td><td>80</td><td>设置在-XX:MinHeapFreeRatioCMS收集周开始之前分配的值所指定的值的百分比（0到100）。 </td></tr>
<tr><td> CMSWaitDuration</td><td>2000</td><td>轮询判断老年代是否需要进行GC，每次循阻塞线程的时间。 </td></tr>
<tr><td> CMSWorkQueueDrainThreshold</td><td>10</td><td>Don't drain below this size per parallel worker/thief </td></tr>
<tr><td> CMSYield</td><td>true</td><td>Yield between steps of concurrent mark & swep </td></tr>
<tr><td> CMSYieldSleepCount</td><td>0</td><td>number of times a GC thread (minus te coordinator) will sleep while yielding before giving up and resuming GC </td></tr>
<tr><td> CMSYoungGenPerWorker</td><td>67108864</td><td>每个worker对应的年轻代小？跟硬件相关，x86机器为64M。 </td></tr>
<tr><td> CMS_FLSPadding</td><td>1</td><td>The multiple of deviation from mean to ue for buffering" against volatility in free list demand. </td></tr>
<tr><td> CMS_FLSWeight</td><td>75</td><td>Percentage (0-100) used to weight the curent sample when" computing exponentially decating averages for CMS FLS statistics. </td></r>
<tr><td> CMS_SweepPadding</td><td>1</td><td>The multiple of deviation from mean to

use for buffering" against volatility in inter-sweep duration. </td></tr>
<tr><td> CMS_SweepTimerThresholdMillis</td><td>10</td><td>Skip block flux-rate samp
ing for an epoch unless inter-sweep duration exceeds this threhold in milliseconds </td></tr

<tr><td> CMS_SweepWeight</td><td>75</td><td>Percentage (0-100) used to weight the
current sample when" computing exponentially decaying average for inter-sweep duration. <
td></tr>
<tr><td> CheckEndorsedAndExtDirs</td><td>false</td><td> </td></tr>
<tr><td> CheckJNICalls</td><td>false</td><td>Verify all arguments to JNI calls </td></t
>
<tr><td> ClassUnloading</td><td>true</td><td>启用对类进行回收。 </td></tr>
<tr><td> ClassUnloadingWithConcurrentMark</td><td>true</td><td>在并发标记阶段结束
，启用该参数则进行类型卸载。 </td></tr>
<tr><td> ClearFPUAtPark</td><td>0</td><td>(Unsafe,Unstable) </td></tr>
<tr><td> ClipInlining</td><td>true</td><td>clip inlining if aggregate method exceeds Des
redMethodLimit </td></tr>
<tr><td> CodeCacheExpansionSize</td><td>65536</td><td>用于设置code cache的expans
on size </td></tr>
<tr><td> CodeCacheMinimumFreeSpace</td><td>512000</td><td>设置编译所需的最小可
空间（以字节为单位）。 </td></tr>
<tr><td> CollectGen0First</td><td>false</td><td>FullGC时是否先YGC。 </td></tr>
<tr><td> CompactFields</td><td>true</td><td>是否在header与下一个8-byte对齐位置之间
空隙处存放成员。 </td></tr>
<tr><td> CompilationPolicyChoice</td><td>0</td><td>which compilation policy </td></
r>
<tr><td> CompileCommand</td><td></td><td> 指定要对方法执行的命令。 </td></tr>
<tr><td> CompileCommandFile</td><td></td><td> 设置从中读取JIT编译器命令的文件。 </
d></tr>
<tr><td> CompileOnly</td><td></td><td> List of methods (pkg/class.name) to restrict c
mpilation to </td></tr>
<tr><td> CompileThreshold</td><td>10000</td><td>方法触发编译时的调用次数。 </td></
r>
<tr><td> CompilerThreadHintNoPreempt</td><td>true</td><td>设置应限制编译的方法列
（以逗号分隔）。仅编译指定的方法。使用完整的类名（包括包和子包）指定每个方法。 </td></tr>
<tr><td> CompilerThreadPriority</td><td>-1</td><td>编译器线程的优先级(- 1表示不变) </
d></tr>
<tr><td> CompilerThreadStackSize</td><td>0</td><td>编译器线程的栈大小 </td></tr>
<tr><td> CompressedClassSpaceSize</td><td>2^10</td><td>类指针压缩空间大小 </td></
r>
<tr><td> ConcGCThreads</td><td>0</td><td>CMS并发阶段的线程数。默认值取决于JVM可
的CPU数。 </td></tr>
<tr><td> ConditionalMoveLimit</td><td>3</td><td>Limit of ops to make speculative wh
n using CMOVE </td></tr>
<tr><td> ContendedPaddingWidth</td><td>128</td><td>缓存行填充宽度。 </td></tr>
<tr><td> ConvertSleepToYield</td><td>true</td><td>Converts sleep(0) to thread yield (
ay be off for SOLARIS to improve GUI) </td></tr>
<tr><td> ConvertYieldToSleep</td><td>false</td><td>Converts yield to a sleep of MinSle
pInterval to simulate Win32 behavior (SOLARIS only) </td></tr>
<tr><td> CrashOnOutOfMemoryError</td><td>false</td><td>If this option is enabled, wh
n an out-of-memory error occurs, the JVM crashes and produces text and binary crash files. <
td></tr>
<tr><td> CreateMinidumpOnCrash</td><td>false</td><td>Create minidump on VM fatal
rror </td></tr>
<tr><td> CriticalJNINatives</td><td>true</td><td>check for critical JNI entry points </td

</tr>
<tr><td> DTraceAllocProbes</td><td>false</td><td>Enable dtrace probes for object alloc
tion </td></tr>
<tr><td> DTraceMethodProbes</td><td>false</td><td>Enable dtrace probes for method-
ntry and method-exit </td></tr>
<tr><td> DTraceMonitorProbes</td><td>false</td><td>Enable dtrace probes for monitor
vents </td></tr>
<tr><td> Debugging</td><td>false</td><td>set when executing debug methods in debu
.ccp (to prevent triggering assertions) </td></tr>
<tr><td> DefaultMaxRAMFraction</td><td>4</td><td>Fraction (1/n) of real memory used
for server class max heap </td></tr>
<tr><td> DefaultThreadPriority</td><td>-1</td><td>what native priority threads run at if
ot specified elsewhere (-1 means no change) </td></tr>
<tr><td> DeferPollingPageLoopCount</td><td>-1</td><td>(Unsafe,Unstable) Number of i
erations in safepoint loop before changing safepoint polling page to RO </td></tr>
<tr><td> DeferThrSuspendLoopCount</td><td>4000</td><td>(Unstable) Number of time
 to iterate in safepoint loop before blocking VM threads </td></tr>
<tr><td> DeoptimizeRandom</td><td>false</td><td>deoptimize random frames on rand
m exit from the runtime system </td></tr>
<tr><td> DisableAttachMechanism</td><td>false</td><td>禁止工具连接到JVM。禁用后将
能使用jcmd、jstack、jmap和jinfo等命令。 </td></tr>
<tr><td> DisableExplicitGC</td><td>false</td><td>禁止显式调用GC。 </td></tr>
<tr><td> DisplayVMOutputToStderr</td><td>false</td><td>If DisplayVMOutput is true, d
splay all VM output to stderr </td></tr>
<tr><td> DisplayVMOutputToStdout</td><td>false</td><td>If DisplayVMOutput is true, d
splay all VM output to stdout </td></tr>
<tr><td> DoEscapeAnalysis</td><td>true</td><td>开启逃逸分析 </td></tr>
<tr><td> DontCompileHugeMethods</td><td>true</td><td>关闭大型方法的JIT编译。 </t
></tr>
<tr><td> DontYieldALot</td><td>false</td><td>Throw away obvious excess yield calls (for
SOLARIS only) </td></tr>
<tr><td> DumpLoadedClassList</td><td></td><td> dump当前JVM装载的类列表 </td></t
>
<tr><td> DumpReplayDataOnError</td><td>true</td><td> </td></tr>
<tr><td> DumpSharedSpaces</td><td>false</td><td>Special mode: JVM reads a class list,
loads classes, builds shared spaces, and dumps the shared spaces to a file to be used in future
JVM runs. </td></tr>
<tr><td> EagerXrunInit</td><td>false</td><td>Eagerly initialize -Xrun libraries; allows star
up profiling, but not all -Xrun libraries may support the state of the VM at this time </td></t
>
<tr><td> EliminateAllocationArraySizeLimit</td><td>64</td><td>大于该数量的元素的数组
会进行逃逸分析优化。 </td></tr>
<tr><td> EliminateAllocations</td><td>true</td><td>开启标量替换。即允许将对象分配在
。 </td></tr>
<tr><td> EliminateAutoBox</td><td>true</td><td>Private flag to control optimizations fo
 autobox elimination </td></tr>
<tr><td> EliminateLocks</td><td>true</td><td>开启锁消除优化。 </td></tr>
<tr><td> EliminateNestedLocks</td><td>true</td><td>开启嵌套锁优化。 </td></tr>
<tr><td> EmitSync</td><td>0</td><td>(Unsafe,Unstable) Controls emission of inline sync
ast-path code </td></tr>
<tr><td> EnableContended</td><td>true</td><td> </td></tr>
<tr><td> EnableResourceManagementTLABCache</td><td>true</td><td> </td></tr>
<tr><td> EnableSharedLookupCache</td><td>true</td><td> </td></tr>
<tr><td> EnableTracing</td><td>false</td><td>Enable event-based tracing </td></tr>

<tr><td> ErgoHeapSizeLimit</td><td>0</td><td>堆最大容量的默认值，如果设置的限制值MaxRAM/MaxRAMFraction还小，就使用该参数指定的值。 </td></tr>
<tr><td> ErrorFile</td><td></td><td> If an error occurs, save the error data to this file [de ault: ./hs_err_pid%p.log] (%p replaced with pid) </td></tr>
<tr><td> ErrorReportServer</td><td></td><td> Override built-in error report server addre s </td></tr>
<tr><td> EscapeAnalysisTimeout</td><td>20.00</td><td> </td></tr>
<tr><td> EstimateArgEscape</td><td>true</td><td>Analyze bytecodes to estimate escape state of arguments </td></tr>
<tr><td> ExitOnOutOfMemoryError</td><td>false</td><td>When you enable this option, he JVM exits on the first occurrence of an out-of-memory error. It can be used if you prefer re tarting an instance of the JVM rather than handling out of memory errors. </td></tr>
<tr><td> ExplicitGCInvokesConcurrent</td><td>false</td><td>如果在代码里显式调用Syste .gc()会做并行CMS GC。该参数在允许system GC且使用CMS GC有效 </td></tr>
<tr><td> ExplicitGCInvokesConcurrentAndUnloadsClasses</td><td>false</td><td>与上面似,额外增加对永久代的gc. </td></tr>
<tr><td> ExtendedDTraceProbes</td><td>false</td><td>启用dtrace影响性能的其他工具探。 </td></tr>
<tr><td> ExtraSharedClassListFile</td><td></td><td> </td></tr>
<tr><td> FLSAlwaysCoalesceLarge</td><td>false</td><td>CMS: Larger free blocks are al ays available for coalescing </td></tr>
<tr><td> FLSCoalescePolicy</td><td>2</td><td>CMS: Aggression level for coalescing, inc easing from 0 to 4 </td></tr>
<tr><td> FLSLargestBlockCoalesceProximity</td><td>0.99</td><td>CMS: the smaller the ercentage the greater the coalition force </td></tr>
<tr><td> FailOverToOldVerifier</td><td>true</td><td>如果新的Class校验器检查失败，则用老的校验器。 </td></tr>
<tr><td> FastTLABRefill</td><td>true</td><td>Use fast TLAB refill code </td></tr>
<tr><td> FenceInstruction</td><td>0</td><td>(Unsafe,Unstable) Experimental </td></tr

<tr><td> FieldsAllocationStyle</td><td>1</td><td>字段排列方式。类型0, 引用在原始类型面, 然后依次是longs/doubles, ints, shorts/chars, bytes, 最后是填充字段, 以满足对其要求。类型1, 用在原始类型后面。类型2, JVM在布局时会尽量使父类对象和子对象挨在一起。 </td></tr>
<tr><td> FilterSpuriousWakeups</td><td>true</td><td>Prevent spurious or premature w keups from object.wait (Solaris only) </td></tr>
<tr><td> FlightRecorderOptions</td><td> </td></tr>
<tr><td> ForceNUMA</td><td>false</td><td>Force NUMA optimizations on single-node UMA systems </td></tr>
<tr><td> ForceTimeHighResolution</td><td>false</td><td>Using high time resolution(For Win32 only) </td></tr>
<tr><td> FreqInlineSize</td><td>325</td><td>经常被调用的方法的进行 inline 的最大 byte c de 大小。 </td></tr>
<tr><td> G1ConcMarkStepDurationMillis</td><td>10.00</td><td>Target duration of indiv dual concurrent marking steps in milliseconds. </td></tr>
<tr><td> G1ConcRSHotCardLimit</td><td>4</td><td>The threshold that defines (>=) a h t card. </td></tr>
<tr><td> G1ConcRSLogCacheSize</td><td>10</td><td>"Log base 2 of the length of conc RS hot-card cache. </td></tr>
<tr><td> G1ConcRefinementGreenZone</td><td>0</td><td>The number of update buffe s that are left in the queue by the concurrent processing threads. Will be selected ergonomica ly by default. </td></tr>
<tr><td> G1ConcRefinementRedZone</td><td>0</td><td>Maximum number of enqueue update buffers before mutator threads start processing new ones instead of enqueueing th m. Will be selected ergonomically by default. Zero will disable concurrent processing. </td>

/tr>
<tr><td> G1ConcRefinementServiceIntervalMillis</td><td>300</td><td>The last concurre
t refinement thread wakes up every specified number of milliseconds to do miscellaneous wo
k. </td></tr>
<tr><td> G1ConcRefinementThreads</td><td>0</td><td>更新RSet的线程组数量。若未设
，则取ParallelGCThreads值。 </td></tr>
<tr><td> G1ConcRefinementThresholdStep</td><td>0</td><td>Each time the rset update
queue increases by this amount activate the next refinement thread if available. Will be select
d ergonomically by default. </td></tr>
<tr><td> G1ConcRefinementYellowZone</td><td>0</td><td>Number of enqueued updat
buffers that will trigger concurrent processing. Will be selected ergonomically by default. </t
></tr>
<tr><td> G1ConfidencePercent</td><td>50</td><td>Confidence coefficient for G1 pause
prediction. Introduced in Java 6 Update 26. </td></tr>
<tr><td> G1HeapRegionSize</td><td>0</td><td>G1区块的大小, 取值为1M至32M. 其取值
要根据最小Heap大小划分出2048个区块。 </td></tr>
<tr><td> G1HeapWastePercent</td><td>5</td><td>控制G1 GC不会回收的空闲内存比例，
认是堆内存的5%。G1 GC在回收过程中会回收所有Region的内存，并持续地做这个工作直到空闲内
比例达到设置的这个值为止，所以对于设置了较大值的堆内存来说，需要采用比较低的比例，这样可
确保较小部分的内存不被回收。 </td></tr>
<tr><td> G1MixedGCCountTarget</td><td>8</td><td>当占用内存超过InitiatingHeapOccu
ancyPercent阀值时， 最多通过多少次Mixed GC来将内存控制在阀值之下。 </td></tr>
<tr><td> G1RSetRegionEntries</td><td>0</td><td>Max number of regions for which we
eep bitmaps. </td></tr>
<tr><td> G1RSetScanBlockSize</td><td>64</td><td>Size of a work unit of cards claimed
y a worker thread during RSet scanning. </td></tr>
<tr><td> G1RSetSparseRegionEntries</td><td>0</td><td>Max number of entries per regi
n in a sparse table.Will be set ergonomically by default. </td></tr>
<tr><td> G1RSetUpdatingPauseTimePercent</td><td>10</td><td>设置更新RSet的目标时
百分比。通过减小该值，G1通常会同时执行更多Rset更新工作。 </td></tr>
<tr><td> G1RefProcDrainInterval</td><td>10</td><td>"The number of discovered refere
ce objects to process before draining concurrent marking work queues. </td></tr>
<tr><td> G1ReservePercent</td><td>10</td><td>设置保留为false上限的堆的百分比（0到5
），以减少G1收集器升级失败的可能性。 </td></tr>
<tr><td> G1SATBBufferEnqueueingThresholdPercent</td><td>60</td><td>"Before enque
eing them, each mutator thread tries to do some filtering on the SATB buffers it generates. If
ost-filtering the percentage of retained entries is over this threshold the buffer will be enque
ed for processing. A value of 0 specifies that mutator threads should not do such filtering. </
d></tr>
<tr><td> G1SATBBufferSize</td><td>1024</td><td>Number of entries in an SATB log buf
er. </td></tr>
<tr><td> G1UpdateBufferSize</td><td>256</td><td>Size of an update buffer </td></tr>
<tr><td> G1UseAdaptiveConcRefinement</td><td>true</td><td>Select green, yellow and
ed zones adaptively to meet the the pause requirements. </td></tr>
<tr><td> GCDrainStackTargetSize</td><td>64</td><td>how many entries we'll try to leav
on the stack during parallel GC </td></tr>
<tr><td> GCHeapFreeLimit</td><td>2</td><td>Heap空闲空间的最低比例下限，当超过下限
，会抛出OOM(HeapSpace)异常 </td></tr>
<tr><td> GCLockerEdenExpansionPercent</td><td>5</td><td>How much the GC can exp
nd the eden by while the GC locker is active (as a percentage) </td></tr>
<tr><td> GCLockerInvokesConcurrent</td><td>false</td><td>加上之后，原本遇上JNI GCL
cker只需要补偿YGC就够的，变成要执行YGC ＋ CMS GC了。 </td></tr>
<tr><td> GCLogFileSize</td><td>8192</td><td>GC日志最大尺寸。 </td></tr>
<tr><td> GCPauseIntervalMillis</td><td>0</td><td>暂停间隔目标。 </td></tr>

<tr><td> GCTaskTimeStampEntries</td><td>200</td><td>Number of time stamp entries er gc worker thread </td></tr>
<tr><td> GCTimeLimit</td><td>98</td><td>花费在GC上的时间上限，当超过该值时，会抛出OM(HeapSpace)异常 </td></tr>
<tr><td> GCTimeRatio</td><td>99</td><td>最大GC占总可用时间比例，若时间过长，会调相应的空间大小（花费在GC上的时间比例不超过1/(1+GCTimeRatio)) </td></tr>
<tr><td> HeapBaseMinAddress</td><td>2^31</td><td>OS specific low limit for heap bas address </td></tr>
<tr><td> HeapDumpAfterFullGC</td><td>false</td><td>实现在Full GC后dump。 </td></t>
<tr><td> HeapDumpBeforeFullGC</td><td>false</td><td>实现在Full GC前dump。 </td><tr>
<tr><td> HeapDumpOnOutOfMemoryError</td><td>false</td><td>在内存出现OOM时，Heap转存到文件以便后续分析，文件名通常是java_pid<pid>.hprof，其中pid为该程序的进程号。 <td></tr>
<tr><td> HeapDumpPath</td><td></td><td> heap转存文件的存储路径。 </td></tr>
<tr><td> HeapFirstMaximumCompactionCount</td><td>3</td><td>The collection count or the first maximum compaction </td></tr>
<tr><td> HeapMaximumCompactionInterval</td><td>20</td><td>How often should we aximally compact the heap (not allowing any dead space) </td></tr>
<tr><td> HeapSizePerGCThread</td><td>87241520</td><td>Size of heap (bytes) per GC hread used in calculating the number of GC threads </td></tr>
<tr><td> IgnoreEmptyClassPaths</td><td>false</td><td> </td></tr>
<tr><td> IgnoreUnrecognizedVMOptions</td><td>false</td><td>让JVM默默地忽略不可识的选项,而不是崩溃。 </td></tr>
<tr><td> IncreaseFirstTierCompileThresholdAt</td><td>50</td><td> </td></tr>
<tr><td> IncrementalInline</td><td>true</td><td>do post parse inlining </td></tr>
<tr><td> InitialBootClassLoaderMetaspaceSize</td><td>4194304</td><td>这个参数决定NoKlass Metaspace的第一个内存Block的大小，即2倍该值，同时为bootstrapClassLoader的第一内存chunk分配了与该值的大小 </td></tr>
<tr><td> InitialCodeCacheSize</td><td>2555904</td><td>设置初始CodeCache大小。 </t></tr>
<tr><td> InitialHeapSize</td><td>0</td><td>JVM初始堆内存，可通过-Xms设置 </td></tr>
<tr><td> InitialRAMFraction</td><td>64</td><td>Fraction (1/n) of real memory used for i itial heap size </td></tr>
<tr><td> InitialSurvivorRatio</td><td>8</td><td>设置吞吐量垃圾收集器使用的初始幸存者间比率（由-XX:+UseParallelGC和/或 - XX:+UseParallelOldGC选项启用）。 </td></tr>
<tr><td> InitialTenuringThreshold</td><td>7</td><td>晋升年龄阈值的最小值。一个对象在Svivor区移动达到晋升年龄阈值后下次MinorGC就进入老年代。JVM会根据YGC与FullGC的耗时来动调整晋升年龄阈值。 </td></tr>
<tr><td> InitiatingHeapOccupancyPercent</td><td>45</td><td>设置启动并发GC循环的堆用百分比（0到100）。它由垃圾收集器使用，它根据整个堆的占用而触发并发GC循环，而不仅仅是中一代（例如，G1垃圾收集器）。 </td></tr>
<tr><td> Inline</td><td>true</td><td>启用方法内联。 </td></tr>
<tr><td> InlineDataFile</td><td> </td></tr>
<tr><td> InlineSmallCode</td><td>1000</td><td>设置应内联的已编译方法的最大代码大小以字节为单位）。 </td></tr>
<tr><td> InlineSynchronizedMethods</td><td>true</td><td>是否允许内联synchronized m thods。 </td></tr>
<tr><td> InsertMemBarAfterArraycopy</td><td>true</td><td>Insert memory barrier after arraycopy call </td></tr>
<tr><td> InteriorEntryAlignment</td><td>16</td><td>Code alignment for interior entry p ints in generated code (in bytes) </td></tr>
<tr><td> InterpreterProfilePercentage</td><td>33</td><td>解释器监控比率 </td></tr>

<tr><td> JNIDetachReleasesMonitors</td><td>true</td><td>JNI DetachCurrentThread rel
ases monitors owned by thread </td></tr>
<tr><td> JavaMonitorsInStackTrace</td><td>true</td><td>标记启用锁定行。 </td></tr>
<tr><td> JavaPriority10_To_OSPriority</td><td>-1</td><td>Map Java priorities to OS prior
ties </td></tr>
<tr><td> JavaPriority1_To_OSPriority</td><td>-1</td><td>Map Java priorities to OS priorit
es </td></tr>
<tr><td> JavaPriority2_To_OSPriority</td><td>-1</td><td>Map Java priorities to OS priorit
es </td></tr>
<tr><td> JavaPriority3_To_OSPriority</td><td>-1</td><td>Map Java priorities to OS priorit
esMap Java priorities to OS priorities </td></tr>
<tr><td> JavaPriority4_To_OSPriority</td><td>-1</td><td>Map Java priorities to OS priorit
es </td></tr>
<tr><td> JavaPriority5_To_OSPriority</td><td>-1</td><td>Map Java priorities to OS priorit
es </td></tr>
<tr><td> JavaPriority6_To_OSPriority</td><td>-1</td><td>Map Java priorities to OS priorit
es </td></tr>
<tr><td> JavaPriority7_To_OSPriority</td><td>-1</td><td>Map Java priorities to OS priorit
es </td></tr>
<tr><td> JavaPriority8_To_OSPriority</td><td>-1</td><td>Map Java priorities to OS priorit
es </td></tr>
<tr><td> JavaPriority9_To_OSPriority</td><td>-1</td><td>Map Java priorities to OS priorit
es </td></tr>
<tr><td> LIRFillDelaySlots</td><td>false</td><td>fill delays on on SPARC with LIR </td><
tr>
<tr><td> LargePageHeapSizeThreshold</td><td>134217728</td><td>Use large pages if
ax heap is at least this big </td></tr>
<tr><td> LargePageSizeInBytes</td><td>0</td><td>设置大内存页的大小，在启用大内存页
持(UseLargePages)时生效 </td></tr>
<tr><td> LazyBootClassLoader</td><td>true</td><td>Enable/disable lazy opening of boo
 class path entries </td></tr>
<tr><td> LiveNodeCountInliningCutoff</td><td>40000</td><td>max number of live node
 in a method </td></tr>
<tr><td> LogCommercialFeatures</td><td>false</td><td> </td></tr>
<tr><td> LoopMaxUnroll</td><td>16</td><td> </td></tr>
<tr><td> LoopOptsCount</td><td>43</td><td>Set level of loop optimization for tier 1 c
mpiles </td></tr>
<tr><td> LoopUnrollLimit</td><td>60</td><td>Unroll loop bodies with server compiler in
ermediate representation node count less than this value. The limit used by the server compil
r is a function of this value, not the actual value. The default value varies with the platform on
which the JVM is running. </td></tr>
<tr><td> LoopUnrollMin</td><td>4</td><td> </td></tr>
<tr><td> LoopUnswitching</td><td>true</td><td>开启循环判断外提。这是一种编译器优化
法。 </td></tr>
<tr><td> ManagementServer</td><td>false</td><td>Minimum number of unroll loop bo
ies before checking progress of rounds of unroll,optimize,.. </td></tr>
<tr><td> MarkStackSize</td><td>4194304</td><td>Size of marking stack </td></tr>
<tr><td> MarkStackSizeMax</td><td>2^29</td><td>Max size of marking stack </td></t
>
<tr><td> MarkSweepAlwaysCompactCount</td><td>4</td><td>在没有明显的压缩效果之
，我们允许一些垃圾对象移动到内存区域的底部，即开始位置，每进行MarkSweepAlwaysCompact
ount(默认4次)FGC时，再进行一次完全压缩。 </td></tr>
<tr><td> MarkSweepDeadRatio</td><td>5</td><td>每经过MarkSweepAlwaysCompactCo
nt次GC，就允许当前区域空间的MarkSweepDeadRatio%(TenuredSpace)/PermMarkSweepDead

atio%(ContigPermSpace)大小被用来将死亡对象当做存活对象处理。 </td></tr>
<tr><td> MaxBCEAEstimateLevel</td><td>5</td><td>Maximum number of nested calls th t are analyzed by BC EA. </td></tr>
<tr><td> MaxBCEAEstimateSize</td><td>150</td><td>Maximum bytecode size of a meth d to be analyzed by BC EA. </td></tr>
<tr><td> MaxDirectMemorySize</td><td>0</td><td>指定直接内存的最大大小，若未指定，默认与Java堆最大值一样。 </td></tr>
<tr><td> MaxFDLimit</td><td>true</td><td>设置java进程可用文件描述符为操作系统允许的大值。 </td></tr>
<tr><td> MaxGCMinorPauseMillis</td><td>2^64-1</td><td>Adaptive size policy maxim m GC minor pause time goal in msec </td></tr>
<tr><td> MaxGCPauseMillis</td><td>2^64-1</td><td>最大的GC暂停时间，如果时间过长会相应调整空间的大小（单位为毫秒） </td></tr>
<tr><td> MaxHeapFreeRatio</td><td>70</td><td>堆最大空闲阈值。GC后如果发现堆内存闲占预估堆的比例超过该值，则收缩堆内存的评估最大值。预估堆内存是堆大小动态调控的重要选项一. 堆内存预估最大值一定小于或等于固定最大值(-Xmx指定的数值)， 前者会根据使用情况动态调大缩小, 以提高GC回收的效率。 </td></tr>
<tr><td> MaxHeapSize</td><td>130862280</td><td>JVM最大堆内存，可通过-Xmx设置 </d></tr>
<tr><td> MaxInlineLevel</td><td>9</td><td>针对嵌套调用的最大内联深度。 </td></tr>
<tr><td> MaxInlineSize</td><td>35</td><td>方法可以被内联的最大bytecode大小。 </td> /tr>
<tr><td> MaxJNILocalCapacity</td><td>65536</td><td> </td></tr>
<tr><td> MaxJavaStackTraceDepth</td><td>1024</td><td>该参数的目的是让JVM在实现 Th owable.fillInStackTrace()时把整个调用栈上所有Java栈帧的信息都记录下来。也就是说fillInStackTra e()在记录到最多MaxJavaStackTraceDepth层调用栈帧信息后就会停止爬栈。这是因为爬栈并将栈信息转换为Java对象（StackTraceElement）是个比较慢的动作，限制一下最大深度可以把fillInStack race()的最大开销给限制住。 </td></tr>
<tr><td> MaxJumpTableSize</td><td>65000</td><td>Maximum number of targets in a g nerated jump table </td></tr>
<tr><td> MaxJumpTableSparseness</td><td>5</td><td>HotSpot JVM can generate jump ables even for switches with gaps and with outliers. -XX:MaxJumpTableSparseness controls h w large the gaps can be. E.g. if there are labels from 1 to 10, then from 13 to 20 and the last l bel with the value 99 - JIT will generate a guard test for the value 99, and for the rest labels it will create a table. </td></tr>
<tr><td> MaxLabelRootDepth</td><td>1100</td><td>Maximum times call Label_Root to prevent stack overflow </td></tr>
<tr><td> MaxLoopPad</td><td>15</td><td>Align a loop if padding size in bytes is less or equal to this value </td></tr>
<tr><td> MaxMetaspaceExpansion</td><td>5452592</td><td>增大触发metaspace GC阈的最大要求。 </td></tr>
<tr><td> MaxMetaspaceFreeRatio</td><td>70</td><td>GC后触发metaspace扩容的最小空比例。 </td></tr>
<tr><td> MaxMetaspaceSize</td><td>2^64-1</td><td>metaspace最大空间大小 </td></t >
<tr><td> MaxNewSize</td><td>2^64-1</td><td>年轻代的最大大小 </td></tr>
<tr><td> MaxNodeLimit</td><td>80000</td><td>设置单个方法编译期间要使用的最大节点。 </td></tr>
<tr><td> MaxRAM</td><td>2^37</td><td> this parameter overrides the actual amount of physical RAM when calculating the heap limits basing on ergonomics. </td></tr>
<tr><td> MaxRAMFraction</td><td>4</td><td>默认值为4，即堆的最大容量是MaxRAM值四分之一。 </td></tr>
<tr><td> MaxRecursiveInlineLevel</td><td>1</td><td>the full stack is searched for recurs ve calls so mutual recursion may also now be blocked from going past MaxRecursiveInlineLev

l.it counts both direct and indirect recursive calls. </td></tr>
<tr><td> MaxTenuringThreshold</td><td>15</td><td>晋升年龄阈值的最大值。一个对象在Suvivor区移动达到晋升年龄阈值后下次MinorGC就进入老年代。如果设置为0，则年轻代对象不经过Suvvor区，直接进入老年代，对于需要大量常驻内存的应用，这样做可以提高效率。如果将该值设置为个将大的值，则年轻代对象会在Suvivor区进行多次复制，这样可以增加对象在年轻代存活时间，增对象在年轻代被垃圾回收的概率，减少Full GC的频率，可以在某种程度上提高服务稳定性。 </td></r>
<tr><td> MaxTrivialSize</td><td>6</td><td>设置要内联的简单方法的最大字节码大小（以字为单位）。</td></tr>
<tr><td> MaxVectorSize</td><td>32</td><td>Max vector size in bytes, actual size could e less depending on elements type </td></tr>
<tr><td> MetaspaceSize</td><td>21810376</td><td>metaspace初始大小，达到该值就会发垃圾回收器进行类型卸载，同时GC会对该值进行调整：如果释放了大量空间，就适当降低该值;如释放了很少空间，那么在不超过MaxMetaspaceSize时适当提高该值 </td></tr>
<tr><td> MethodFlushing</td><td>true</td><td>Reclamation of zombie and not-entrant methods </td></tr>
<tr><td> MinHeapDeltaBytes</td><td>170392</td><td>为了防止频繁扩展内存代空间,每次展内存代时都有一个最小值，由该参数定义。 </td></tr>
<tr><td> MinHeapFreeRatio</td><td>40</td><td>堆最小空闲阈值。GC后如果发现空闲堆内占到整个预估堆内存的N%(百分比), 则放大堆内存的预估最大值。 </td></tr>
<tr><td> MinInliningThreshold</td><td>250</td><td>方法可以被内联的最小调用次数。 </t></tr>
<tr><td> MinJumpTableSize</td><td>10</td><td>case的数量小于这个不会生成jump_table </td></tr>
<tr><td> MinMetaspaceExpansion</td><td>340784</td><td>增大触发metaspace GC阈值最小要求。 </td></tr>
<tr><td> MinMetaspaceFreeRatio</td><td>40</td><td>触发metaspaceGC的最小空闲阈值表示每次GC完之后，假设我们允许接下来metaspace可以继续被commit的内存占到了被commit之总共committed的内存量的MinMetaspaceFreeRatio%，如果这个总共被committed的量比当前触metaspaceGC的阈值要大，那么将尝试做扩容，也就是增大触发metaspaceGC的阈值，不过这个增至少是MinMetaspaceExpansion才会做，不然不会增加这个阈值。这个参数主要是为了避免触发metspaceGC的阈值和gc之后committed的内存的量比较接近，于是将这个阈值进行扩大。一般情况下在c完之后，如果被committed的量还是比较大的时候，换个说法就是离触发metaspaceGC的阈值比较近的时候，这个调整会比较明显。 </td></tr>
<tr><td> MinRAMFraction</td><td>2</td><td>JVM需要确保预留足够的内存给操作系统使。JVM占用的内容最大不超过物理内存除以该值。 </td></tr>
<tr><td> MinSurvivorRatio</td><td>3</td><td>Minimum ratio of young generation/survior space size </td></tr>
<tr><td> MinTLABSize</td><td>2048</td><td>Minimum allowed TLAB size (in bytes) </t></tr>
<tr><td> MonitorBound</td><td>0</td><td> </td></tr>
<tr><td> MonitorInUseLists</td><td>false</td><td>在STW时，所有的Java线程都会暂停在安全点(SafePoint)”，此时VMThread通过对所有Monitor的遍历。该值表示启用对当前正在“使用中的Monitor子序列进行遍历，从而得到哪些未被使用的“Monitor”作为降级对象。 </td></tr>
<tr><td> MultiArrayExpandLimit</td><td>6</td><td>Maximum number of individual allo ations in an inline-expanded multianewarray instruction </td></tr>
<tr><td> MustCallLoadClassInternal</td><td>false</td><td>loadClassInternal() 替代 loadCass() </td></tr>
<tr><td> NUMAChunkResizeWeight</td><td>20</td><td>"Percentage (0-100) used to we ght the current sample when "computing exponentially decaying average for" AdaptiveNUMChunkSizing </td></tr>
<tr><td> NUMAInterleaveGranularity</td><td>2097152</td><td>Granularity to use for NMA interleaving on Windows OS </td></tr>
<tr><td> NUMAPageScanRate</td><td>256</td><td>Maximum number of pages to incl

de in the page scan procedure </td></tr>
<tr><td> NUMASpaceResizeRate</td><td>2^30</td><td>Do not reallocate more that this amount per collection </td></tr>
<tr><td> NUMAStats</td><td>false</td><td>Print NUMA stats in detailed heap informati n </td></tr>
<tr><td> NativeMemoryTracking</td><td>off</td><td>该参数开启NMT。Native Memory racking (NMT) 是Hotspot VM用来分析VM内部内存使用情况的一个功能。 </td></tr>
<tr><td> NeedsDeoptSuspend</td><td>false</td><td>True for register window machines (sparc/ia64) </td></tr>
<tr><td> NeverActAsServerClassMachine</td><td>false</td><td>Never act like a server-c ass machine </td></tr>
<tr><td> NeverTenure</td><td>false</td><td>永不晋升 </td></tr>
<tr><td> NewRatio</td><td>2</td><td>年轻代与老年代的比例 </td></tr>
<tr><td> NewSize</td><td>1363144</td><td>默认年轻代的大小 </td></tr>
<tr><td> NewSizeThreadIncrease</td><td>5320</td><td>允许您指定每个活动线程会增加少初期对象空间。这个选项在调节由于线程增加而增加的分配率时可能会有用。 </td></tr>
<tr><td> NmethodSweepActivity</td><td>10</td><td> </td></tr>
<tr><td> NmethodSweepCheckInterval</td><td>5</td><td>Compilers wake up every n se onds to possibly sweep nmethods </td></tr>
<tr><td> NmethodSweepFraction</td><td>16</td><td>Number of invocations of sweepe to cover all nmethods </td></tr>
<tr><td> NodeLimitFudgeFactor</td><td>2000</td><td>Fudge Factor for certain optimiz tions </td></tr>
<tr><td> NumberOfGCLogFiles</td><td>0</td><td>Number of gclog files in rotation, Def ult: 0, no rotation </td></tr>
<tr><td> NumberOfLoopInstrToAlign</td><td>4</td><td>Number of first instructions in a loop to align </td></tr>
<tr><td> ObjectAlignmentInBytes</td><td>8</td><td>Default object alignment in bytes, is minimum </td></tr>
<tr><td> OldPLABSize</td><td>1024</td><td>Size of old gen promotion labs (in HeapW rds) </td></tr>
<tr><td> OldPLABWeight</td><td>50</td><td>Percentage (0-100) used to weight the cur ent sample when computing exponentially decaying average for resizing CMSParPromoteBlo ksToClaim </td></tr>
<tr><td> OldSize</td><td>5452592</td><td>老年代的默认大小 </td></tr>
<tr><td> OmitStackTraceInFastThrow</td><td>true</td><td>Omit backtraces for some 'h t' exceptions in optimized code </td></tr>
<tr><td> OnError</td><td></td><td> Run user-defined commands on fatal error </td></r>
<tr><td> OnOutOfMemoryError</td><td></td><td> 指定一个脚本路径，当OOM时，去执该脚本。 </td></tr>
<tr><td> OnStackReplacePercentage</td><td>140</td><td>方法中循环执行部分代码的执次数触发OSR编译时的阈值。 </td></tr>
<tr><td> OptimizeFill</td><td>true</td><td>convert fill/copy loops into intrinsic </td></r>
<tr><td> OptimizePtrCompare</td><td>true</td><td>Use escape analysis to optimize po nters compare </td></tr>
<tr><td> OptimizeStringConcat</td><td>true</td><td>启用String串联操作的优化。 </td>
/tr>
<tr><td> OptoBundling</td><td>false</td><td>Generate nops to fill i-cache lines </td><
tr>
<tr><td> OptoLoopAlignment</td><td>16</td><td>Align inner loops to zero relative to th s modulus </td></tr>
<tr><td> OptoScheduling</td><td>false</td><td>Instruction Scheduling after register all

cation </td></tr>
<tr><td> PLABWeight</td><td>75</td><td>Percentage (0-100) used to weight the current sample when computing exponentially decaying average for ResizePLAB. </td></tr>
<tr><td> PSChunkLargeArrays</td><td>true</td><td>true: process large arrays in chunks </td></tr>
<tr><td> ParGCArrayScanChunk</td><td>50</td><td>Scan a subset and push remainder, f array is bigger than this </td></tr>
<tr><td> ParGCDesiredObjsFromOverflowList</td><td>20</td><td>The desired number o objects to claim from the overflow list </td></tr>
<tr><td> ParGCTrimOverflow</td><td>true</td><td>Eagerly trim the local overflow lists (hen ParGCUseLocalOverflow </td></tr>
<tr><td> ParGCUseLocalOverflow</td><td>false</td><td>Instead of a global overflow list, use local overflow stacks </td></tr>
<tr><td> ParallelGCBufferWastePct</td><td>10</td><td>wasted fraction of parallel allocat on buffer </td></tr>
<tr><td> ParallelGCThreads</td><td>0</td><td>并行垃圾回收的线程数。默认的配置是如果 理器的个数小于8，那么就是处理器的个数；如果处理器大于8，它的值就是3+5N/8。 </td></tr>
<tr><td> ParallelGCVerbose</td><td>false</td><td>Verbose output for parallel GC. </td </tr>
<tr><td> ParallelOldDeadWoodLimiterMean</td><td>50</td><td>The mean used by the ar compact dead wood limiter (a number between 0-100). </td></tr>
<tr><td> ParallelOldDeadWoodLimiterStdDev</td><td>80</td><td>The standard deviati n used by the par compact dead wood limiter (a number between 0-100). </td></tr>
<tr><td> ParallelRefProcBalancingEnabled</td><td>true</td><td>Enable balancing of refe ence processing queues </td></tr>
<tr><td> ParallelRefProcEnabled</td><td>false</td><td>并行处理Reference，加快处理速 ，缩短耗时。 </td></tr>
<tr><td> PartialPeelAtUnsignedTests</td><td>true</td><td>Partial peel at unsigned tests f no signed test exists </td></tr>
<tr><td> PartialPeelLoop</td><td>true</td><td>Partial peel (rotate) loops </td></tr>
<tr><td> PartialPeelNewPhiDelta</td><td>0</td><td>Additional phis that can be created y partial peeling </td></tr>
<tr><td> PausePadding</td><td>1</td><td>How much buffer to keep for pause time </t ></tr>
<tr><td> PerBytecodeRecompilationCutoff</td><td>200</td><td>Per-BCI limit on repeat d recompilation (-1=>'Inf') </td></tr>
<tr><td> PerBytecodeTrapLimit</td><td>4</td><td>Limit on traps (of one kind) at a parti ular BCI </td></tr>
<tr><td> PerMethodRecompilationCutoff</td><td>400</td><td>After recompiling N time , stay in the interpreter (-1=>'Inf') </td></tr>
<tr><td> PerMethodTrapLimit</td><td>100</td><td>Limit on traps (of one kind) in a met od (includes inlines) </td></tr>
<tr><td> PerfAllowAtExitRegistration</td><td>false</td><td>Allow registration of atexit() methods </td></tr>
<tr><td> PerfBypassFileSystemCheck</td><td>false</td><td>Bypass Win32 file system cri eria checks (Windows Only) </td></tr>
<tr><td> PerfDataMemorySize</td><td>32768</td><td>Size of performance data memor region. Will be rounded up to a multiple of the native os page size. </td></tr>
<tr><td> PerfDataSamplingInterval</td><td>50</td><td>Data sampling interval in millise onds </td></tr>
<tr><td> PerfDataSaveFile</td><td></td><td> Save PerfData memory to the specified abs lute pathname, %p in the file name if present will be replaced by pid </td></tr>
<tr><td> PerfDataSaveToFile</td><td>false</td><td>Save PerfData memory to hsperfdata <pid> file on exit </td></tr>

<tr><td> PerfDisableSharedMem</td><td>false</td><td>Store performance data in standrd memory </td></tr>
<tr><td> PerfMaxStringConstLength</td><td>1024</td><td>Maximum PerfStringConstan string length before truncation </td></tr>
<tr><td> PreInflateSpin</td><td>10</td><td>Number of times to spin wait before inflatio </td></tr>
<tr><td> PreferInterpreterNativeStubs</td><td>false</td><td>Use always interpreter stub for native methods invoked via interpreter </td></tr>
<tr><td> PrefetchCopyIntervalInBytes</td><td>-1</td><td>How far ahead to prefetch destination area (<= 0 means off) </td></tr>
<tr><td> PrefetchFieldsAhead</td><td>-1</td><td>How many fields ahead to prefetch in oop scan (<= 0 means off) </td></tr>
<tr><td> PrefetchScanIntervalInBytes</td><td>-1</td><td>How far ahead to prefetch sca area (<= 0 means off) </td></tr>
<tr><td> PreserveAllAnnotations</td><td>false</td><td>Preserve RuntimeInvisibleAnnotaions as well as RuntimeVisibleAnnotations </td></tr>
<tr><td> PreserveFramePointer</td><td>false</td><td> </td></tr>
<tr><td> PretenureSizeThreshold</td><td>0</td><td>设置让大于该阈值的对象直接分配在年代（只对Serial、ParNew收集器有效），单位为字节 </td></tr>
<tr><td> PrintAdaptiveSizePolicy</td><td>false</td><td>允许打印有关自适应生成大小的信。 </td></tr>
<tr><td> PrintCMSInitiationStatistics</td><td>false</td><td>Statistics for initiating a CMS collection </td></tr>
<tr><td> PrintCMSStatistics</td><td>0</td><td>Statistics for CMS </td></tr>
<tr><td> PrintClassHistogram</td><td>false</td><td>在Control+C事件（SIGTERM）之后用类实例直方图的打印。 </td></tr>
<tr><td> PrintClassHistogramAfterFullGC</td><td>false</td><td>在FullGC后打印类实例直图。 </td></tr>
<tr><td> PrintClassHistogramBeforeFullGC</td><td>false</td><td>在FullGC前打印类实例方图。 </td></tr>
<tr><td> PrintCodeCache</td><td>false</td><td>在jvm停止的时候打印出codeCache的使情况。 </td></tr>
<tr><td> PrintCodeCacheOnCompilation</td><td>false</td><td>用于在方法每次被编译时出code cache的使用情况。 </td></tr>
<tr><td> PrintCommandLineFlags</td><td>false</td><td>Print flags specified on command line or set by ergonomics </td></tr>
<tr><td> PrintCompilation</td><td>false</td><td>每次编译方法时，通过向控制台打印消息从JVM启用详细诊断输出。这使您可以查看实际编译的方法。 </td></tr>
<tr><td> PrintConcurrentLocks</td><td>false</td><td>java.util.concurrent在Control+C事（SIGTERM）之后启用锁的打印。设置此选项等同于运行jstack -l命令或jcmd pid Thread.print -l令，其中pid是当前Java进程标识符。 </td></tr>
<tr><td> PrintFLSCensus</td><td>0</td><td>Census for CMS' FreeListSpace </td></tr>
<tr><td> PrintFLSStatistics</td><td>0</td><td>若该参数不为0，那么每次GC前后都会调用reortFreeListStatistics()方法打印出free list的统计信息。 </td></tr>
<tr><td> PrintFlagsFinal</td><td>false</td><td>打印出XX选项在运行程序时生效的值 </td>/tr>
<tr><td> PrintFlagsInitial</td><td>false</td><td>打印出所有XX选项的默认值 </td></tr>
<tr><td> PrintGC</td><td>false</td><td>等同于-verbose:gc表示打开简化的GC日志 </td><tr>
<tr><td> PrintGCApplicationConcurrentTime</td><td>false</td><td>允许打印自上次暂停经过的时间（例如，GC暂停）。 </td></tr>
<tr><td> PrintGCApplicationStoppedTime</td><td>false</td><td>打印GC时线程的停顿时间 </td></tr>
<tr><td> PrintGCCause</td><td>true</td><td>Include GC cause in GC logging </td></tr

<tr><td> PrintGCDateStamps</td><td>false</td><td>打印出GC发生的具体时间 </td></tr>
<tr><td> PrintGCDetails</td><td>false</td><td>在发生垃圾回收时打印内存回收日志，并在程退出时输出当前内存各区域分配情况。 </td></tr>
<tr><td> PrintGCID</td><td>false</td><td> </td></tr>
<tr><td> PrintGCTaskTimeStamps</td><td>false</td><td>允许为每个GC工作线程任务打印间戳。 </td></tr>
<tr><td> PrintGCTimeStamps</td><td>false</td><td>打印GC发生时相对于JVM启动时的时。 </td></tr>
<tr><td> PrintHeapAtGC</td><td>false</td><td>每次GC前和GC后，都打印堆信息。 </td>
/tr>
<tr><td> PrintHeapAtGCExtended</td><td>false</td><td>Prints extended information ab
ut the layout of the heap when -XX:+PrintHeapAtGC is set </td></tr>
<tr><td> PrintHeapAtSIGBREAK</td><td>true</td><td>Print heap layout in response to S
GBREAK </td></tr>
<tr><td> PrintJNIGCStalls</td><td>false</td><td>Print diagnostic message when GC is sta
led by JNI critical section </td></tr>
<tr><td> PrintJNIResolving</td><td>false</td><td>Used to implement -v:jni </td></tr>
<tr><td> PrintOldPLAB</td><td>false</td><td>Print (old gen) promotion labs sizing decis
ons </td></tr>
<tr><td> PrintOopAddress</td><td>false</td><td>Always print the location of the oop </
d></tr>
<tr><td> PrintPLAB</td><td>false</td><td>Print (survivor space) promotion labs sizing d
cisions </td></tr>
<tr><td> PrintParallelOldGCPhaseTimes</td><td>false</td><td>Print the time taken by ea
h parallel old gc phase. PrintGCDetails must also be enabled. </td></tr>
<tr><td> PrintPromotionFailure</td><td>false</td><td>晋升失败时打印日志。 </td></tr>
<tr><td> PrintReferenceGC</td><td>false</td><td>Print times spent handling reference
bjects during GC (enabled only when PrintGCDetails) </td></tr>
<tr><td> PrintSafepointStatistics</td><td>false</td><td>print statistics about safepoint s
nchronization </td></tr>
<tr><td> PrintSafepointStatisticsCount</td><td>300</td><td>total number of safepoint s
atistics collected before printing them out </td></tr>
<tr><td> PrintSafepointStatisticsTimeout</td><td>-1</td><td>print safepoint statistics on
y when safepoint takes more than PrintSafepointSatisticsTimeout in millis </td></tr>
<tr><td> PrintSharedArchiveAndExit</td><td>false</td><td> </td></tr>
<tr><td> PrintSharedDictionary</td><td>false</td><td> </td></tr>
<tr><td> PrintSharedSpaces</td><td>false</td><td>Print usage of shared spaces </td></
r>
<tr><td> PrintStringDeduplicationStatistics</td><td>false</td><td>打印详细的重复数据删统计信息。 </td></tr>
<tr><td> PrintStringTableStatistics</td><td>false</td><td>print statistics about the String
able and SymbolTable </td></tr>
<tr><td> PrintTLAB</td><td>false</td><td>Print various TLAB related information </td><
tr>
<tr><td> PrintTenuringDistribution</td><td>false</td><td>让JVM在每次MinorGC后打印当前使用的Survivor中对象的年龄分布。 </td></tr>
<tr><td> PrintTieredEvents</td><td>false</td><td>Print tiered events notifications </td>
/tr>
<tr><td> PrintVMOptions</td><td>false</td><td>Print flags that appeared on the comm
nd line </td></tr>
<tr><td> PrintVMQWaitTime</td><td>false</td><td>Prints out the waiting time in VM op
ration queue </td></tr>
<tr><td> PrintWarnings</td><td>true</td><td>Prints JVM warnings to output stream </t

></tr>
<tr><td> ProcessDistributionStride</td><td>4</td><td>Stride through processors when d
stributing processes </td></tr>
<tr><td> ProfileInterpreter</td><td>true</td><td>Profile at the bytecode level during inte
pretation </td></tr>
<tr><td> ProfileIntervals</td><td>false</td><td>Prints profiles for each interval (see Profil
IntervalsTicks) </td></tr>
<tr><td> ProfileIntervalsTicks</td><td>100</td><td># of ticks between printing of interval
profile (+ProfileIntervals) </td></tr>
<tr><td> ProfileMaturityPercentage</td><td>20</td><td>number of method invocations
branches (expressed as % of CompileThreshold) before using the method's profile </td></t
>
<tr><td> ProfileVM</td><td>false</td><td>Profiles ticks that fall within VM (either in the
M Thread or VM code called through stubs) </td></tr>
<tr><td> ProfilerPrintByteCodeStatistics</td><td>false</td><td>Prints byte code statictics
when dumping profiler output </td></tr>
<tr><td> ProfilerRecordPC</td><td>false</td><td>Collects tick for each 16 byte interval of
compiled code </td></tr>
<tr><td> PromotedPadding</td><td>3</td><td>How much buffer to keep for promotion
ailure </td></tr>
<tr><td> QueuedAllocationWarningCount</td><td>0</td><td>Number of times an allocat
on that queues behind a GC will retry before printing a warning </td></tr>
<tr><td> RTMRetryCount</td><td>5</td><td>RTM锁定代码将在中止或忙碌时重试此选项指
的次数，然后再回退到正常锁定机制。必须启用UseRTMLocking才能生效。 </td></tr>
<tr><td> RangeCheckElimination</td><td>true</td><td>Split loop iterations to eliminate
ange checks </td></tr>
<tr><td> ReadPrefetchInstr</td><td>0</td><td>Prefetch instruction to prefetch ahead </
d></tr>
<tr><td> ReassociateInvariants</td><td>true</td><td>Enable reassociation of expressions
with loop invariants. </td></tr>
<tr><td> ReduceBulkZeroing</td><td>true</td><td>When bulk-initializing, try to avoid n
edless zeroing </td></tr>
<tr><td> ReduceFieldZeroing</td><td>true</td><td>When initializing fields, try to avoid
eedless zeroing </td></tr>
<tr><td> ReduceInitialCardMarks</td><td>true</td><td>When initializing fields, try to avo
d needless card marks </td></tr>
<tr><td> ReduceSignalUsage</td><td>false</td><td>Reduce the use of OS signals in Java
and/or the VM </td></tr>
<tr><td> RefDiscoveryPolicy</td><td>0</td><td>Whether reference-based(0) or referent
based(1) </td></tr>
<tr><td> ReflectionWrapResolutionErrors</td><td>true</td><td>Temporary flag for transi
ion to AbstractMethodError wrapped in InvocationTargetException. </td></tr>
<tr><td> RegisterFinalizersAtInit</td><td>true</td><td>Register finalizable objects at end
of Object.<init> or after allocation </td></tr>
<tr><td> RelaxAccessControlCheck</td><td>false</td><td>在Class校验器中，放松对访问
制的检查,作用与reflection里的setAccessible类似。 </td></tr>
<tr><td> ReplayDataFile</td><td> </td></tr>
<tr><td> RequireSharedSpaces</td><td>false</td><td>Require shared spaces in the per
anent generation </td></tr>
<tr><td> ReservedCodeCacheSize</td><td>50331648</td><td>设置JIT编译代码的最大代
缓存大小（以字节为单位）。 </td></tr>
<tr><td> ResizeOldPLAB</td><td>true</td><td>Dynamically resize (old gen) promotion l
bs </td></tr>
<tr><td> ResizePLAB</td><td>true</td><td>Dynamically resize (survivor space) promotio

labs </td></tr>
<tr><td> ResizeTLAB</td><td>true</td><td>Dynamically resize tlab size for threads </td
</tr>
<tr><td> RestoreMXCSROnJNICalls</td><td>false</td><td>Restore MXCSR when returni
g from JNI calls </td></tr>
<tr><td> RestrictContended</td><td>true</td><td> </td></tr>
<tr><td> RewriteBytecodes</td><td>true</td><td>Allow rewriting of bytecodes (bytecod
s are not immutable) </td></tr>
<tr><td> RewriteFrequentPairs</td><td>true</td><td>Rewrite frequently used bytecode
airs into a single bytecode </td></tr>
<tr><td> SafepointPollOffset</td><td>256</td><td>Offset added to polling address (Intel
only) </td></tr>
<tr><td> SafepointSpinBeforeYield</td><td>2000</td><td>(Unstable) </td></tr>
<tr><td> SafepointTimeout</td><td>false</td><td>启用线程Safepoint超时 </td></tr>
<tr><td> SafepointTimeoutDelay</td><td>10000</td><td>线程Safepoint超时时间，单位
秒 </td></tr>
<tr><td> ScavengeBeforeFullGC</td><td>true</td><td>在每个完整GC之前启用年轻代的GC
</td></tr>
<tr><td> SelfDestructTimer</td><td>0</td><td>Will cause VM to terminate after a given t
me (in minutes) (0 means off) </td></tr>
<tr><td> SharedBaseAddress</td><td>2^35</td><td> </td></tr>
<tr><td> SharedClassListFile</td><td> </td></tr>
<tr><td> SharedMiscCodeSize</td><td>122880</td><td>Size of the shared code area adj
cent to the heap (in bytes) </td></tr>
<tr><td> SharedMiscDataSize</td><td>4194304</td><td>Size of the shared data area adj
cent to the heap (in bytes) </td></tr>
<tr><td> SharedReadOnlySize</td><td>16777216</td><td>Size of read-only space in pe
manent generation (in bytes) </td></tr>
<tr><td> SharedReadWriteSize</td><td>16777216</td><td>Size of read-write space in pe
manent generation (in bytes)" </td></tr>
<tr><td> ShowMessageBoxOnError</td><td>false</td><td>Keep process alive on VM fata
error </td></tr>
<tr><td> SoftRefLRUPolicyMSPerMB</td><td>1000</td><td>设置软引用在上次使用后在堆
保持活动状态的时间（以毫秒为单位）。 </td></tr>
<tr><td> SpecialEncodeISOArray</td><td>true</td><td> </td></tr>
<tr><td> SplitIfBlocks</td><td>true</td><td>Clone compares and control flow through
erge points to fold some branches </td></tr>
<tr><td> StackRedPages</td><td>1</td><td>Number of red zone (unrecoverable overflo
s) pages </td></tr>
<tr><td> StackShadowPages</td><td>20</td><td>Number of shadow zone (for overflow
hecking) pages this should exceed the depth of the VM and native call stack </td></tr>
<tr><td> StackTraceInThrowable</td><td>true</td><td> </td></tr>
<tr><td> StackYellowPages</td><td>2</td><td>Number of yellow zone (recoverable overf
ows) pages </td></tr>
<tr><td> StartAttachListener</td><td>false</td><td>Always start Attach Listener at VM st
rtup </td></tr>
<tr><td> StarvationMonitorInterval</td><td>200</td><td>Pause between each check in
s </td></tr>
<tr><td> StressLdcRewrite</td><td>false</td><td>Force ldc -> ldc_w rewrite during Redef
neClasses </td></tr>
<tr><td> StringDeduplicationAgeThreshold</td><td>3</td><td>String达到指定年龄的对
被视为重复数据删除的候选对象。对象的年龄是对垃圾收集存活多少次的度量。 </td></tr>
<tr><td> StringTableSize</td><td>60013</td><td>Number of buckets in the interned Stri
g table </td></tr>

<tr><td> SuppressFatalErrorMessage</td><td>false</td><td>Do NO Fatal Error report [A
oid deadlock] </td></tr>
<tr><td> SurvivorPadding</td><td>3</td><td>How much buffer to keep for survivor overf
ow </td></tr>
<tr><td> SurvivorRatio</td><td>8</td><td>Eden Space与Suvivor Space的比例，默认配置
，每个Suvivor Space占整个年轻代的1/10 </td></tr>
<tr><td> SuspendRetryCount</td><td>50</td><td>Maximum retry count for an external
uspend request </td></tr>
<tr><td> SuspendRetryDelay</td><td>5</td><td>Milliseconds to delay per retry (* curren
_retry_count) </td></tr>
<tr><td> SyncFlags</td><td>0</td><td>(Unsafe,Unstable) Experimental Sync flags </td>
/tr>
<tr><td> SyncKnobs</td><td></td><td> (Unstable) Various monitor synchronization tuna
les </td></tr>
<tr><td> SyncVerbose</td><td>0</td><td>(Unstable) </td></tr>
<tr><td> TLABAllocationWeight</td><td>35</td><td>Allocation averaging weight </td>
/tr>
<tr><td> TLABRefillWasteFraction</td><td>64</td><td>Max TLAB waste at a refill (interna
fragmentation) </td></tr>
<tr><td> TLABSize</td><td>0</td><td>设置线程局部分配缓冲区（TLAB）的初始大小（以字
为单位）。 </td></tr>
<tr><td> TLABStats</td><td>true</td><td>Print various TLAB related information </td><
tr>
<tr><td> TLABWasteIncrement</td><td>4</td><td>Increment allowed waste at slow alloc
tion </td></tr>
<tr><td> TLABWasteTargetPercent</td><td>1</td><td>Percentage of Eden that can be w
sted </td></tr>
<tr><td> TargetPLABWastePct</td><td>10</td><td>target wasted space in last buffer as
ct of overall allocation </td></tr>
<tr><td> TargetSurvivorRatio</td><td>50</td><td>设置年轻垃圾收集后使用的幸存者空间（
到100）的所需百分比。 </td></tr>
<tr><td> TenuredGenerationSizeIncrement</td><td>20</td><td>Adaptive size percentag
change in tenured generation </td></tr>
<tr><td> TenuredGenerationSizeSupplement</td><td>80</td><td>Supplement to Tenure
GenerationSizeIncrement used at startup </td></tr>
<tr><td> TenuredGenerationSizeSupplementDecay</td><td>2</td><td>Decay factor to T
nuredGenerationSizeIncrement </td></tr>
<tr><td> ThreadPriorityPolicy</td><td>0</td><td>0 : Normal. VM chooses priorities that
re appropriate for normal applications. On Solaris NORM_PRIORITY and above are mapped
to normal native priority. Java priorities below NORM_PRIORITY map to lower native priorit
values. On Windows applications are allowed to use higher native priorities. However, with T
readPriorityPolicy=0, VM will not use the highest possible native priority, THREAD_PRIORIT
_TIME_CRITICAL, as it may interfere with system threads. On Linux thread priorities ar
ignored because the OS does not support static priority in SCHED_OTHER scheduling class
which is the only choice for non-root, non-realtime applications. 1 : Aggressive. Java thread
priorities map over to the entire range of native thread priorities. Higher Java thread priorities
map to higher native thread priorities. This policy should be used with care, as sometimes
it can cause performance degradation in the application and/or the entire system. On Linux th
s policy requires root privilege. </td></tr>
<tr><td> ThreadPriorityVerbose</td><td>false</td><td>Print priority changes </td></tr>
<tr><td> ThreadSafetyMargin</td><td>52428800</td><td>Thread safety margin is used
n fixed-stack LinuxThreads (on Linux/x86 only) to prevent heap-stack collision. Set to 0 to di
able this feature </td></tr>
<tr><td> ThreadStackSize</td><td>1024</td><td>线程的栈大小(字节数)，也可用-Xss来配置

</td></tr>
<tr><td> ThresholdTolerance</td><td>10</td><td>Allowed collection cost difference bet
een generations </td></tr>
<tr><td> Tier0BackedgeNotifyFreqLog</td><td>10</td><td>Interpreter (tier 0) invocation
notification frequency. </td></tr>
<tr><td> Tier0InvokeNotifyFreqLog</td><td>7</td><td>Interpreter (tier 0) invocation noti
ication frequency. </td></tr>
<tr><td> Tier0ProfilingStartPercentage</td><td>200</td><td>Start profiling in interpreter
if the counters exceed tier 3 thresholds by the specified percentage </td></tr>
<tr><td> Tier23InlineeNotifyFreqLog</td><td>20</td><td>Inlinee invocation (tiers 2 and
) notification frequency </td></tr>
<tr><td> Tier2BackEdgeThreshold</td><td>0</td><td>Back edge threshold at which tier 2
compilation is invoked </td></tr>
<tr><td> Tier2BackedgeNotifyFreqLog</td><td>14</td><td>C1 without MDO (tier 2) invo
ation notification frequency. </td></tr>
<tr><td> Tier2CompileThreshold</td><td>0</td><td>threshold at which tier 2 compilatio
is invoked </td></tr>
<tr><td> Tier2InvokeNotifyFreqLog</td><td>11</td><td>C1 without MDO (tier 2) invocat
on notification frequency. </td></tr>
<tr><td> Tier3BackEdgeThreshold</td><td>60000</td><td>Back edge threshold at which
ier 3 OSR compilation is invoked </td></tr>
<tr><td> Tier3BackedgeNotifyFreqLog</td><td>13</td><td>C1 with MDO profiling (tier 3
invocation notification frequency. </td></tr>
<tr><td> Tier3CompileThreshold</td><td>2000</td><td>Threshold at which tier 3 compil
tion is invoked (invocation minimum must be satisfied. </td></tr>
<tr><td> Tier3DelayOff</td><td>2</td><td>If C2 queue size is less than this amount per
ompiler thread allow methods compiled at tier 2 transition to tier 3 </td></tr>
<tr><td> Tier3DelayOn</td><td>5</td><td>If C2 queue size grows over this amount per
ompiler thread stop compiling at tier 3 and start compiling at tier 2 </td></tr>
<tr><td> Tier3InvocationThreshold</td><td>200</td><td>Compile if number of method i
vocations crosses this threshold </td></tr>
<tr><td> Tier3InvokeNotifyFreqLog</td><td>10</td><td>C1 with MDO profiling (tier 3) in
ocation notification frequency. </td></tr>
<tr><td> Tier3LoadFeedback</td><td>5</td><td>Tier 3 thresholds will increase twofold
hen C1 queue size reaches this amount per compiler thread </td></tr>
<tr><td> Tier3MinInvocationThreshold</td><td>100</td><td>Minimum invocation to co
pile at tier 3 </td></tr>
<tr><td> Tier4BackEdgeThreshold</td><td>40000</td><td>Back edge threshold at which
ier 4 OSR compilation is invoked </td></tr>
<tr><td> Tier4CompileThreshold</td><td>15000</td><td>Threshold at which tier 4 compi
ation is invoked (invocation minimum must be satisfied. </td></tr>
<tr><td> Tier4InvocationThreshold</td><td>5000</td><td>Compile if number of method
nvocations crosses this threshold </td></tr>
<tr><td> Tier4LoadFeedback</td><td>3</td><td>Tier 4 thresholds will increase twofold
hen C2 queue size reaches this amount per compiler thread </td></tr>
<tr><td> Tier4MinInvocationThreshold</td><td>600</td><td>Minimum invocation to co
pile at tier 4 </td></tr>
<tr><td> TieredCompilation</td><td>true</td><td>启用分层编译。 </td></tr>
<tr><td> TieredCompileTaskTimeout</td><td>50</td><td>Kill compile task if method was
not used within given timeout in milliseconds </td></tr>
<tr><td> TieredRateUpdateMaxTime</td><td>25</td><td>Maximum rate sampling interv
l (in milliseconds) </td></tr>
<tr><td> TieredRateUpdateMinTime</td><td>1</td><td>Minimum rate sampling interval
in milliseconds) </td></tr>

<tr><td> TieredStopAtLevel</td><td>4</td><td>Stop at given compilation level </td></t
>
<tr><td> TimeLinearScan</td><td>false</td><td>detailed timing of LinearScan phases </
d></tr>
<tr><td> TraceBiasedLocking</td><td>false</td><td>Trace biased locking in JVM </td><
tr>
<tr><td> TraceClassLoading</td><td>false</td><td>监控类的加载 </td></tr>
<tr><td> TraceClassLoadingPreorder</td><td>false</td><td>跟踪所有加载的引用类 </td>
/tr>
<tr><td> TraceClassPaths</td><td>false</td><td> </td></tr>
<tr><td> TraceClassResolution</td><td>false</td><td>跟踪常量池的变化 </td></tr>
<tr><td> TraceClassUnloading</td><td>false</td><td>跟踪类的卸载 </td></tr>
<tr><td> TraceDynamicGCThreads</td><td>false</td><td>Trace the dynamic GC thread u
age </td></tr>
<tr><td> TraceGen0Time</td><td>false</td><td>Trace accumulated time for Gen 0 collect
on </td></tr>
<tr><td> TraceGen1Time</td><td>false</td><td>Trace accumulated time for Gen 1 collect
on </td></tr>
<tr><td> TraceJVMTI</td><td></td><td> Trace flags for JVMTI functions and events </td
</tr>
<tr><td> TraceLoaderConstraints</td><td>false</td><td>跟踪类加载器约束的相关信息 </t
></tr>
<tr><td> TraceMetadataHumongousAllocation</td><td>false</td><td> </td></tr>
<tr><td> TraceMonitorInflation</td><td>false</td><td>Trace monitor inflation in JVM </
d></tr>
<tr><td> TraceParallelOldGCTasks</td><td>false</td><td>Trace multithreaded GC activity
</td></tr>
<tr><td> TraceRedefineClasses</td><td>0</td><td>Trace level for JVMTI RedefineClasses
</td></tr>
<tr><td> TraceSafepointCleanupTime</td><td>false</td><td>print the break down of cle
n up tasks performed during safepoint </td></tr>
<tr><td> TraceSharedLookupCache</td><td>false</td><td> </td></tr>
<tr><td> TraceSuspendWaitFailures</td><td>false</td><td>Trace external suspend wait fa
lures </td></tr>
<tr><td> TrackedInitializationLimit</td><td>50</td><td>When initializing fields, track up
o this many words </td></tr>
<tr><td> TransmitErrorReport</td><td>false</td><td>Enable error report transmission on
erroneous termination </td></tr>
<tr><td> TrapBasedNullChecks</td><td>false</td><td> </td></tr>
<tr><td> TrapBasedRangeChecks</td><td>false</td><td> </td></tr>
<tr><td> TypeProfileArgsLimit</td><td>2</td><td> </td></tr>
<tr><td> TypeProfileLevel</td><td>111</td><td> </td></tr>
<tr><td> TypeProfileMajorReceiverPercent</td><td>90</td><td>% of major receiver type
o all profiled receivers </td></tr>
<tr><td> TypeProfileParmsLimit</td><td>2</td><td> </td></tr>
<tr><td> TypeProfileWidth</td><td>2</td><td>number of receiver types to record in call/
ast profile </td></tr>
<tr><td> UnguardOnExecutionViolation</td><td>0</td><td>Unguard page and retry on
o-execute fault (Win32 only) 0=off, 1=conservative, 2=aggressive </td></tr>
<tr><td> UnlinkSymbolsALot</td><td>false</td><td>unlink unreferenced symbols from t
e symbol table at safepoints </td></tr>
<tr><td> Use486InstrsOnly</td><td>false</td><td>Use 80486 Compliant instruction subs
t </td></tr>
<tr><td> UseAES</td><td>false</td><td>为Intel，AMD和SPARC硬件启用基于硬件的AES内

函数。 </td></tr>
<tr><td> UseAESIntrinsics</td><td>false</td><td>同上一起使用。 </td></tr>
<tr><td> UseAVX</td><td>99</td><td>Highest supported AVX instructions set on x86/x6
</td></tr>
<tr><td> UseAdaptiveGCBoundary</td><td>false</td><td>Allow young-old boundary to
move </td></tr>
<tr><td> UseAdaptiveGenerationSizePolicyAtMajorCollection</td><td>true</td><td>Use
daptive young-old sizing policies at major collections </td></tr>
<tr><td> UseAdaptiveGenerationSizePolicyAtMinorCollection</td><td>true</td><td>Use
daptive young-old sizing policies at minor collections </td></tr>
<tr><td> UseAdaptiveNUMAChunkSizing</td><td>true</td><td>Enable adaptive chunk si
ing for NUMA </td></tr>
<tr><td> UseAdaptiveSizeDecayMajorGCCost</td><td>true</td><td>Adaptive size decays
the major cost for long major intervals </td></tr>
<tr><td> UseAdaptiveSizePolicy</td><td>true</td><td>使用自适应自动调整空间大小，即J
M会自动调整年轻代与老年代的比例、Eden Space与Suvivor Space比例来达到性能目标，GC处理
先级是MaxGCPauseMillis最高，GCTimeRatio次之，其它的空间大小配置优先级最低 </td></tr>
<tr><td> UseAdaptiveSizePolicyFootprintGoal</td><td>true</td><td>Use adaptive minim
m footprint as a goal </td></tr>
<tr><td> UseAdaptiveSizePolicyWithSystemGC</td><td>false</td><td>Use statistics from
System.GC for adaptive size policy </td></tr>
<tr><td> UseAddressNop</td><td>false</td><td>Use '0F 1F [addr]' NOP instructions on x
6 cpus </td></tr>
<tr><td> UseAltSigs</td><td>false</td><td>为了防止与其他发送信号的应用程序冲突，允许
用候补信号替代 SIGUSR1和SIGUSR2 </td></tr>
<tr><td> UseAutoGCSelectPolicy</td><td>false</td><td>Use automatic collection selecti
n policy </td></tr>
<tr><td> UseBMI1Instructions</td><td>false</td><td> </td></tr>
<tr><td> UseBMI2Instructions</td><td>false</td><td> </td></tr>
<tr><td> UseBiasedLocking</td><td>true</td><td>启用偏向锁 </td></tr>
<tr><td> UseBimorphicInlining</td><td>true</td><td>Profiling based inlining for two rece
vers </td></tr>
<tr><td> UseBoundThreads</td><td>true</td><td>绑定所有的用户线程到内核线程, 减少线
进入饥饿状态（得不到任何cpu time）的次数。 </td></tr>
<tr><td> UseBsdPosixThreadCPUClocks</td><td>true</td><td> </td></tr>
<tr><td> UseCLMUL</td><td>false</td><td> </td></tr>
<tr><td> UseCMSBestFit</td><td>true</td><td>Use CMS best fit allocation strategy </td
</tr>
<tr><td> UseCMSCollectionPassing</td><td>true</td><td>Use passing of collection from
background to foreground </td></tr>
<tr><td> UseCMSCompactAtFullCollection</td><td>true</td><td>CMS在FullGC前进行老
代的内存碎片整理 </td></tr>
<tr><td> UseCMSInitiatingOccupancyOnly</td><td>false</td><td>CMS会根据历史记录，
测老年代还需要多久填满及进行一次回收所需要的时间。在老年代空间用完之前，CMS可以根据自己
预测自动执行垃圾回收。若设置该值为true，则关闭该预测。 </td></tr>
<tr><td> UseCRC32Intrinsics</td><td>false</td><td> </td></tr>
<tr><td> UseCodeCacheFlushing</td><td>true</td><td>是否在code cache满的时候先尝试
理一下，如果还是不够用再关闭编译。 </td></tr>
<tr><td> UseCompiler</td><td>true</td><td>use compilation </td></tr>
<tr><td> UseCompilerSafepoints</td><td>true</td><td>Stop at safepoints in compiled c
de </td></tr>
<tr><td> UseCompressedClassPointers</td><td>false</td><td>启用压缩类指针（对象中指
类元数据的指针会被压缩成32位） </td></tr>
<tr><td> UseCompressedOops</td><td>false</td><td>启用压缩对象指针（堆中对象指针会

压缩成32位） </td></tr>
<tr><td> UseConcMarkSweepGC</td><td>false</td><td>开启CMS收集器收集老年代，新代默认使用并行收集器。 </td></tr>
<tr><td> UseCondCardMark</td><td>false</td><td>在更新卡表之前，可以检查卡是否已经记。 </td></tr>
<tr><td> UseCountLeadingZerosInstruction</td><td>false</td><td>Use count leading zers instruction </td></tr>
<tr><td> UseCountTrailingZerosInstruction</td><td>false</td><td> </td></tr>
<tr><td> UseCountedLoopSafepoints</td><td>false</td><td>强制在Counted loop循环回之前插入Safepoint，也就是说即使循环比较短，JVM也会帮忙插入Safepoint了，用于防止大循环执时间过长导致进入Safepoint卡住的问题。但是这个参数在JDK8上是有Bug的，可能会导致JVM Cras，而且是到JDK9才修复的，具体参考JDK-8161147。 </td></tr>
<tr><td> UseCounterDecay</td><td>true</td><td>开启方法调用计数器的热度衰减。 </td>/tr>
<tr><td> UseDivMod</td><td>true</td><td>Use combined DivMod instruction if availabl </td></tr>
<tr><td> UseDynamicNumberOfGCThreads</td><td>false</td><td>Dynamically choose te number of parallel threads parallel gc will use </td></tr>
<tr><td> UseFPUForSpilling</td><td>false</td><td>Spill integer registers to FPU instead f stack when possible </td></tr>
<tr><td> UseFastAccessorMethods</td><td>true</td><td>优化原始类型的getter方法性能 <td></tr>
<tr><td> UseFastEmptyMethods</td><td>true</td><td>Use fast method entry code for mpty methods </td></tr>
<tr><td> UseFastJNIAccessors</td><td>true</td><td>Use optimized versions of Get<Primtive>Field </td></tr>
<tr><td> UseFastStosb</td><td>false</td><td>Use fast-string operation for zeroing: rep sosb </td></tr>
<tr><td> UseG1GC</td><td>false</td><td>允许使用垃圾优先（G1）垃圾收集器。它是一个务器式垃圾收集器，针对具有大量RAM的多处理器机器。它以高概率满足GC暂停时间目标，同时保良好的吞吐量。G1收集器推荐用于需要大堆（大小约为6 GB或更大）且GC延迟要求有限的应用（稳且可预测的暂停时间低于0.5秒）。 </td></tr>
<tr><td> UseGCLogFileRotation</td><td>false</td><td>Prevent large gclog file for long rnning app. Requires -Xloggc:<filename> </td></tr>
<tr><td> UseGCOverheadLimit</td><td>true</td><td>允许使用策略来限制在OutOfMemorError引发异常之前JVM在GC上花费的时间比例。 </td></tr>
<tr><td> UseGCTaskAffinity</td><td>false</td><td>保证每次工作线程被唤醒的时候争取拿上次没有完成的工作。 </td></tr>
<tr><td> UseHeavyMonitors</td><td>false</td><td>use heavyweight instead of lightweiht Java monitors </td></tr>
<tr><td> UseHugeTLBFS</td><td>false</td><td> </td></tr>
<tr><td> UseInlineCaches</td><td>true</td><td>Use Inline Caches for virtual calls </td>/tr>
<tr><td> UseInterpreter</td><td>true</td><td>Use interpreter for non-compiled method </td></tr>
<tr><td> UseJumpTables</td><td>true</td><td>Use JumpTables instead of a binary sear h tree for switches </td></tr>
<tr><td> UseLWPSynchronization</td><td>true</td><td>使用轻量级进程（内核线程）替线程同步。 </td></tr>
<tr><td> UseLargePages</td><td>false</td><td>启用大内存页支持 </td></tr>
<tr><td> UseLargePagesInMetaspace</td><td>false</td><td>在metaspace启用大内存页依赖于UseLargePages参数 </td></tr>
<tr><td> UseLargePagesIndividualAllocation</td><td>false</td><td>Allocate large pages ndividually for better affinity </td></tr>

<tr><td> UseLockedTracing</td><td>false</td><td>Use locked-tracing when doing event based tracing </td></tr>
<tr><td> UseLoopCounter</td><td>true</td><td>Increment invocation counter on back ard branch </td></tr>
<tr><td> UseLoopInvariantCodeMotion</td><td>true</td><td> </td></tr>
<tr><td> UseLoopPredicate</td><td>true</td><td>Generate a predicate to select fast/sl w loop versions </td></tr>
<tr><td> UseMathExactIntrinsics</td><td>true</td><td> </td></tr>
<tr><td> UseMaximumCompactionOnSystemGC</td><td>true</td><td>In the Parallel Old garbage collector maximum compaction for a system GC </td></tr>
<tr><td> UseMembar</td><td>true</td><td>(Unstable) Issues membars on thread state t ansitions </td></tr>
<tr><td> UseMontgomeryMultiplyIntrinsic</td><td>false</td><td> </td></tr>
<tr><td> UseMontgomerySquareIntrinsic</td><td>false</td><td> </td></tr>
<tr><td> UseMulAddIntrinsic</td><td>false</td><td> </td></tr>
<tr><td> UseMultiplyToLenIntrinsic</td><td>false</td><td> </td></tr>
<tr><td> UseNUMA</td><td>false</td><td>通过增加应用程序对低延迟内存的使用，在具有均匀内存架构（NUMA）的计算机上实现应用程序的性能优化。 </td></tr>
<tr><td> UseNUMAInterleaving</td><td>false</td><td>Interleave memory across NUMA nodes if available </td></tr>
<tr><td> UseNewLongLShift</td><td>false</td><td>Use optimized bitwise shift left </td </tr>
<tr><td> UseOSErrorReporting</td><td>false</td><td>Let VM fatal error propagate to th OS (ie. WER on Windows) </td></tr>
<tr><td> UseOldInlining</td><td>true</td><td>Enable the 1.3 inlining strategy </td></t >
<tr><td> UseOnStackReplacement</td><td>true</td><td>Use on stack replacement, calls untime if invoc. counter overflows in loop </td></tr>
<tr><td> UseOnlyInlinedBimorphic</td><td>true</td><td>Don't use BimorphicInlining if an't inline a second method </td></tr>
<tr><td> UseOprofile</td><td>false</td><td> </td></tr>
<tr><td> UseOptoBiasInlining</td><td>true</td><td>Generate biased locking code in C2 deal graph </td></tr>
<tr><td> UsePSAdaptiveSurvivorSizePolicy</td><td>true</td><td>Use adaptive survivor s zing policies </td></tr>
<tr><td> UseParNewGC</td><td>false</td><td>针对年轻代使用多线程垃圾收集器(ParNew)可与CMS同时使用。在serial基础上实现的多线程收集器。 </td></tr>
<tr><td> UseParallelGC</td><td>false</td><td>针对年轻代使用并行垃圾收集器(Parallel Sca enge)，可以并行多个垃圾收集线程，但此时用户线程必须停止。不能与CMS一起使用，系统吞吐量用优化，会有较长时间的应用暂停，后台系统任务可以使用该GC。 </td></tr>
<tr><td> UseParallelOldGC</td><td>false</td><td>允许将并行垃圾收集器用于完整的GC。 < td></tr>
<tr><td> UsePerfData</td><td>true</td><td>Flag to disable jvmstat instrumentation for erformance testing and problem isolation purposes. </td></tr>
<tr><td> UsePopCountInstruction</td><td>false</td><td>Use population count instructi n </td></tr>
<tr><td> UseRDPCForConstantTableBase</td><td>false</td><td>Use Sparc RDPC instruct on for the constant table base. </td></tr>
<tr><td> UseRTMDeopt</td><td>false</td><td>根据中止率自动调谐RTM锁定。该比率由-XX RTMAbortRatio选项指定。如果中止事务的数量超过中止率，则包含锁定的方法将被取消优化并重编译，并将所有锁定为正常锁定。必须启用UseRTMLockingf才生效。 </td></tr>
<tr><td> UseRTMLocking</td><td>false</td><td>为所有膨胀的锁生成受限制的事务性内存RTM）锁定代码，使用正常的锁定机制作为回退处理程序。RTM是英特尔TSX的一部分，它是x86指集扩展，有助于创建多线程应用程序。RTM引入了新的指示XBEGIN，XABORT，XEND，和XTEST。

</td></tr>
<tr><td> UseSHA</td><td>false</td><td>为SPARC硬件启用SHA加密散列函数的基于硬件的在函数。 </td></tr>
<tr><td> UseSHA1Intrinsics</td><td>false</td><td>为SHA-1加密哈希函数启用内在函数。 <td></tr>
<tr><td> UseSHA256Intrinsics</td><td>false</td><td>为SHA-224和SHA-256加密哈希函数用内在函数。 </td></tr>
<tr><td> UseSHA512Intrinsics</td><td>false</td><td>为SHA-384和SHA-512加密散列函数用内在函数。 </td></tr>
<tr><td> UseSHM</td><td>false</td><td>使JVM能够使用共享内存来设置大页面。 </td></t>
<tr><td> UseSSE</td><td>99</td><td> </td></tr>
<tr><td> UseSSE42Intrinsics</td><td>false</td><td> </td></tr>
<tr><td> UseSerialGC</td><td>false</td><td>允许使用串行垃圾收集器。对于不需要垃圾收的任何特殊功能的小型和简单应用程序，这通常是最佳选择。 </td></tr>
<tr><td> UseSharedSpaces</td><td>true</td><td>Use shared spaces in the permanent g neration </td></tr>
<tr><td> UseSignalChaining</td><td>true</td><td>Use signal-chaining to invoke signal andlers installed by the application (Solaris & Linux only) </td></tr>
<tr><td> UseSquareToLenIntrinsic</td><td>false</td><td> </td></tr>
<tr><td> UseStoreImmI16</td><td>true</td><td>Use store immediate 16-bits value instr ction on x86 </td></tr>
<tr><td> UseStringDeduplication</td><td>false</td><td>启用字符串重复数据删除。要使用选项，必须启用垃圾优先（G1）垃圾收集器。字符串重复数据删除String通过利用许多String对象相的事实来减少Java堆上对象的内存占用。String相同的String对象可以指向并共享相同的字符数组，不是每个对象指向其自己的字符数组。 </td></tr>
<tr><td> UseSuperWord</td><td>true</td><td>允许将标量操作转换为超级字操作。UseSup rWord会在字节运算时启用布隆过滤器之类的特性，能在haswell架构的cpu上提高2倍以上的速度。 /td></tr>
<tr><td> UseTLAB</td><td>true</td><td>允许在年轻代空间中使用线程局部分配块（TLAB 。 </td></tr>
<tr><td> UseThreadPriorities</td><td>true</td><td>使用本地线程的优先级。 </td></tr>
<tr><td> UseTypeProfile</td><td>true</td><td>Check interpreter profile for historically onomorphic calls </td></tr>
<tr><td> UseTypeSpeculation</td><td>true</td><td> </td></tr>
<tr><td> UseUnalignedLoadStores</td><td>false</td><td>Use SSE2 MOVDQU instruction for Arraycopy </td></tr>
<tr><td> UseVMInterruptibleIO</td><td>false</td><td>在solaris中，允许运行时中断线程。 </td></tr>
<tr><td> UseXMMForArrayCopy</td><td>false</td><td>Use SSE2 MOVQ instruction for A raycopy </td></tr>
<tr><td> UseXmmI2D</td><td>false</td><td>Use SSE2 CVTDQ2PD instruction to convert nteger to Double </td></tr>
<tr><td> UseXmmI2F</td><td>false</td><td>Use SSE2 CVTDQ2PS instruction to convert nteger to Float </td></tr>
<tr><td> UseXmmLoadAndClearUpper</td><td>true</td><td>Load low part of XMM regi ter and clear upper part </td></tr>
<tr><td> UseXmmRegToRegMoveAll</td><td>false</td><td>Copy all XMM register bits hen moving value between registers </td></tr>
<tr><td> VMThreadHintNoPreempt</td><td>false</td><td>(Solaris only) Give VM thread n extra quanta </td></tr>
<tr><td> VMThreadPriority</td><td>-1</td><td>The native priority at which the VM thre d should run (-1 means no change) </td></tr>
<tr><td> VMThreadStackSize</td><td>1024</td><td>Non-Java Thread Stack Size (in Kbyt

s) </td></tr>
<tr><td> ValueMapInitialSize</td><td>11</td><td>Initial size of a value map </td></tr>
<tr><td> ValueMapMaxLoopSize</td><td>8</td><td>maximum size of a loop optimized
y global value numbering </td></tr>
<tr><td> ValueSearchLimit</td><td>1000</td><td>Recursion limit in PhaseMacroExpand::
alue_from_mem_phi </td></tr>
<tr><td> VerifyMergedCPBytecodes</td><td>true</td><td>Verify bytecodes after Redefi
eClasses constant pool merging </td></tr>
<tr><td> VerifySharedSpaces</td><td>false</td><td> </td></tr>
<tr><td> WorkAroundNPTLTimedWaitHang</td><td>1</td><td>(Unstable, Linux-specific)
 avoid NPTL-FUTEX hang pthread_cond_timedwait </td></tr>
<tr><td> YoungGenerationSizeIncrement</td><td>20</td><td>Adaptive size percentage
hange in young generation </td></tr>
<tr><td> YoungGenerationSizeSupplement</td><td>80</td><td>Supplement to Younge
GenerationSizeIncrement used at startup </td></tr>
<tr><td> YoungGenerationSizeSupplementDecay</td><td>8</td><td>Decay factor to Yo
ngedGenerationSizeSupplement </td></tr>
<tr><td> YoungPLABSize</td><td>4096</td><td>Size of young gen promotion labs (in H
apWords) </td></tr>
<tr><td> ZeroTLAB</td><td>false</td><td>Zero out the newly created TLAB </td></tr>
<tr><td> hashCode</td><td>5</td><td>(Unstable) select hashCode generation algorithm
</td></tr></table>

查看JVM自动设置的XX配置

查看JVM自动设置的XX配置的命令：java -XX:+PrintCommandLineFlags,输出如下：

-XX:InitialHeapSize=268435456 -XX:MaxHeapSize=4294967296 -XX:+PrintCommandLineFlags -XX:+UseCompressedClassPointers -XX:+UseCompressedOops -XX:+UseParallelGC

参考资料

1. https://www.cnblogs.com/z-sm/p/6253335.html
2. https://docs.oracle.com/javase/8/docs/technotes/guides/vm/server-class.html
3. https://www.iteye.com/blog/qingfeng825-1781617
4. https://blog.csdn.net/lxlmycsdnfree/article/details/81531363
5. https://blog.csdn.net/huanxianglove/article/details/90247994
6. https://blog.csdn.net/foolishandstupid/article/details/77596050
7. http://lovestblog.cn/blog/2015/05/07/system-gc/
8. http://ifeve.com/jvm-cms-log/
9. https://blog.csdn.net/lz710117239/article/details/78565926
10. https://blog.csdn.net/shine0181/article/details/8533834
11. https://www.cnblogs.com/moonandstar08/p/4924588.html
12. https://blog.csdn.net/zero__007/article/details/52926366
13. https://www.jianshu.com/p/a6f19189ec62
14. https://www.jianshu.com/p/b99c9ef0eb16

15. https://www.cnblogs.com/milton/p/6134251.html
16. https://www.jianshu.com/p/92a5fbb33764
17. https://emacsist.github.io/2019/07/31/java%E5%86%85%E8%81%94inline%E7%9B%B8%E%85%B3%E8%B5%84%E6%96%99/
18. https://www.jianshu.com/p/7414fd6862c5
19. https://blog.csdn.net/blueheart20/article/details/52050545
20. http://www.voidcn.com/article/p-kowussju-brm.html
21. https://www.iteye.com/blog/softbeta-1467379
22. https://blog.csdn.net/jadar_ly/article/details/670751
23. http://www.seotest.cn/jishu/27902.html
24. https://www.jianshu.com/p/d3c3301740d0
25. https://blog.csdn.net/pengzhouzhou/article/details/94516616
26. https://blog.csdn.net/weixin_34255793/article/details/91666030
27. https://www.cnblogs.com/syuf/p/10183683.html
28. https://www.bbsmax.com/A/rV574r1XdP/
29. https://my.oschina.net/Rayn/blog/842868
30. https://www.jianshu.com/p/00a7a023adf3
31. https://www.cnblogs.com/yingsong/p/5896207.html
32. https://www.jianshu.com/p/f65eebe38382
33. https://www.jianshu.com/p/5fbe42932f81
34. http://www.voycn.com/index.php/article/mianshitishenrujiexisynchronizeddicengshixian
35. https://blog.csdn.net/chenmh12/article/details/90256808
36. https://toutiao.io/posts/hltb1e/preview
37. https://blog.csdn.net/www_changer/article/details/82285842
38. https://www.520mwx.com/view/61784
39. https://www.jianshu.com/p/45b125e5ec61
40. https://www.codercto.com/a/85400.html
41. https://ifeve.com/useful-jvm-flags-part-7-cms-collector/
42. http://www.chepoo.com/application-pressure-promotion-failed.html
43. https://juejin.im/entry/5c382ce9e51d45515f24521c
44. https://www.jianshu.com/p/2a1b2f17d3e4
45. https://www.jianshu.com/p/be5389ca93f7
46. https://www.jianshu.com/p/12eda04e6f87
47. https://blog.csdn.net/foolishandstupid/article/details/77430875
48. https://www.jianshu.com/p/832fc4d4cb53
49. https://blog.csdn.net/yxc135/article/details/12068177
50. https://blog.csdn.net/weixin_39267363/article/details/100415791

51. https://blog.csdn.net/sky1young/article/details/27063821
52. https://www.cnblogs.com/beichenroot/p/11134409.html
53. https://www.jianshu.com/p/0964124ae822
54. https://stackoverflow.com/questions/5792049/xxonoutofmemoryerror-kill-9-p-problem
55. https://blog.csdn.net/maosijunzi/article/details/46410697
56. https://www.jianshu.com/p/0b8a9d137ee7
57. https://cloud.tencent.com/developer/article/1486792
58. https://www.cnblogs.com/BlueStarWei/p/9358757.html
59. https://www.jianshu.com/p/5638c378a910
60. https://blog.csdn.net/qq_26222859/article/details/80546917
61. http://ifeve.com/jvm%E4%BC%98%E5%8C%96%E4%B9%8B%E9%80%83%E9%80%B8%E%88%86%E6%9E%90%E5%8F%8A%E9%94%81%E6%B6%88%E9%99%A4/
62. https://www.jianshu.com/p/871b993fb3ad
63. http://www.itkeyword.com/doc/5667377716363278222/Windows-XP-TomcatJVMJava
64. https://blog.csdn.net/qq_27529917/article/details/86978237
65. https://segmentfault.com/a/1190000007815623
66. https://www.jiankunking.com/java-jvm-gc-g1-note.html
67. https://mp.weixin.qq.com/s/FHY0MelBfmgdRpT4zWF9dQ
68. https://www.cnblogs.com/ityouknow/p/5614961.html
69. https://blog.csdn.net/qq_34446485/article/details/80968685
70. https://www.2cto.com/kf/201806/757352.html
71. https://www.jianshu.com/p/29c20f0684d0
72. https://www.cnblogs.com/iceAeterNa/p/4878112.html
73. https://www.zhihu.com/question/57212072
74. http://www.hackerav.com/?post=43633
75. https://blog.homurax.com/2018/09/20/heap-space/
76. https://stackoverflow.com/questions/32503669/why-does-the-jvm-have-a-maximum-inline-depth
77. https://blog.csdn.net/xhh198781/article/details/41622925
78. https://www.jianshu.com/p/c46f6fff17cb
79. https://zhuanlan.zhihu.com/p/28505703
80. https://my.oschina.net/Rayn/blog/1510535
81. https://blog.csdn.net/bdx_hadoop_opt/article/details/34101365
82. https://my.oschina.net/foxty/blog/1934968
83. https://blog.csdn.net/yanguz/article/details/84536440