### HotSpot JVM garbage collection options cheat sheet (v2)

# HotSpot GC collectors

HotSpot JVM may use one of 6 combinations of garbage collection algorithms listed below.

|  |  |  |
| --- | --- | --- |
| **Young collector** | **Old collector** | **JVM option** |
| Serial (DefNew) | Serial Mark-Sweep-Compact | -XX:+UseSerialGC |
| Parallel scavenge (PSYoungGen) | Serial Mark-Sweep-Compact (PSOldGen) | -XX:+UseParallelGC |
| Parallel scavenge (PSYoungGen) | Parallel Mark-Sweep-Compact (ParOldGen) | -XX:+UseParallelOldGC |
| Serial (DefNew) | Concurrent Mark Sweep | -XX:+UseConcMarkSweepGC  -XX:-UseParNewGC |
| Parallel (ParNew) | Concurrent Mark Sweep | -XX:+UseConcMarkSweepGC  -XX:+UseParNewGC |
| G1 | | -XX:+UseG1GC |

# GC logging options

Please note that many of logging options could be modified on running JVM using JMX (e.g. via JConsole).

|  |  |
| --- | --- |
| **JVM option** | **Description** |
| General options | |
| -verbose:gc or -XX:+PrintGC | Print basic GC info |
| -XX:+PrintGCDetails | Print more elaborated GC info |
| -XX:+PrintGCTimeStamps | Print timestamps for each GC event (seconds count from start of JVM) |
| -XX:+PrintGCDateStamps | Print date stamps at garbage collection events (e.g. 2011-09-08T14:20:29.557+0400: [GC... ) |
| -XX:+PrintGC\  TaskTimeStamps | Print timestamps for individual GC worker thread tasks (very verbose) |
| -Xloggc: | Redirects GC output to a file instead of console |
| -XX:+Print\  TenuringDistribution | Print detailed demography of young space after each collection |
| -XX:+PrintTLAB | Print TLAB allocation statistics |
| -XX:+PrintReferenceGC | Print times for weak/soft/JNI/etc reference processing during STW pause |
| -XX:+PrintJNIGCStalls | Reports if GC is waiting for native code to unpin object in memory |
| -XX:+PrintGC\  ApplicationStoppedTime | Print pause summary after each stop-the-world pause |
| -XX:+PrintGC\  ApplicationConcurrentTime | Print time for each concurrent phase of GC |
| -XX:+Print\  ClassHistogramAfterFullGC | Prints class histogram after full GC |
| -XX:+Print\  ClassHistogramBeforeFullGC | Prints class histogram before full GC |
| -XX:+HeapDump\  AfterFullGC | Creates heap dump file after full GC |
| -XX:+HeapDump\  BeforeFullGC | Creates heap dump file before full GC |
| -XX:+HeapDump\  OnOutOfMemoryError | Creates heap dump in out-of-memory condition |
| -XX:HeapDumpPath=<path> | Specifies path to save heap dumps |
| CMS specific options | |
| -XX:PrintCMSStatistics=2 | Print additional CMS statistics if n >= 1 |
| -XX:+Print\  CMSInitiationStatistics | Print CMS initiation details |
| -XX:PrintFLSStatistics=2 | Print additional info concerning free lists |
| -XX:PrintFLSCensus=2 | Print additional info concerning free lists |
| -XX:+PrintPromotionFailure | Print additional diagnostic information following promotion failure |
| -XX:+CMSDumpAt\  PromotionFailure | Dump useful information about the state of the CMS old generation upon a promotion failure. |
| -XX:+CMSPrint\  ChunksInDump | In a CMS dump enabled by option above, include more detailed information about the free chunks. |
| -XX:+CMSPrint\  ObjectsInDump | In a CMS dump enabled by option above, include more detailed information about the allocated objects. |

# JVM sizing options

|  |  |
| --- | --- |
| **JVM option** | **Description** |
| -Xms2048m -Xmx2g or  ‑XX:InitialHeapSize=2g  ‑XX:MaxHeapSize=2g | Initial and max size of heap space (young space + tenured space). Permanent space does not count to this size. |
| -XX:NewSize=64m  -XX:MaxNewSize=64m | Initial and max size of young space. |
| -XX:NewRatio=3 | Alternative way to specify young space size. Sets ration of young vs tenured space (e.g. -XX:NewRatio=2 means that young space will be 2 time smaller than tenuted space). |
| -XX:SurvivorRatio=15 | Sets size of single survivor space as a portion of Eden space size (e.g. -XX:NewSize=64m -XX:SurvivorRatio=6 means that each survivor space will be 8m and eden will be 48m). |
| -XX:PermSize=256m  -XX:MaxPermSize=256m | Initial and max size of permanent space. |
| -Xss256k (size in bytes) or  -XX:ThreadStackSize=256 (size in Kbytes) | Sets size of stack area dedicated to each thread. Thread stacks do not count to heap size. |
| -XX:MaxDirectMemorySize=2g | Maximum size of off-heap memory available for JVM |

# Young collection tuning

|  |  |
| --- | --- |
| **JVM option** | **Description** |
| -XX:Initial\  TenuringThreshold=8 | Initial value for tenuring threshold (number of collections before object will be promoted to tenured space). |
| -XX:Max\  TenuringThreshold=15 | Max value for tenuring threshold. |
| -XX:Pretenure\  SizeThreshold=2m | Max object size allowed to be allocated in young space (large objects will be allocated directly in old space). Thread local allocation bypasses this check so if TLAB is large enough object exciding size threshold still may be allocated in young. |
| -XX:+AlwaysTenure | Promote all objects surviving young collection immediately to tenured space (equivalent of -XX:MaxTenuringThreshold=0) |
| -XX:+NeverTenure | Objects from young space will never get promoted to tenured space while survivor space is large enough to keep them. |
| Thread local allocation blocks | |
| -XX:+UseTLAB | Use thread local allocation blocks in young space. Enabled by default. |
| -XX:+ResizeTLAB | Allow JVM to adaptively resize TLAB for threads. |
| -XX:TLABSize=1m | Initial size of TLAB for thread |
| -XX:MinTLABSize=64k | Minimal allowed size of TLAB |

# CMS tuning options

|  |  |
| --- | --- |
| **JVM option** | **Description** |
| Controlling initial mark phase | |
| -XX:+UseCMSInitiating\  OccupancyOnly | Only use occupancy as a criterion for starting a CMS collection. |
| -XX:CMSInitiating\  OccupancyFraction=70 | Percentage CMS generation occupancy to start a CMS collection cycle. A negative value means that CMSTriggerRatio is used. |
| -XX:CMSBootstrap\  Occupancy=10 | Percentage CMS generation occupancy at which to initiate CMS collection for bootstrapping collection stats. |
| -XX:CMSTriggerRatio=70 | Percentage of MinHeapFreeRatio in CMS generation that is allocated before a CMS collection cycle commences. |
| -XX:CMSTriggerPermRatio=90 | Percentage of MinHeapFreeRatio in the CMS perm generation that is allocated before a CMS collection cycle commences, that also collects the perm generation. |
| -XX:CMSWaitDuration=2000 | Once CMS collection is triggered, it will wait for next young collection to perform initial mark right after. This parameter specifies how long CMS can wait for young collection. |
| Controlling remark phase | |
| -XX:+CMSParallel\  RemarkEnabled | Whether parallel remark enabled (only if ParNewGC), enabled by default |
| -XX:+CMSParallel\  SurvivorRemarkEnabled | Whether parallel remark of survivor space enabled (effective only with option above), enabled by default |
| -XX:+CMSScavengeBeforeRemark | Force young collection before remark phase. |
| -XX:+CMSScheduleRemark\  EdenSizeThreshold | If Eden used is below this value, don't try to schedule remark |
| -XX:CMSScheduleRemark\  EdenPenetration=20 | The Eden occupancy % at which to try and schedule remark pause |
| -XX:CMSScheduleRemark\  SamplingRatio=4 | Start sampling Eden top at least before young generation occupancy reaches 1/ of the size at which we plan to schedule remark |
| -XX:CMSRemarkVerifyVariant=1 | Choose variant (1,2) of verification following remark |
| Parallel execution | |
| -XX:+UseParNewGC | Use parallel algorithm for young space collection. |
| -XX:+CMSConcurrentMTEnabled | Use multiple threads for concurrent phases. |
| -XX:ConcGCThreads=2 | Number of parallel threads used for concurrent phase. |
| -XX:ParallelGCThreads=2 | Number of parallel threads used for stop-the-world phases. |
| CMS incremental mode | |
| -XX:+CMSIncrementalMode | Enable incremental CMS mode. Incremental mode is meant for severs with small number of CPU. Do not use it on modern hardware. |
| Miscellaneous options | |
| -XX:+CMSClassUnloadingEnabled | If not enabled, CMS will not clean permanent space. You should always enable it in multiple class loader environments such as JEE or OSGi. |
| -XX:+ExplicitGCInvokes\  Concurrent | Let System.gc() trigger concurrent collection instead of full GC. |
| ‑XX:+ExplicitGCInvokes\  ConcurrentAndUnloadsClasses | Same as above but also triggers permanent space collection. |

# Miscellaneous GC options

|  |  |
| --- | --- |
| **JVM option** | **Description** |
| -XX:+DisableExplicitGC | JVM will ignore application calls to System.gc() |
| -XX:SoftRefLRU\  PolicyMSPerMB=1000 | Soft references time to live in milliseconds per MB of free space in the heap |
| OutOfMemoryError conditions | |
| -XX:+UseGCOverheadLimit | Use policy to limit of proportion of time spent in GC before an OutOfMemory error is thrown |
| -XX:GCTimeLimit=98 | Limit of proportion of time spent in GC before an OutOfMemory error is thrown (used with GCHeapFreeLimit) |
| -XX:GCHeapFreeLimit=2 | Minimum percentage of free space after a full GC before an OutOfMemoryError is thrown (used with GCTimeLimit) |

# G1 (garbage first) GC options

G1 is still evolving, so table below have known supported JVM version column

|  |  |  |
| --- | --- | --- |
| **JVM option** | **JVM** | **Description** |
| -XX:G1HeapRegionSize=8m | 6u26 | Size of heap region for G1 collector. Region size could be from 1MiB to 32MiB. Default is initial heap size / 2048. |
| -XX:G1ConfidencePercent=50 | 6u26 | Confidence coefficient for G1 pause prediction heuristics |
| -XX:G1ReservePercent=10 | 6u26 | It determines the minimum reserve we should have in the heap to minimize the probability of promotion failure |
| -XX:MaxGCPauseMillis=200 | 6u26 | GC time per MMU time slice in milliseconds (for G1 only) |
| -XX:GCPauseIntervalMillis=300 | 6u26 | Time slice for MMU specification in milliseconds (for G1 only). Should be greater than parameter above. |

1. <http://blog.ragozin.info/2011/09/hotspot-jvm-garbage-collection-options.html>