Java 6 JVM參數選項大全（中文版）

作者：Ken Wu
Email: ken.wug@gmail.com
轉載本文檔請註明原文鏈接 http://kenwublog.com/docs/java6-jvm-options-chinese-edition.htm！

本文是基於最新的SUN官方文檔Java SE 6 Hotspot VM Options 編寫的譯文。主要介紹JVM中的非穩態選項及其使用說明。
為了讓讀者明白每個選項的含義，作者在原文基礎上**補充了大量的資料**。希望這份文檔，對正在研究JVM參數的朋友有幫助！

另外，考慮到本文檔是初稿，如有描述錯誤，敬請指正。

非穩態選項使用說明
-XX:+<option> 啟用選項
-XX:-<option> 不啟用選項
-XX:<option>=<number> 給選項設置一個數字類型值，可跟單位，例如 32k, 1024m, 2g
-XX:<option>=<string> 給選項設置一個字符串值，例如-XX:HeapDumpPath=./dump.core

行為選項

選項	默認值與限制	描述
-XX:-AllowUserSignalHandlers	限於Linux和Solaris，默認不啟用	允許為java進程安裝信號處理器。 Java信號處理相關知識，詳見 http://kenwublog.com/java-asynchronous-notify-based-on-signal
-XX:-DisableExplicitGC	默認不啟用	禁止在運行期顯式地調用 System.gc()。 開啟該選項後，GC的觸發時機將由Garbage Collector全權掌控。 注意：你熟悉的代碼裡沒調用System.gc()，不代表你依賴的框架工具沒在使用。 例如RMI就在多數用戶毫不知情的情況下，顯示地調用GC來防止自身OOM。請仔細權衡禁用帶來的影響。
-XX:-RelaxAccessControlCheck	默認不啟用	在Class校驗器中，放鬆對訪問控制的檢查。 作用與reflection裡的setAccessible類似。
-XX:-UseConcMarkSweepGC	默認不啟用	啟用CMS低停頓垃圾收集器。 資料詳見：http://kenwublog.com/docs/CMS_GC.pdf
-XX:-UseParallelGC	-server時啟用 其他情況下，默認不啟用	策略為新生代使用並行清除，年老代使用單線程Mark-Sweep-Compact的垃圾收集器。
-XX:-UseParallelOldGC	默認不啟用	策略為老年代和新生代都使用並行清除的垃圾收集器。
-XX:-UseSerialGC	-client時啟用 其他情況下，默認不啟用	使用串行垃圾收集器。
-XX:+UseSplitVerifier	java5默認不啟用 java6默認啟用	使用新的Class類型校驗器 。 **新Class類型校驗器有什麼特點？** 新Class類型校驗器，將老的校驗步驟拆分成了兩步： 1，類型推斷。 2，類型校驗。 新類型校驗器通過在javac編譯時嵌入類型信息到bytecode中，省略了類型推斷這一步，從而提升了classloader的性能。 Classload順序（供參考） load -> **verify** -> prepare -> resove -> init **關聯選項：** -XX:+FailOverToOldVerifier
-XX:+FailOverToOldVerifier	Java6新引入選項，默認啟用	如果新的Class校驗器檢查失敗，則使用老的校驗器。 **為什麼會失敗？** 因為JDK6最高向下兼容到JDK1.2，而JDK1.2的class info 與JDK6的info存在

		較大的差異，所以新校驗器可能會出現校驗失敗的情況。 **關聯選項：** -XX:+UseSplitVerifier
-XX:+HandlePromotionFailure	java5以前是默認不啟用，java6默認啟用	關閉新生代收集擔保。 **什麼是新生代收集擔保？** 在一次理想化的minor gc中，Eden和First Survivor中的活躍對象會被覆制到Second Survivor。 然而，Second Survivor不一定能容納下所有從E和F區copy過來的活躍對象。 為了確保minor gc能夠順利完成，GC需要在年老代中額外保留一塊足以容納所有活躍對象的內存空間。 這個預留操作，就被稱之為新生代收集擔保（New Generation Guarantee）。如果預留操作無法完成時，仍會觸發major gc(full gc)。 **為什麼要關閉新生代收集擔保？** 因為在年老代中預留的空間大小，是無法精確計算的。 為了確保極端情況的發生，GC參考了最壞情況下的新生代內存佔用，即Eden+First Survivor。 這種策略無疑是在浪費年老代內存，從時序角度看，還會提前觸發Full GC。 為了避免如上情況的發生，JVM允許開發者手動關閉新生代收集擔保。 在開啟本選項後，minor gc將不再提供新生代收集擔保，而是在出現survior或年老代不夠用時，拋出promotion failed異常。
-XX:+UseSpinning	java1.4.2和1.5需要手動啟用, java6默認已啟用	啟用多線程自旋鎖優化。 **自旋鎖優化原理** 大家知道，Java的多線程安全是基於Lock機制實現的，而Lock的性能往往不如人意。 原因是，monitorenter與monitorexit這兩個控制多線程同步的bytecode原語，是JVM依賴操作系統互斥(mutex)來實現的。 互斥是一種會導致線程掛起，並在較短的時間內又必須重新調度回原線程的，較為消耗資源的操作。 為了避免進入OS互斥，Java6的開發者們提出了自旋鎖優化。 自旋鎖優化的原理是在線程進入OS互斥前，通過CAS自旋一定的次數來檢測鎖的釋放。 如果在自旋次數未達到預設值前鎖已被釋放，則當前線程會立即持有該鎖。 CAS檢測鎖的原理詳見: http://kenwublog.com/theory-of-lightweight-locking-upon-cas **關聯選項：** -XX:PreBlockSpin=10
-XX:PreBlockSpin=10	-XX:+UseSpinning必須先啟用，對於java6來說已經默認啟用了，這裡默認自旋10次	控制多線程自旋鎖優化的自旋次數。(什麼是自旋鎖優化？見-XX:+UseSpinning 處的描述) **關聯選項：** -XX:+UseSpinning
-XX:+ScavengeBeforeFullGC	默認啟用	在Full GC前觸發一次Minor GC。
-XX:+UseGCOverheadLimit	默認啟用	限制GC的運行時間。如果GC耗時過長，就拋OOM。
-XX:+UseTLAB	1.4.2以前和使用-client選項時，默認不啟用，其餘版本默認啟用	啟用線程本地緩存區（Thread Local）。
-XX:+UseThreadPriorities	默認啟用	使用本地線程的優先級。
-XX:+UseAltSigs	限於Solaris，默認啟用	為了防止與其他發送信號的應用程序衝突，允許使用候補信號替代 SIGUSR1 和SIGUSR2。
-XX:+UseBoundThreads	限於Solaris, 默認啟用	綁定所有的用戶線程到內核線程。 減少線程進入飢餓狀態（得不到任何cpu time）的次數。
-XX:+UseLWPSynchronization	限於solaris，默認啟用	使用輕量級進程（內核線程）替換線程同步。

-XX:+MaxFDLimit	限於Solaris，默認啟用	設置java進程可用文件描述符為操作系統允許的最大值。
-XX:+UseVMInterruptibleIO	限於solaris，默認啟用	在solaris中，允許運行時中斷線程 。

性能選項

選項與默認值	**默認值與限制**	**描述**
-XX:+AggressiveOpts	JDK 5 update 6後引入，但需要手動啟用。 JDK6默認啟用。	啟用JVM開發團隊最新的調優成果。例如編譯優化，偏向鎖，並行年老代收集等。
-XX:CompileThreshold=10000	1000	通過JIT編譯器，將方法編譯成機器碼的觸發閥值，可以理解為調用方法的次數，例如調1000次，將方法編譯為機器碼。
-XX:LargePageSizeInBytes=4m	默認4m amd64位：2m	設置堆內存的內存頁大小。 調整內存頁的方法和性能提升原理，詳見 http://kenwublog.com/tune-large-page-for-jvm-optimization
-XX:MaxHeapFreeRatio=70	70	GC後，如果發現空閒堆內存佔到整個預估上限值的70%，則收縮預估上限值。 **什麼是預估上限值？** JVM在啟動時，會申請最大值（-Xmx指定的數值）的地址空間，但其中絕大部分空間不會被立即分配(virtual)。 它們會一直保留著，直到運行過程中，JVM發現實際佔用接近已分配上限值時，才從virtual裡再分配掉一部分內存。 這裡提到的已分配上限值，也可以叫做預估上限值。 引入預估上限值的好處是，可以有效地控制堆的大小。堆越小，GC效率越高嘛。 注意：預估上限值的大小一定小於或等於最大值。
-XX:MaxNewSize=size	1.3.1 Sparc: 32m 1.3.1 x86: 2.5m	新生代佔整個堆內存的最大值。
-XX:MaxPermSize=64m	5.0以後: 64 bit VMs會增大預設值的30% 1.4 amd64: 96m 1.3.1 -client: 32m 其他默認64m	Perm（俗稱方法區）佔整個堆內存的最大值。
-XX:MinHeapFreeRatio=40	40	GC後，如果發現空閒堆內存佔到整個預估上限值的40%，則增大上限值。 (什麼是預估上限值？見 -XX:MaxHeapFreeRatio 處的描述) **關聯選項：** -XX:MaxHeapFreeRatio=70
-XX:NewRatio=2	Sparc -client: 8 x86 -server: 8 x86 -client: 12 -client: 4 (1.3)	新生代和年老代的堆內存佔用比例。 例如2例如2表示新生代佔年老代的1/2，佔整個堆內存的1/3。

	8 (1.3.1+) x86: 12 其他默認 2	
-XX:NewSize=2.125m	5.0以後: 64 bit Vms 會增大預設值的30% x86: 1m x86, 5.0以後: 640k 其他默認 2.125m	新生代預估上限的默認值。(什麼是預估上限值？見 -XX:MaxHeapFreeRatio 處的描述)
-XX:ReservedCodeCacheSize=32m	Solaris 64-bit, amd64, -server x86: 48m 1.5.0_06之前, Solaris 64-bit amd64: 1024m 其他默認 32m	設置代碼緩存的最大值，編譯時用。
-XX:SurvivorRatio=8	Solaris amd64: 6 Sparc in 1.3.1: 25 Solaris platforms 5.0以前: 32 其他默認 8	Eden與Survivor的佔用比例。例如8表示，一個survivor區佔用 1/8 的Eden內存，即1/10的新生代內存，為什麼不是1/9？ 因為我們的新生代有2個survivor，即S1和S22。所以survivor總共是佔用新生代內存的 2/10，Eden與新生代的佔比則為 8/10。
-XX:TargetSurvivorRatio=50	50	實際使用的survivor空間大小佔比。默認是50%，最高90%。
-XX:ThreadStackSize=512	Sparc: 512 Solaris x86: 320 (5.0以前 256) Sparc 64 bit: 1024 Linux amd64: 1024 (5.0以前 0) 其他默認 512.	線程堆棧大小
-XX:+UseBiasedLocking	JDK 5 update 6後引入，但需要手動啟用。 JDK6默認啟用。	啟用偏向鎖。 偏向鎖原理詳見 http://kenwublog.com/theory-of-java-biased-locking
-XX:+UseFastAccessorMethods	默認啟用	優化原始類型的getter方法性能。
-XX:-UseISM	默認啟用	啟用solaris的ISM。 詳見Intimate Shared Memory.
-XX:+UseLargePages	JDK 5 update 5後引入，但需	啟用大內存分頁。 調整內存頁的方法和性能提升原理，詳見http://kenwublog.com/tune-large-

	要手動啟用。 JDK6默認啟用。	page-for-jvm-optimization **關聯選項** -XX:LargePageSizeInBytes=4m
-XX:+UseMPSS	1.4.1 之前: 不啟用 其餘版本默認啟用	啟用solaris的MPSS，不能與ISM同時使用。
-XX:+StringCache	默認啟用	啟用字符串緩存。
-XX:AllocatePrefetchLines=1	1	與機器碼指令預讀相關的一個選項，資料比較少，本文檔不做解釋。有興趣的朋友請自行閱讀官方doc。
-XX:AllocatePrefetchStyle=1	1	與機器碼指令預讀相關的一個選項，資料比較少，本文檔不做解釋。有興趣的朋友請自行閱讀官方doc。

調試選項

選項與默認值	**默認值與限制**	**描述**
-XX:-CITime	1.4引入。默認啟用	打印JIT編譯器編譯耗時。
-XX:ErrorFile=./hs_err_pid<pid>.log	Java 6引入。	如果JVM crashed，將錯誤日誌輸出到指定文件路徑。
-XX:-ExtendedDTraceProbes	Java6引入，限於solaris默認不啟用	啟用dtrace診斷。
-XX:HeapDumpPath=./java_pid<pid>.hprof	默認是java進程啟動位置，即user.dir	堆內存快照的存儲文件路徑。 **什麼是堆內存快照？** 當java進程因OOM或crash被OS強制終止後，會生成一個hprof（HeapPROFling）格式的堆內存快照文件。該文件用於線下調試，診斷，查找問題。 文件名一般為 java_<pid>_<date>_<time>_heapDump.hprof 解析快照文件，可以使用 jhat, eclipse MAT，gdb等工具。
-XX:-HeapDumpOnOutOfMemoryError	1.4.2 update12和 5.0 update 7引入。默認不啟用	在OOM時，輸出一個dump.core文件，記錄當時的堆內存快照（什麼是堆內存快照? 見 -XX:HeapDumpPath 處的描述）。
-XX:OnError="<cmd args>;<cmd args>"	1.4.2 update 9引入	當java每拋出一個ERROR時，運行指定命令行指令集。指令集是與OS環境相關的，在linux下多數是bash腳本，windows下是dos批處理。
-XX:OnOutOfMemoryError="<cmd args>;<cmd args>"	1.4.2 update 12和java6時引入	當第一次發生OOM時，運行指定命令行指令集。指令集是與OS環境相關的，在linux下多數是bash腳本，windows下是dos批處理。
-XX:-PrintClassHistogram	默認不啟用	在Windows下, 按ctrl-break或Linux下是執行kill -3（發送SIGQUIT信號）時，打印class柱狀圖。 Jmap –histo pid也實現了相同的功能。 詳見 http://java.sun.com/javase/6/docs/technotes/tools/share/jmap.html
-XX:-PrintConcurrentLocks	默認不啟用	在thread dump的同時，打印java.util.concurrent的鎖狀態。 Jstack –l pid 也同樣實現了同樣的功能。 詳見 http://java.sun.com/javase/6/docs/technotes/tools/share/jstack.html

-XX:-PrintCommandLineFlags	5.0 引入，默認不啟用	Java啟動時，往stdout打印當前啟用的非穩態jvm options。 例如： -XX:+UseConcMarkSweepGC -XX:+HeapDumpOnOutOfMemoryError -XX:+DoEscapeAnalysis
-XX:-PrintCompilation	默認不啟用	往stdout打印方法被JIT編譯時的信息。 例如： 1 java.lang.String::charAt (33 bytes)
-XX:-PrintGC	默認不啟用	開啟GC日誌打印。 打印格式例如： [Full GC 131115K->7482K(1015808K), 0.1633180 secs] 該選項可通過 com.sun.management.HotSpotDiagnosticMXBean API 和 Jconsole 動態啟用。 詳見 http://java.sun.com/developer/technicalArticles/J2SE/monitoring/#Heap_Dump
-XX:-PrintGCDetails	1.4.0引入，默認不啟用	打印GC回收的細節。 打印格式例如： [Full GC (System) [Tenured: 0K->2394K(466048K), 0.0624140 secs] 30822K->2394K(518464K), [Perm : 10443K->10443K(16384K)], 0.0625410 secs] [Times: user=0.05 sys=0.01, real=0.06 secs] 該選項可通過 com.sun.management.HotSpotDiagnosticMXBean API 和 Jconsole 動態啟用。 詳見 http://java.sun.com/developer/technicalArticles/J2SE/monitoring/#Heap_Dump
-XX:-PrintGCTimeStamps	默認不啟用	打印GC停頓耗時。 打印格式例如： **2.744**: [Full GC (System) 2.744: [Tenured: 0K->2441K(466048K), 0.0598400 secs] 31754K->2441K(518464K), [Perm : 10717K->10717K(16384K)], 0.0599570 secs] [Times: user=0.06 sys=0.00, real=0.06 secs] 該選項可通過 com.sun.management.HotSpotDiagnosticMXBean API 和 Jconsole 動態啟用。 詳見 http://java.sun.com/developer/technicalArticles/J2SE/monitoring/#Heap_Dump
-XX:-PrintTenuringDistribution	默認不啟用	打印對象的存活期限信息。 打印格式例如： [GC Desired survivor size 4653056 bytes, new threshold 32 (max 32) - age 1: 2330640 bytes, 2330640 total - age 2: 9520 bytes, 2340160 total 204009K->21850K(515200K), 0.1563482 secs] Age1 2表示在第1和2次GC後存活的對象大小。
-XX:-TraceClassLoading	默認不啟用	打印class裝載信息到stdout。記Loaded狀態。 例如： [Loaded java.lang.Object from /opt/taobao/install/jdk1.6.0_07/jre/lib/rt.jar]
-XX:-TraceClassLoadingPreorder	1.4.2引入，默認不啟用	按class的引用/依賴順序打印類裝載信息到stdout。不同於TraceClassLoading，本選項只記 Loading狀態。 例如： [Loading java.lang.Object from /home/confsrv/jdk1.6.0_14/jre/lib/rt.jar]

-XX:-TraceClassResolution	1.4.2引入，默認不啟用	打印所有靜態類，常量的代碼引用位置。用於debug。 例如： RESOLVE java.util.HashMap java.util.HashMap$Entry HashMap.java:209 說明HashMap類的209行引用了靜態類 java.util.HashMap$Entry
-XX:-TraceClassUnloading	默認不啟用	打印class的卸載信息到stdout。記Unloaded狀態。
-XX:-TraceLoaderConstraints	Java6 引入，默認不啟用	打印class的裝載策略變化信息到stdout。 例如： [Adding new constraint for name: java/lang/String, loader[0]: sun/misc/Launcher$ExtClassLoader, loader[1]: <bootloader>] [Setting class object in existing constraint for name: [Ljava/lang/Object; and loader sun/misc/Launcher$ExtClassLoader] [Updating constraint for name org/xml/sax/InputSource, loader <bootloader>, by setting class object] [Extending constraint for name java/lang/Object by adding loader[15]: sun/reflect/DelegatingClassLoader] 裝載策略變化是實現classloader隔離/名稱空間一致性的關鍵技術。 對此感興趣的朋友，詳見 http://kenwublog.com/docs/Dynamic+Class+Loading+in+the+Java+Virtual+Machine.pdf 中的 contraint rules一章。
-XX:+PerfSaveDataToFile	默認啟用	當java進程因OOM或crashed被強制終止後，生成一個堆快照文件（什麼是堆內存快照? 見 -XX:HeapDumpPath 處的描述）。

作者敬告
完善的單元測試，功能回歸測試，和性能基準測試可以減少因調整非穩態JVM選項帶來的風險。

參考資料
Java6性能調優白皮書
http://java.sun.com/performance/reference/whitepapers/6_performance.html
Java6 GC調優指南
http://java.sun.com/javase/technologies/hotspot/gc/gc_tuning_6.html
更為全面的options列表
http://blogs.sun.com/watt/resource/jvm-options-list.html