Java Platform, Standard Edition

JRockit to HotSpot Migration Guide

Release 8

E63235-01

June 2015

ORACLE

Java Platform, Standard Edition JRockit to HotSpot Migration Guide, Release 8

E63235-01

Primary Author: Devika Gollapudi

Contributing Author:

Contributor:

Preface

This guide helps users of Oracle JRockit migrate to Java HotSpot VM (Java platform, Standard Edition). The document describes the command-line options and tools available in Oracle JRockit and their equivalents in the HotSpot JVM.

Audience

The target audience for this document comprises of developers and users who are using Oracle JRockit and planning to migrate to Java Development Kit (JDK), which is Oracle's implementation of the Java Platform, Standard Edition (Java SE). The current release is Java SE 8 and JDK 8; however, most of the information in this document can be applied to releases earlier than JDK 8.

This document is intended for readers with a detailed understanding of the components of the Java HotSpot VM, and also some understanding of concepts such as garbage collection, threads, and native libraries. In addition, it is assumed that the reader is reasonably proficient with the operating system where the Java application is developed and run.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Documents

For more information, see the Java SE 8 documentation at:

http://docs.oracle.com/javase/8/docs/

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1 Introduction

This document provides simple guidelines to help migrate applications from Oracle JRockit to HotSpot JVM. It contains sections for each component of the JVM system that describe the equivalents of those in both Oracle JRockit and HotSpot JVM, and also list the corresponding important JVM options of those components. It includes tables mapping the complete set of Oracle JRockit `-X` and `-XX` command-line options to the ones available in the HotSpot.

> **Note:** Some of the tools described in this document require a commercial license for use in production. To learn more about commercial features and how to enable them, see `http://www.oracle.com/technetwork/java/javaseproducts/`.

Heap Sizing

HotSpot has the same options as Oracle JRockit to set the initial and the maximum Java heap size.

Table 1–1 Heap Size

Option	Oracle JRockit	HotSpot
`-Xms`	Sets the initial and minimum size of the heap	Sets the initial and minimum size of the heap
`-Xmx`	Sets the maximum size of the heap	Sets the maximum size of the heap

When migrating from Oracle JRockit to HotSpot, the Java heap size should essentially remain the same.

2

Garbage Collectors

This chapter describes garbage collection tuning options available in Oracle JRockit and HotSpot and compares their functionality and performance.

- Tuning Garbage Collection
- HotSpot GC Tuning Guide

> **Note:** Some of the tools described in this document require a commercial license for use in production. To learn more about commercial features and how to enable them, see http://www.oracle.com/technetwork/java/javaseproducts/.

Tuning Garbage Collection

The following table lists important garbage collection (GC) tuning options available in Oracle JRockit and HotSpot.

Table 2–1 Garbage Collectors

Oracle JRockit Garbage Collectors	HotSpot Garbage Collectors	Note
Throughput collector set using any of these options: ■ `-Xgc:throughput` ■ `-Xgc:genpar` ■ `-Xgc:singlepar` ■ `-Xgc:parallel`	Throughput collector: `-XX:+UseParallelGC` – Use parallel collector for the young generation `-XX:+UseParallelOldGC` – Use parallel collector for both young and old generation	On Server-class machines, throughput collector is the default collector. Since JDK 7u4, using `-XX:+UseParallelGC` (explicitly set or picked ergonomically) also sets `UseParallelOldGC` and enables the parallel collector for the tenured generation as well. The number of parallel GC threads can be controlled using `-XX:ParallelGCThreads=n`

Low latency collector set using any of the following options:
`-Xgc:pausetime`
`-Xgc:gencon`
`-Xgc:singlecon`
	`-XX:+UseConcurrentMarkSweepGC`
	Or
	`-XX:+UseG1GC`
		The Java HotSpot VM offers a choice between two mostly concurrent collectors:
		■ Concurrent Mark Sweep (CMS) Collector is for applications that prefer shorter garbage collection pauses and can afford to share processor resources with the garbage collection.
		■ Garbage-First Garbage Collector is a server-style collector is for multiprocessor machines with large memories. It meets garbage collection pause time goals with high probability while achieving high throughput.

Table 2–1 (Cont.) Garbage Collectors

Oracle JRockit Garbage Collectors	HotSpot Garbage Collectors	Note
`-Xgc:deterministic`	(see note)	There is no real-time deterministic collector available in HotSpot. However G1 collector (enabled using `-XX:+UseG1GC`) attempts to meet the garbage collection pause time goals with high probability while achieving high throughput. Note that `DetGC` was designed to provide really short (for example, 1ms) and highly predictable pause times for collections but G1GC does not promise to work with such short pause times.

HotSpot GC Tuning Guide

For further GC tuning, refer to the following document:

http://docs.oracle.com/javase/8/docs/technotes/guides/vm/gctuning/

3

Runtime

This chapter describes important options that control the runtime behavior of the HotSpot VM.

> **Note:** Some of the tools described in this document require a commercial license for use in production. To learn more about commercial features and how to enable them, see http://www.oracle.com/technetwork/java/javaseproducts/.

Runtime Options

The following table lists some important equivalent options of the runtime subsystem in Oracle JRockit and HotSpot.

Table 3–1 Runtime Options

Oracle JRockit	HotSpot	Note
`-XX:+UseLazyUnlocking`	`-XX:+UseBiasedLocking`	`UseBiasedLocking` improves the performance of uncontended synchronization. This option is enabled by default. However If the application has high contended synchronization, then disabling `UseBiasedLocking` benefits the performance.
`-XlargePages`	`-XX:+UseLargePages`	In HotSpot VM, this option is ON by default on Solaris. On Linux, this has been disabled since 7u60 and 8. Use `-XX:+UseLargePages` to enable the use of large pages on the platforms where it is disabled by default. It must be noted that `-XX:+UseLargePages` does not enable the use of large pages in the MetaSpace. To enable this option, add `-XX:+UseLargePagesInMetaspace`.
`-XX:MaxLargePageSize`	`-XX:LargePageSizeInBytes=size`	Sets the maximum size (in bytes) for large pages used for Java heap. By default, the size is set to 0, meaning that the JVM chooses the size for large pages automatically.
`-XXcompressedRefs`	`-XX:+UseCompressedOops`	Use of compressed oops is the default for 64-bit HotSpot JVM processes when `-Xmx` is not specified and for values of `-Xmx` less than 32 gigabytes.

4

Compilation Optimization

This chapter describes various compiler options available in Oracle JRockit and HotSpot VMs to optimize compilation.

- Compiler Considerations

> **Note:** Some of the tools described in this document require a commercial license for use in production. To learn more about commercial features and how to enable them, see http://www.oracle.com/technetwork/java/javaseproducts/.

4.1 Compiler Considerations

Unlike Oracle JRockit, HotSpot features a Java byte code interpreter in addition to two different JIT compilers - client (aka C1) and server (aka C2)). HotSpot VM defaults to interpreting Java byte code, and will only JIT compile methods that runtime profiling determines to be "hot" - the methods that have been executed for a threshold number of times. Originally, users had to chose at startup which of the two JIT compilers, client or server, would be used. The client compiler compiles methods quickly, but emits machine code that is less optimized than the server compiler. By comparison, the server JIT compiler often takes more time (and memory) to compile the same methods, but generates better optimized machine code than the code produced by the client compiler. The result is that the client compiler allows most applications to start up faster (because of less compilation overhead), but the server compiler untimely provides better run-time performance once the application has reached stead-state (has warmed up). Used independently, each of these two compilers serve two different use-cases:

- client: quick startup and smaller memory footprint is more important than steady-state performance
- server: steady–state performance is more important than a quick startup

If having to chose a single JIT compiler, most Oracle JRockit users should chose the server compiler. As Oracle JRockit was designed as a server side JVM, most environments that use Oracle JRockit are server deployments like WLS or Coherence. The one notable exception would be cases where Oracle JRockit was used to run a smaller client application. For example, the client compiler would probably be a better fit for a command line administration tool like WLST.

Oracle JRockit JVM compiles a Java method and generates the machine code for it the very first time it is invoked. This compiled code of frequently invoked methods is then later optimized in the background by an Optimizer thread. This is completely different

from the HotSpot JVM where methods are first interpreted and later compiled either by the Client (less optimizations) or the Server (more optimizations) compiler.

The client compiler can be invoked using `-client` JVM option and the server compile can be invoked using `-server` JVM option. The server compiler is selected by default on the server-class machines.

Tiered compilation, introduced in Java SE 7, brings client startup speeds to the server VM. A server VM uses the interpreter to collect profiling information about methods that is fed into the compiler. In the tiered scheme, in addition to the interpreter, the client compiler generates compiled versions of methods that collect profiling information about themselves. Since the compiled code is substantially faster than the interpreter, the program executes with greater performance during this profiling phase. In many cases, a startup that is even faster than with the client VM can be achieved because the final code produced by the server compiler may be already available during the early stages of application initialization. The tiered scheme can also achieve better peak performance than a regular server VM because the faster profiling phase allows a longer period of profiling, which may yield better optimization. Use the `-XX:+TieredCompilation` flag with the java command to enable tiered compilation.

In Java SE 8, Tiered compilation is the default mode for the server VM. Both 32 and 64 bit modes are supported. `-XX:-TieredCompilation` flag can be used to disable tiered compilation.

Important HotSpot JIT Compiler Options

The following table lists some important Oracle JRockit and HotSpot compiler options.

Table 4–1 JIT Compiler Options

Oracle JRockit	HotSpot	Note
`-XnoOpt` `-XXoptFile:<file>`	Because JIT compilation in HotSpot can be considered analogous to optimization in Oracle JRockit (that is both techniques are only used on methods that are determined by profiling to be "hot"), the HotSpot equivalent to Oracle JRockit's -XnoOpt is -Xint, where no JIT compilation is done at all, and only the byte code interpreter is used to execute all methods. This may result in a substantial performance impact, but can be useful for the same types of situations where -XnoOpt was used for Oracle JRockit: Troubleshooting or working around possible compiler issues. Like Oracle JRockit, HotSpot also offers ways to exclude methods from compilation and/or to turn off specific optimizations on them. If you are using XnoOpt or XXoptFile options with Oracle JRockit VM to turn off the optimization on certain methods as you were facing some issues when these methods were optimized, then these options should not be directly translated to HotSpot options to exclude the compilation and/or turn off specific optimizations on these methods. The exact same compilation/optimization issues observed with the Oracle JRockit JVM for any specific methods are very unlikely to be present with the HotSpot JVM. So, to begin with, it is best to remove these options when migrating to the HotSpot JVM. Equivalent HotSpot JVM options: ■ `-XX:CompileCommand=command,method[,option]` Specifies a command to perform on a method. For example, to exclude the indexOf() method of the String class from being compiled, use the following: `-XX:CompileCommand=exclude,java/lang/String.indexOf` ■ `-XX:CompileCommandFile=<filename>` Sets the file from which JIT compiler commands are read. By default, the .hotspot_compiler file is used to store commands performed by the JIT compiler. ■ `-XX:CompileOnly=<methods>` Sets the list of methods (separated by commas) to which compilation should be restricted. ■ `-XX:CompileThreshold=<invocations>` Sets the number of interpreted method invocations before compilation. By default, in the server JVM, the JIT compiler performs 10,000 interpreted method invocations to gather information for efficient compilation. For the client JVM, the default setting is 1,500 invocations.	Options CompileCommand, CompileCommandFile, CompileOnly and CompileThreshold can be used to disable or delay the compilation of specified methods.

Table 4–1 (Cont.) JIT Compiler Options

Oracle JRockit	HotSpot	Note
`-XX:OptThreads`	There are no optimization threads in HotSpot JVM. The count of compiler threads that perform both the compilation and the optimizations can be set using: `-XX:CICompilerCount=<threads>`	Sets the number of compiler threads to use for compilation. By default, the number of threads is set to 2 for the server JVM, to 1 for the client JVM, and it scales to the number of cores if tiered compilation is used.
`-XX:+ReserveCodeMemory` `-XX:MaxCodeMemory=<size>`	`-XX:ReservedCodeCacheSize=<size>`	Sets the maximum code cache size (in bytes) for JIT-compiled code. This option is equivalent to `-Xmaxjitcodesize`.
None	`-XX:+TieredCompilation`	Enables the use of tiered compilation. On JDK8 this option is enabled by default. Only the Java HotSpot Server VM supports this option.

5

Logging

This chapter describes various logging options available in Oracle JRockit and HotSpot.

- Verbose Logging
- HotSpot Logging Options

> **Note:** Some of the tools described in this document require a commercial license for use in production. To learn more about commercial features and how to enable them, see http://www.oracle.com/technetwork/java/javaseproducts/.

Verbose Logging

Verbose logging in HotSpot can be turned on using the `-verbose` option. There are some specific flags that can be used with this option to get area-specific verbose output.

The following table lists various logging options available in Oracle JRockit and compares them with the options available in HotSpot.

Table 5–1 Verbose Logging

Oracle JRockit Verbose Module	HotSpot Option	Note
`alloc`	--	--
`class`	`-verbose:class`	Displays information.
`codegen`	--	--
`compaction`	--	--
`cpuinfo`	--	--
`exceptions`	--	--
`gc`	-verbose:gc	Displays information about each garbage collection (GC) event.
`gcheuristic`	--	--
`gcpause`	--	--
`gcpausetree`	--	--
`gcreport`	--	--
`load`	--	--
`memory`	--	--
`memdbg`	--	--
`opt`	--	--
`refobj`	--	--
`starttime`	--	--
`shutdown`	--	--
`systemgc`	--	--
`timing`	--	--
--	`-verbose:jni`	Displays information about the use of native methods and other Java Native Interface (JNI) activity.

HotSpot Logging Options

These are some of the common logging options available in HotSpot that can be used to turn on the diagnostic output for a specific subsystem within the HotSpot JVM.

Table 5–2 Logging Options

HotSpot Logging Options	Note
`-Xloggc:<filename>`	Sets the file to which verbose GC event information should be redirected for logging. The information written to this file is similar to the output of -verbose:gc with the time elapsed since the first GC event preceding each logged event. The `-Xloggc` option overrides `-verbose:gc` if both are given with the same java command.
`-XX:LogFile=<path>`	Sets the path and file name where log data is written.
`-XX:+PrintCommandLineFlags`	Enables printing of ergonomically selected JVM flags that appeared on the command line.
`-XX:+PrintNMTStatistics`	Enables printing of collected native memory tracking data at JVM exit when native memory tracking is enabled
`-XX:+LogCompilation`	Enables logging of compilation activity to a file named hotspot.log in the current working directory. You can specify a different log file path and name using the `-XX:LogFile` option. The `-XX:+LogC`ompilation option must be used together with the `-XX:UnlockDiagnosticVMOptions` option that unlocks diagnostic JVM options.
`-XX:+PrintAssembly`	Enables printing of assembly code resulting from JIT compilation of Java bytecode by using the external disassembler.so library. This option enables you to see the generated code, which may help you to diagnose performance issues. This option must be used together with the `-XX:UnlockDiagnosticVMOptions` option that unlocks diagnostic JVM options.
`-XX:+PrintCompilation`	Enables verbose diagnostic output from the JVM by printing a message to the console every time a method is compiled.
`-XX:+PrintInlining`	Enables printing of inlining decisions. This option enables you to see which methods are getting inlined.
`-XX:+PrintClassHistogram`	Enables printing of a class instance histogram after a Control+C event (SIGTERM). By default, this option is disabled.
`-XX:+PrintConcurrentLocks`	Enables printing of java.util.concurrent locks after a Control+C event (SIGTERM). By default, this option is disabled.
`-XX:+G1PrintHeapRegions`	Enables the printing of information about which regions are allocated and which are reclaimed by the G1 collector.
`-XX:+PrintAdaptiveSizePolicy`	Enables printing of information about adaptive generation sizing.
`-XX:+PrintGC`	Enables printing of messages at every GC.
`-XX:+PrintGCApplicationConcurrentTime`	Enables printing of how much time elapsed since the last pause (for example, a GC pause).
`-XX:+PrintGCApplicationStoppedTime`	Enables printing of how much time the pause (for example, a GC pause) lasted.

Table 5–2 (Cont.) Logging Options

HotSpot Logging Options	Note
`-XX:+PrintGCDateStamps`	Enables printing of a date stamp at every GC.
`-XX:+PrintGCDetails`	Enables printing of detailed messages at every GC.
`-XX:+PrintGCTaskTimeStamps`	Enables printing of time stamps for every individual GC worker thread task.
`-XX:+PrintGCTimeStamps`	Enables printing of time stamps at every GC.
`-XX:+PrintStringDeduplicationStatistics`	Prints detailed deduplication statistics.
`-XX:+PrintTenuringDistribution`	Enables printing of tenuring age information.

6

Command Line Options

This chapter describes various HotSpot command line options and compares them with those available in Oracle JRockit.

- Mapping of Oracle JRockit to HotSpot Command Line Options

> **Note:** Some of the tools described in this document require a commercial license for use in production. To learn more about commercial features and how to enable them, see `http://www.oracle.com/technetwork/java/javaseproducts/`.

Mapping of Oracle JRockit to HotSpot Command Line Options

This section can be used as a reference by users who are searching for functionality similar to a specific Oracle JRockit flag they may be familiar with. This section seeks to provide either a one-to-one mapping of Oracle JRockit options to HotSpot options, or refers to the other sections of this document. There may be certain Oracle JRockit options for which there are no corresponding HotSpot JVM options. Some of the mapped HotSpot options may not be exactly equivalent to the Oracle JRockit options and may provide slightly different behavior on the HotSpot.

When migrating, simply translating every option used with Oracle JRockit into similar HotSpot option is not recommended. Especially for performance-related options, the best practice is to start by only specifying the Java heap size and the garbage collector (CMS, G1, and so on). Any additional tuning for HotSpot, if necessary at all, should only be done based off of new benchmarking and profiling done with HotSpot. It is not advised to assume that most, if any, JVM-level tuning decisions made for an Oracle JRockit configuration will also apply as-is to a HotSpot configuration.

See Oracle JRockit Documentation for more information.

Table 6–1 -X Command-Line Options

Oracle JRockit	HotSpot	Added In	Note
`-Xbootclasspath`	SAME		
`-Xbootclasspath/a`	SAME		
`-Xbootclasspath/p`	SAME		
`-Xcheck:jni`	SAME		
`-Xdebug`	SAME		
`-Xgc`	--		See section on GC for more details.
`-XgcPrio` (deprecated)	--		See section on GC for more details.
`-XlargePages`	`-XX:+UseLargePages`	5u5	See https://blogs.oracle.com/poonam/entry/uselargepages_on_linux
`-Xmanagement`	--		See http://docs.oracle.com/javase/7/docs/technotes/guides/management/agent.html
`-Xms`	SAME		
`-Xmx`	SAME		
`-XnoClassGC` (deprecated)	SAME		Should not use except for troubleshooting.
`-XnoOpt`	--		See section on Compilation Optimization for more details.
`-Xns`	SAME		
`-XpauseTarget`	`-XX:MaxGCPauseMillis=n`		See section on GC for more details.

Table 6–1 (Cont.) -X Command-Line Options

Oracle JRockit	HotSpot	Added In	Note
-Xrs	SAME		
-Xss	SAME		
-XstrictFP	--		
-Xverbose	-verbose		See section on Logging.
-Xverbosedecorations	--		See section on Logging.
-XverboseLog	--		See section on Logging.
-XverboseTimeStamp	--		See section on Logging.
-Xverify	SAME		

Table 6–2 -XX Command-Line Options

Oracle JRockit	HotSpot	Note On HotSpot Options
-XXaggressive	-XX:+AggressiveHeap -XX:+AggressiveOpts	-XX:+AggressiveHeap enables Java heap optimization. This sets various parameters to be optimal for long-running jobs with intensive memory allocation, based on the configuration of the computer (RAM and CPU). By default, the option is disabled and the heap is not optimized. -XX:+AggressiveOpts enables other non-heap related optimization.
-XX:AllocChunkSize	Related options: ■ -XX:AllocateInstancePrefetchLines=<lines> ■ -XX:AllocatePrefetchDistance=<size> ■ -XX:AllocatePrefetchInstr=<instruction> ■ -XX:AllocatePrefetchLines=<lines> ■ -XX:AllocatePrefetchStepSize=<size> ■ -XX:AllocatePrefetchStyle=<style>	
-XX:+|-CheckJNICalls	-Xcheck:jni	
-XX:+|-CheckStacks	--	
-XXcompaction	--	
-XXcompactRatio (deprecated)	--	

Table 6–2 (Cont.) -XX Command-Line Options

Oracle JRockit	HotSpot	Note On HotSpot Options
`-XXcompactSetLimit` (deprecated)	--	
`-XXcompactSetLimitPerObject` (deprecated)	--	
`-XXcompressedRefs`	`-XX:-UseCompressedOops`	See section on Runtime options for more details.
`-XX:+|-CrashOnOutOfMemoryError`	Can achieve the same by using `-XX:OnOutOfMemoryError=<command>`	Sets a custom command or a series of semicolon-separated commands to run when an OutOfMemoryError exception is first thrown. For example: `java -XX:OnOutOfMemoryError=" kill -11 %p" JavaProgram`
`-XX:+|-DisableAttachMechanism`	SAME	
`-XXdumpFullState`	--	On HotSpot side, there is an option `CreateMinidumpOnCrash` to enable the dumping of minidumps upon fatal errors on Windows platform.
`-XXdumpSize`	--	
`-XX:ExceptionTraceFilter`	--	
`-XX:+|-ExitOnOutOfMemoryError`	Can achieve the same by using `-XX:OnOutOfMemoryError=<command>`	Sets a custom command or a series of semicolon-separated commands to run when an OutOfMemoryError exception is first thrown. For example: `java -XX:OnOutOfMemoryError=" kill -9 %p" JavaProgram`
`-XX:ExitOnOutOfMemoryErrorExitCode`	--	
`-XXexternalCompactRatio` (deprecated)	--	
`-XX:+|-FailOverToOldVerifier`	SAME	
`-XX:+|-FlightRecorder`	SAME	Enables the use of the Java Flight Recorder (JFR) during the runtime of the application. This is a commercial feature that requires you to also specify the `-XX:+UnlockCommercialFeatures` option.

Table 6–2 (Cont.) -XX Command-Line Options

Oracle JRockit	HotSpot	Note On HotSpot Options
`-XX:FlightRecorderOptions`	SAME	
`-XX:+|-FlightRecordingDumpOnUnhandledException`	--	
`-XX:FlightRecordingDumpPath`	--	
`-XXfullSystemGC`	Related options: ■ `-XX:+DisableExplicitGC` ■ `-XX:+ExplicitGCInvokesConcurrent` ■ `-XX:+ExplicitGCInvokesConcurrentAndUnloadsClasses`	See GC section for more details.
`-XXgcThreads`	Related options: ■ `-XX:ParallelGCThreads=<threads>` ■ `-XX:ConcGCThreads=<threads>`	See GC section for more details.
`-XX:GCTimePercentage`	--	
`-XX:GCTimeRatio`	--	
`-XXgcTrigger`	Related options: ■ `-XX:CMSInitiatingOccupancyFraction=<percent>` ■ `-XX:CMSTriggerRatio=<percent>`	See section on GC for more details.
`-XX:+|-HeapDiagnosticsOnOutOfMemoryError`	Can achieve the same by using `-XX:OnOutOfMemoryError=<command>`	Example: `java -XX:OnOutOfMemoryError="jmap -heap %p" JavaProgram`
`-XX:HeapDiagnosticsPath`	--	
`-XX:+|-HeapDumpOnCtrlBreak`	--	
`-XX:+|-HeapDumpOnOutOfMemoryError`	SAME	
`-XX:HeapDumpPath`	SAME	
`-XX:HeapDumpSegmentSize`	--	
`-XXheapParts` (deprecated)	--	
`-XXinternalCompactRatio` (deprecated)	--	
`-XX:+|-JavaDebug`	--	

Table 6–2 (Cont.) -XX Command-Line Options

Oracle JRockit	HotSpot	Note On HotSpot Options
`-XXkeepAreaRatio`	`XX:SurvivorRatio=<ratio>`	Sets the ratio between eden space size and survivor space size. By default, this option is set to 8. There is another option `-XX:InitialSurvivorRatio=ratio` to set the initial survivor space ratio used by the throughput garbage collector.. Adaptive sizing is enabled by default with the throughput garbage collector by using the `-XX:+UseParallelGC` and `-XX:+UseParallelOldGC` options, and survivor space is resized according to the application behavior, starting with this initial value.
`-XXlargeObjectLimit` (deprecated)	--	
`-XX:MaxCodeMemory`	`-XX:ReservedCodeCacheSize=<size>`	See section on Compilation/Optimization for more details.
`-XX:MaxDirectMemorySize`	SAME	
`-XX:MaximumNurseryPercentage`	`-XX:NewRatio=<ratio>`	Sets the ratio between young and old generation sizes. By default, this option is set to 2.
`-XX:MaxLargePageSize`	`-XX:LargePageSizeInBytes=<size>`	See section on Runtime options for more details.
`-XX:MaxRecvBufferSize`	--	
`-XXminBlockSize` (deprecated)	--	
`-XXnoSystemGC`	Related options: ■ `-XX:+DisableExplicitGC` ■ `-XX:+ExplicitGCInvokesConcurrent` ■ `-XX:+ExplicitGCInvokesConcurrentAndUnloadsClasses`	See GC section for details.
`-XX:OptThreads`	`-XX:CICompilerCount=threads`	See section on Compilation/Optimization for more details.

Table 6–2 (Cont.) -XX Command-Line Options

Oracle JRockit	HotSpot	Note On HotSpot Options
-XX:+|-RedoAllocPrefetch	Related options: ■ -XX:AllocateInstancePrefetchLines=<lines> ■ -XX:AllocatePrefetchDistance=<size> ■ -XX:AllocatePrefetchInstr=<instruction> ■ -XX:AllocatePrefetchLines=<lines> ■ -XX:AllocatePrefetchStepSize=<size> ■ -XX:AllocatePrefetchStyle=<style>	
-XX:+|-ReserveCodeMemory	-XX:ReservedCodeCacheSize=<size>	See section on Compilation/Optimization for more details.
-XX:SegmentedHeapDumpThreshold	--	
-XXsetGC (deprecated)	--	
-XX:+|-StrictFP	--	
-XX:StartFlightRecording	SAME	
-XXtlaSize	XX:TLABSize=<size>	Sets the initial size (in bytes) of a thread-local allocation buffer (TLAB). If this option is set to 0, then the JVM chooses the initial size automatically.
-XX:TreeMapNodeSize	--	
-XX:+|-UseAdaptiveFatSpin	--	
-XX:+|-UseAllocPrefetch	Related options: ■ -XX:AllocateInstancePrefetchLines=<lines> ■ -XX:AllocatePrefetchDistance=<size> ■ -XX:AllocatePrefetchInstr=<instruction> ■ -XX:AllocatePrefetchLines=<lines> ■ -XX:AllocatePrefetchStepSize=<size> ■ -XX:AllocatePrefetchStyle=<style	
-XX:+|-UseCallProfiling	-XX:+UseTypeProfile	
-XX:+|-UseCfsAdaptedYield	--	

Table 6–2 (Cont.) -XX Command-Line Options

Oracle JRockit	HotSpot	Note On HotSpot Options
-XX:+|-UseClassGC	-Xnoclassgc	Disables garbage collection (GC) of classes. This can save some GC time, which shortens interruptions during the application run. When you specify Xnoclassgc at startup, the class objects in the application will be left untouched during GC and will always be considered live.
-XX:+|-UseCPoolGC	--	
-XX:+|-UseFastTime	--	
-XX:+|-UseFatSpin	--	
-XX:+|-UseLargePagesFor[Heap|Code]	■ -XX:+UseLargePages ■ -XX:+UseLargePagesInMetaspace	See section on Runtime options for more details.
-XX:+|-UseLazyUnlocking	-XX:+UseBiasedLocking	See section on Runtime options for more details.
-XX:+|-UseLockProfiling	--	
-XX:+|-UseLowAddressForHeap	--	No direct corresponding option available in HotSpot but the low heap base can be specified explicitly using HeapBaseMinAddress option.
-XX:+|-UseNewHashFunction	SAME	Only relevant for JDK5. Should not be used on JDK 6 or higher.
-XX:+|-UseThreadPriorities	SAME	On HS, enabled by default for Windows. On JR, disabled by default for Windows.

Table 6–3 Diagnostic Commands

Oracle JRockit	HotSpot
check_flightrecording	JFR.check
command_line	VM.command_line
dump_flightrecording	JFR.dump
exception_trace_filter	--
force_crash	--
heap_diagnostics	--
help	help
hprofdump	GC.heap_dump
kill_management_server	ManagementAgent.stop

Table 6–3 (Cont.) Diagnostic Commands

Oracle JRockit	HotSpot
list_vmflags	VM.flags
lockprofile_print	--
lockprofile_reset	--
memleakserver	--
print_class_summary	GC.class_stats
print_exceptions	--
print_memusage	VM.native_memory
print_object_summary	GC.class_histogram
print_threads	Thread.print
print_utf8pool	--
print_vm_state	--
runsystemgc	GC.run
set_filename	--
start_flightrecording	JFR.start
start_management_server	ManagementAgent.start ManagementAgent.start_local
stop_flightrecording	JFR.stop
stop_management_server	ManagementAgent.stop
timestamp	--
verbosity	--
version	VM.version

7

Common Migration Issues and Solutions

This chapter describes some common issues that can occur while migrating from Oracle JRockit to HotSpot VM, along with their solutions.

> **Note:** Some of the tools described in this document require a commercial license for use in production. To learn more about commercial features and how to enable them, see `http://www.oracle.com/technetwork/java/javaseproducts/`.

Common Migration Issues and Solutions

The following table lists some common issues that can occur during the migration process and solutions for resolving them.

Table 7–1 Migrations Issues and Solutions

Problem	Oracle JRockit Option	HotSpot Option	Comments
Performance degradation after migrating to JDK7. Issue was resolved with the use of `-XX:ReservedCodeCacheSize=1g`	`-XX:+ReserveCodeMemory` Default values: ■ When you use `-XX:+UseLargePagesForCode`: 64 MB ■ When you use `-XX:-UseLargePagesForCode`: 1024 MB	`-XX:ReservedCodeCacheSize` Default value on most of the platforms is 48 MB	With HotSpot VM, it was observed that in some cases increasing `ReservedCodeCacheSize` value, for example, `-XX:ReservedCodeCacheSize=1g`, improves the performance significantly.
Increased locking/unlocking events observed after switching to HotSpot. Disabling `UseBiasedLocking` helped improve the overall performance.	`-XX:-UseLazyUnlocking` (to disable)	`-XX:-UseBiasedLocking` (to disable)	`UseBiasedLocking` option improves the performance of uncontended synchronization. This option is enabled by default. However if the application has high contended synchronization, then disabling `UseBiasedLocking` benefits the performance. If you face performance issues due to locking/synchronization after migrating to HotSpot, turning off this option may provide some performance gains.

8

Troubleshooting Tools

This chapter describes various troubleshooting tools available in Java SE and compares their functionality to those available in Oracle JRockit.

> **Note:** Some of the tools described in this document require a commercial license for use in production. To learn more about commercial features and how to enable them, see `http://www.oracle.com/technetwork/java/javaseproducts/`.

Troubleshooting Tools Available in Java SE

The following table lists various tools available for troubleshooting in Java SE. Some of these tools have been brought over from Oracle JRockit to HotSpot VM for providing comparable functionality.

Table 8–1 Tools

Java SE Troubleshooting Tools	Notes/Resources
Java Flight Recorder and Mission Control	▪ Java Mission Control ▪ What are Java Flight Recordings ▪ How to produce a Flight Recording ▪ Inspect a Flight Recording ▪ Debug a Memory Leak Using Java Flight Recorder ▪ Java Mission Control User's Guide
Serviceability Agent	▪ Article on Serviceability Agent
Java VisualVM	▪ Troubleshoot with Java VisualVM ▪ Java VisualVM Guide
JConsole	▪ Troubleshoot with JConsole ▪ JConsole
jcmd command utility	▪ Troubleshoot with jcmd Utility ▪ jcmd
JDK utilities	There are many useful utilities bundled with JDK: ▪ jdb ▪ jhat ▪ jinfo ▪ jmap ▪ jps ▪ jstack ▪ jstat ▪ jrunscript ▪ jsadebugd ▪ jstatd
visualgc	visualgc Tool
Native Memory Tracking Too	-XX:NativeMemoryTracking=mode Specifies the mode for tracking JVM native memory usage. This option is useful for tracking the native memory usage by the JVM. http://hirt.se/blog/?p=401
JOverflow - Experimental plug-in	There is still no Memory Leak Detector Tool available in Java SE. However, there is an experimental plugin for Java Mission Control that can be used to detect memory leaks: http://hirt.se/blog/?p=343