Java Platform, Standard Edition

JRockit to HotSpot Migration Guide

Release 10
E91473-01
March 2018

Java Platform, Standard Edition JRockit to HotSpot Migration Guide, Release 10

E91473-01

Copyright © 1995, 2018, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Contents

Preface

This guide helps users of Oracle JRockit to 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 Java HotSpot VM (HotSpot).

Audience

The target audiences for this document are developers and users who are working on Oracle JRockit and planning to migrate to the Java Development Kit (JDK). The JDK is Oracle's implementation of the Java Platform, Standard Edition (Java SE). The current release is Java SE 10 and JDK 10. However, most of the information in this document can be applied to releases earlier than JDK 10.

This document is intended for readers who have a detailed understanding of the Java HotSpot VM components, and also have 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 systems 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

See JDK 10 Documentation for other JDK 10 guides.

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 Java HotSpot VM (HotSpot). It contains sections for each JVM system component that describe the equivalents of those components in both Oracle JRockit and HotSpot. The document also lists the corresponding JVM options of those components. It includes tables that map the complete set of Oracle JRockit `-X` and `-XX` command-line options to the ones available in 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 Oracle Java SE Advanced and Oracle Java SE Suite.

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

> **Note:**
>
> When migrating from Oracle JRockit to HotSpot, the Java heap size must essentially be the same.

2

Garbage Collectors

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

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	Notes
Throughput collector set using any of these options: • `-Xgc:throughput` • `-Xgc:genpar` • `-Xgc:singlepar` • `-Xgc:parallel`	Throughput collector: `–XX:+UseParallelGC` - Use parallel compacting collector for both young and old generation `-XX:+UseParallelGC -XX:-UseParallelOldGC` - Use parallel compacting collector for the young generation and serial mark-sweep for the old generation	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:+UseG1GC`
	Or
	`-XX:+UseConcurrentMarkSweepGC`
	Note:
	The CMS collector was deprecated in JDK 9.
		The HotSpot offers a choice between the two mostly concurrent collectors:
		• Garbage-First (G1) Garbage Collector is a server-style collector for multiprocessor machines with large memories. It meets garbage collection pause time goals with high probability while achieving high throughput
		• 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
		The Garbage-First Garbage Collector is the default collector.

To understand the various garbage collectors that are included with the HotSpot, see Introduction to Garbage Collection Tuning in *Java Platform, Standard Edition HotSpot Virtual Machine Garbage Collection Tuning Guide*.

3
Runtime

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

Runtime Options

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

Table 3-1 Runtime Options

Oracle JRockit	HotSpot	Notes
`-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 disable the `UseBiasedLocking` option to enhance the performance.
`-XlargePages`	`-XX:+UseLargePages`	In HotSpot, this option is enabled by default on the Solaris platform. On the Linux platform, this option was disabled from version 7u60 onwards. Use `-XX:+UseLargePages` to enable the use of large pages on the platforms where it is disabled by default. However, `-XX:+UseLargePages` doesn't enable the use of large pages in the MetaSpace. To enable this option, add `-XX:+UseLargePagesInMetaspace`.

Table 3-1 (Cont.) Runtime Options

Oracle JRockit	HotSpot	Notes
`-XX:MaxLargePageSize`	`-XX:LargePageSizeInBytes=size`	Sets the maximum size (in bytes) for large pages used for the Java heap. By default, the size is set to 0, which implies that the JVM chooses the size for large pages automatically.
`-XXcompressedRefs`	`-XX:+UseCompressedOops`	Use of `Compressed Oops` is the default for 64-bit HotSpot processes when `-Xmx` isn't specified and the values of `-Xmx` are less than 32 gigabytes.

4
Compilation Optimization

This topic describes the various compiler options available in Oracle JRockit and HotSpot to optimize compilation.

Compiler Considerations

Unlike Oracle JRockit, HotSpot features a Java byte code interpreter in addition to two different Just In Time (JIT) compilers, client (also known as C1) and sever (also known as C2).

This section provides details about the complier that you can use.

HotSpot VM defaults to interpreting Java byte code. It compiles (JIT compilation) methods that are executed for a predetermined number of times. JIT compliers are either client or server compilers.

- Client compiler: It compiles methods quickly but emits machine code that is less optimized than the server compiler. This complier is used for quick startup. Also, in this compiler, the smaller memory footprint is more important than steady-state performance.
- Server compiler: The compiler often takes more time (and memory) to compile the same methods. However, it generates better optimized machine code than the code generated by the client compiler. It provides better runtime performance after the application reaches the steady state.

The tiered compilation enhances the server VM startup speed by using client compiler as the first tier. A server VM uses the interpreter to collect the profiling information about the 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. As the compiled code is substantially faster than the interpreter, the program executes with greater performance during this profiling phase. Often, even a startup is faster because the final code produced by the server compiler is available during the early stages of application initialization. The tiered scheme can also achieve better peak performance than a regular server VM. This is because the faster profiling phase allows a longer period of profiling, which yields better optimization.

Tiered compilation is the default mode for the server VM. The 64-bit mode is supported. To enable tiered compilation manually, use the `-XX:+TieredCompilation` flag. You can disable tiered compilation by using the `-XX:-TieredCompilation` flag.

Oracle JRockit JVM compiles a Java method and generates the machine code for the first time it is invoked. This compiled code of frequently invoked methods is optimized in the background by an Optimizer thread. This code is different from the HotSpot where methods are interpreted first and compiled later, either by the client (fewer optimizations) or the server (more optimizations) compiler.

Compiler Control

Compiler Control provides an improved way to control the JVM compilers. The JVM compilation is done through compiler directive options. These options are method-specific and can be changed at run time. See Compiler Control.

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	Notes
`-XnoOpt` `-XXoptFile:<file>`	As JIT compilation in HotSpot is 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`. In this technique, no JIT compilation is done and only the byte code interpreter is used to execute all methods. This compilation might impact the performance. However, it can be useful when `-XnoOpt` is used for troubleshooting or working around possible compiler issues of Oracle JRockit. Like Oracle JRockit, HotSpot also offers ways to exclude methods from compilation or to disable specific optimizations on them. If there are any problems while optimizing the methods, then use `XnoOpt` or `XXoptFile` options with Oracle JRockit VM to disable the optimization on those methods. However, to exclude the compilation or disable specific optimizations on these methods, ensure that you don't directly translate to HotSpot options. The same compilation or optimization problems observed with the Oracle JRockit JVM for any specific methods are unlikely to happen with the HotSpot JVM. So, to begin with, it is best to remove these options while migrating to the HotSpot JVM. Equivalent HotSpot JVM options are: • `-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 must 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.	Options `CompileCommand`, `CompileCommandFile`, `CompileOnly`, and `CompileThreshold` can be used to disable or delay the compilation of specified methods.
`-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 and it scales to number of cores if tiered compilation is used.

Table 4-1 (Cont.) JIT Compiler Options

Oracle JRockit	HotSpot	Notes
`-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. This option is enabled by default from JDK 8 and later versions. Only the Java HotSpot Server VM supports this option.

5
Logging

This topic describes the various logging options available in Oracle JRockit and HotSpot:

- Verbose Logging
- HotSpot Logging Options

Verbose Logging

Verbose logging in HotSpot can be enabled 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	Notes
`alloc`	*NA*	*NA*
`class`	`-verbose:class`	Displays information about the classes that are being loaded.
`codegen`	*NA*	*NA*
`compaction`	*NA*	*NA*
`cpuinfo`	*NA*	*NA*
`exceptions`	*NA*	*NA*
`gc`	`-verbose:gc`	Displays information about each garbage collection (GC) event.
`gcheuristic`	*NA*	*NA*
`gcpause`	*NA*	*NA*
`gcpausetree`	*NA*	*NA*
`gcreport`	*NA*	*NA*
`load`	*NA*	*NA*
`memory`	*NA*	*NA*
`memdbg`	*NA*	*NA*
`opt`	*NA*	*NA*
`refobj`	*NA*	*NA*
`starttime`	*NA*	*NA*
`shutdown`	*NA*	*NA*
`systemgc`	*NA*	*NA*

Table 5-1 (Cont.) Verbose Logging

Oracle JRockit Verbose Module	HotSpot Option	Notes
`timing`	*NA*	*NA*
NA	`-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 enable the diagnostic output for a specific subsystem within the HotSpot JVM.

Table 5-2 Logging Options

HotSpot Logging Options	Notes
`-Xlog`	Enables the common logging system for all JVM components.
`-Xloggc:<filename>`	Sets the file to which verbose GC event information must be redirected for logging. The information written to this file is similar to the output of `-verbose:gc` with the time elapsed from the first GC event preceding each logged event. The `-Xloggc` option overrides the `-verbose:gc`, if both are given with the same `java` command. **Note:** This option was deprecated in JDK 9.
`-XX:LogFile=<path>`	Sets the path and file name where the log data is written.
`-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:+LogCompilation` option must be used together with the `-XX:UnlockDiagnosticVMOptions` option that unlocks diagnostic JVM options.
`-XX:+PrintCommandLineFlags`	Enables printing of the selected JVM flags that appeared on the command-line.
`-XX:+PrintGC`	Enables printing of messages at every GC. **Note:** This option was deprecated in JDK 9.

Table 5-2 (Cont.) Logging Options

HotSpot Logging Options	Notes
`-XX:+PrintGCDetails`	Enables printing of detailed messages at every GC. **Note:** This option was deprecated in JDK 9.
`-XX:+PrintNMTStatistics`	Enables printing of collected native memory tracking data at JVM exit when native memory tracking is enabled.
`-XX:+PrintNMTStatistics`	Enables printing of collected native memory tracking data at JVM exit when native memory tracking is enabled.
`-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 view the generated code, which helps you to diagnose the 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 view the methods that 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.

Table 5-3 GC Logging Options

GC Logging Options	Notes	
`-Xlog:gc`	Enables printing of messages at every GC. The `gc` is the main tag to log all GC related information. The `gc` tag is combined with other tags to log specific information. A few tags are listed in the following rows.	
	`-Xlog:gc +region=trace`	Enables the printing of information about the regions that are allocated and that are reclaimed by the G1 collector.
	`-Xlog:gc +ergo*=trace`	Enables printing of information about adaptive generation sizing.
	`-Xlog:safepoint`	Enables printing of the time elapsed from the last pause (for example, a GC pause).
	`-Xlog:gc+task=trace`	Enables printing of time stamps for every individual GC worker thread task.
	`-Xlog:gc::uptime,tid`	Enables printing of time stamps at every GC using the decorators `uptime` and `tid`.

Table 5-3 (Cont.) GC Logging Options

GC Logging Options		Notes
	`-Xlog:gc +stringdedup`	Prints detailed deduplication statistics.
	`-Xlog:gc+age=trace`	Enables printing of tenuring age information.
	`-Xlog:gc*`	Enables printing of detailed messages at every GC.
`-Xlog:gc:file=<filename>`	Logs messages with the `gc` tag to the file name specified. For example the option `-Xlog:gc:file=gc.txt` logs the messages to the `gc.txt` file.	

See Enable Logging with the JVM Unified Logging Framework in the *Java Platform, Standard Edition Tools Reference* guide.

6
Command-Line Options

This topic describes the various HotSpot command-line options and compares them with those available in Oracle JRockit:

- Mapping of Oracle JRockit to HotSpot Command-Line Options
- jcmd Commands

Mapping of Oracle JRockit to HotSpot Command-Line Options

Certain Oracle JRockit command-line options are similar to HotSpot options.

This section provides either a one-to-one mapping of Oracle JRockit options to HotSpot options, or refers you to other sections of this document. There are certain Oracle JRockit options for which there are no corresponding HotSpot JVM options. Also, some of the mapped HotSpot options aren't 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 options isn't recommended. Especially for performance-related options, the best practice is to start by only specifying the Java heap size and the garbage collector, such as CMS or G1. Any additional tuning for HotSpot, if necessary, must be done based on new benchmarking and profiling done with HotSpot. It isn't advised to assume that most, if any, JVM-level tuning decisions made for an Oracle JRockit configuration will also apply to a HotSpot configuration.

Table 6-1 -X Command-Line Options

Oracle JRockit	HotSpot	Notes
`-Xbootclasspath`	Same	*NA*
`-Xbootclasspath/a`	Same	*NA*
`-Xbootclasspath/p`	Same	*NA*
`-Xcheck:jni`	Same	*NA*
`-Xdebug`	Same	*NA*
`-Xgc`	*NA*	See Tuning Garbage Collection.
`-XgcPrio` (deprecated)	*NA*	See Garbage Collectors .
`-XlargePages`	`-XX:+UseLargePages`	*NA*
`-Xmanagement`	*NA*	NA
`-Xms`	Same	*NA*
`-Xmx`	Same	*NA*

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

Oracle JRockit	HotSpot	Notes
`-XnoClassGC` (deprecated)	Same	Don't use, except for troubleshooting.
`-XnoOpt`	*NA*	See Compilation Optimization .
`-Xns`	Same	*NA*
`-XpauseTarget`	`-XX:MaxGCPauseMillis=n`	See Garbage Collectors .
`-Xrs`	Same	*NA*
`-Xss`	Same	*NA*
`-XstrictFP`	*NA*	*NA*
`-Xverbose`	`-verbose`	See Logging .
`-Xverbosedecorations`	*NA*	See Logging.
`-XverboseLog`	*NA*	See Logging.
`-XverboseTimeStamp`	*NA*	See Logging.
`-Xverify`	Same	*NA*

Table 6-2 -XX Command-Line Options

Oracle JRockit	HotSpot	Notes 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 isn't 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>`	*NA*
`-XX:+|-CheckJNICalls`	`-Xcheck:jni`	*NA*
`-XX:+|-CheckStacks`	*NA*	*NA*

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

Oracle JRockit	HotSpot	Notes on HotSpot Options
`-XXcompaction`	*NA*	*NA*
`-XXcompactRatio` (deprecated)	*NA*	*NA*
`-XXcompactSetLimit` (deprecated)	*NA*	*NA*
`-XXcompactSetLimitPerObject` (deprecated)	*NA*	*NA*
`-XXcompressedRefs`	`-XX:-UseCompressedOops`	See Runtime Options.
`-XX:+|-CrashOnOutOfMemoryError`	Same	*NA*
`-XX:+|-DisableAttachMechanism`	Same	*NA*
`-XXdumpFullState`	*NA*	On the HotSpot side, there is an option `CreateMinidumpOnCrash` to enable the dumping of minidumps when fatal errors occur on the Windows platform.
`-XXdumpSize`	*NA*	*NA*
`-XX:ExceptionTraceFilter`	*NA*	*NA*
`-XX:+|-ExitOnOutOfMemoryError`	Same	*NA*
`-XX:ExitOnOutOfMemoryErrorExitCode`	*NA*	*NA*
`-XXexternalCompactRatio` (deprecated)	*NA*	*NA*
`-XX:+|-FailOverToOldVerifier`	Same	*NA*
`-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.
`-XX:FlightRecorderOptions`	Same	*NA*
`-XX:+|-FlightRecordingDumpOnUnhandledException`	*NA*	*NA*
`-XX:FlightRecordingDumpPath`	*NA*	*NA*

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

Oracle JRockit	HotSpot	Notes on HotSpot Options
`-XXfullSystemGC`	Related options: • `-XX:+DisableExplicitGC` • `-XX:+ExplicitGCInvokesConcurrent` • `-XX:+ExplicitGCInvokesConcurrentAndUnloadsClasses`	See Garbage Collectors .
`-XXgcThreads`	Related options: • `-XX:ParallelGCThreads=<threads>` • `-XX:ConcGCThreads=<threads>`	See Garbage Collectors.
`-XX:GCTimePercentage`	*NA*	*NA*
`-XX:GCTimeRatio`	*NA*	*NA*
`-XXgcTrigger`	Related options: • `-XX:CMSInitiatingOccupancyFraction=<percent>` • `-XX:CMSTriggerRatio=<percent>`	See Garbage Collectors .
`-XX:+|-HeapDiagnosticsOnOutOfMemoryError`	Can achieve the same by using `-XX:OnOutOfMemoryError=<command>`	Example: `java -XX:OnOutOfMemoryError="jmap -heap %p" JavaProgram`
`-XX:HeapDiagnosticsPath`	*NA*	*NA*
`-XX:+|-HeapDumpOnCtrlBreak`	*NA*	*NA*
`-XX:+|-HeapDumpOnOutOfMemoryError`	Same	*NA*
`-XX:HeapDumpPath`	Same	*NA*
`-XX:HeapDumpSegmentSize`	*NA*	*NA*
`-XXheapParts` (deprecated)	*NA*	*NA*
`-XXinternalCompactRatio` (deprecated)	*NA*	*NA*
`-XX:+|-JavaDebug`	*NA*	*NA*

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

Oracle JRockit	HotSpot	Notes on HotSpot Options
`-XXkeepAreaRatio`	`XX:SurvivorRatio=<ratio>`	Sets the ratio between the eden space size and the 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 the survivor space is resized according to the application behavior, starting with this initial value.
`-XXlargeObjectLimit` (deprecated)	*NA*	*NA*
`-XX:MaxCodeMemory`	`-XX:ReservedCodeCacheSize=<size>`	See Compilation Optimization .
`-XX:MaxDirectMemorySize`	Same	*NA*
`-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 Runtime options.
`-XX:MaxRecvBufferSize`	*NA*	*NA*
`-XXminBlockSize` (deprecated)	*NA*	*NA*
`-XXnoSystemGC`	Related options: • `-XX:+DisableExplicitGC` • `-XX:+ExplicitGCInvokesConcurrent` • `-XX:+ExplicitGCInvokesConcurrentAndUnloadsClasses`	See Garbage Collectors .
`-XX:OptThreads`	`-XX:CICompilerCount=threads`	See Compilation Optimization .

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

Oracle JRockit	HotSpot	Notes 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>`	*NA*
`-XX:+|-ReserveCodeMemory`	`-XX:ReservedCodeCacheSize=<size>`	See Compilation Optimization .
`-XX:SegmentedHeapDumpThreshold`	*NA*	*NA*
`-XXsetGC` (deprecated)	*NA*	*NA*
`-XX:+|-StrictFP`	*NA*	*NA*
`-XX:StartFlightRecording`	Same	*NA*
`-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`	*NA*	*NA*
`-XX:+|-UseAdaptiveFatSpin`	*NA*	*NA*
`-XX:+|-UseAllocPrefetch`	Related options: • `-XX:AllocateInstancePrefetchLines=<lines>` • `-XX:AllocatePrefetchDistance=<size>` • `-XX:AllocatePrefetchInstr=<instruction>` • `-XX:AllocatePrefetchLines=<lines>` • `-XX:AllocatePrefetchStepSize=<size>` • `-XX:AllocatePrefetchStyle=<style`	*NA*
`-XX:+|-UseCallProfiling`	`-XX:+UseTypeProfile`	*NA*
`-XX:+|-UseCfsAdaptedYield`	*NA*	*NA*

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

Oracle JRockit	HotSpot	Notes on HotSpot Options
`-XX:+|-UseClassGC`	`-Xnoclassgc`	Disables garbage collection (GC) of classes. This can save the 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 active.
`-XX:+|-UseCPoolGC`	*NA*	*NA*
`-XX:+|-UseFastTime`	*NA*	*NA*
`-XX:+|-UseFatSpin`	*NA*	*NA*
`-XX:+|-UseLargePagesFor[Heap|Code]`	• `-XX:+UseLargePages` • `-XX:+UseLargePagesInMetaspace`	See Runtime Options.
`-XX:+|-UseLazyUnlocking`	`-XX:+UseBiasedLocking`	See Runtime Options.
`-XX:+|-UseLockProfiling`	*NA*	*NA*
`-XX:+|-UseLowAddressForHeap`	*NA*	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 JDK 5. This option must not be used on JDK 6 or later versions.
`-XX:+|-UseThreadPriorities`	Same	On HotSpot, this option is enabled by default for the Windows platform. On JRockit, this option is disabled by default for the Windows platform.

Table 6-3 Diagnostic Commands

Oracle JRockit	HotSpot
`check_flightrecording`	`JFR.check`
`command_line`	`VM.command_line`
`dump_flightrecording`	`JFR.dump`
`exception_trace_filter`	*NA*
`force_crash`	*NA*

Table 6-3 (Cont.) Diagnostic Commands

Oracle JRockit	HotSpot
heap_diagnostics	GC.heap_info
help	help
hprofdump	GC.heap_dump
kill_management_server	ManagementAgent.stop
list_vmflags	VM.flags
lockprofile_print	*NA*
lockprofile_reset	*NA*
memleakserver	*NA*
print_class_summary	GC.class_stats
print_exceptions	*NA*
print_memusage	VM.native_memory
print_object_summary	GC.class_histogram
print_threads	Thread.print
print_utf8pool	VM.stringtable and VM.symboltable
print_vm_state	VM.info
runsystemgc	GC.run
set_filename	*NA*
start_flightrecording	JFR.start
start_management_server	ManagementAgent.start ManagementAgent.start_local
stop_flightrecording	JFR.stop
stop_management_server	ManagementAgent.stop
timestamp	*NA*
verbosity	*NA*
version	VM.version

jcmd Commands

The following are the list of jcmd commands:

- JFR.configure
- JFR.stop
- JFR.start
- JFR.dump
- JFR.check
- VM.log
- VM.native_memory
- VM.check_commercial_features

- `VM.unlock_commercial_features`
- `ManagementAgent.status`
- `ManagementAgent.stop`
- `ManagementAgent.start_local`
- `ManagementAgent.start`
- `Compiler.directives_clear`
- `Compiler.directives_remove`
- `Compiler.directives_add`
- `Compiler.directives_print`
- `VM.print_touched_methods`
- `Compiler.codecache`
- `Compiler.codelist`
- `Compiler.queue`
- `VM.classloader_stats`
- `Thread.print`
- `JVMTI.data_dump`
- `JVMTI.agent_load`
- `VM.stringtable`
- `VM.symboltable`
- `VM.class_hierarchy`
- `GC.class_stats`
- `GC.class_histogram`
- `GC.heap_dump`
- `GC.finalizer_info`
- `GC.heap_info`
- `GC.run_finalization`
- `GC.run`
- `VM.info`
- `VM.uptime`
- `VM.dynlibs`
- `VM.set_flag`
- `VM.flags`
- `VM.system_properties`
- `VM.command_line`
- `VM.version help`

For the complete list of commands, see jcmd Commands in the *Java Platform, Standard Edition Tools Reference* guide.

7

Common Migration Issues and Solutions

This topic describes some common issues that can occur while migrating from Oracle JRockit to HotSpot, along with their 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	Notes
Performance degradation after migrating to JDK 7. The 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` The default value on most of the platforms is 48 MB.	With HotSpot VM, it was observed that in some cases increasing the `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)	The `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 or synchronization after migrating to HotSpot, then disabling this option might provide some performance gains.

8
Troubleshooting Tools

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

Troubleshooting Tools Available in Java SE

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

Table 8-1 Tools

Java SE Troubleshooting Tools	Notes and Resources
Java Flight Recorder and Mission Control	See the following topics in *Java Platform, Standard Edition Troubleshooting Guide*: • 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
Serviceability Agent	See the article about the Serviceability Agent published in the Java Magazine dated July 2012: HotSpot's Hidden Treasure
JConsole	See Troubleshoot with JConsole in *Java Platform, Standard Edition Troubleshooting Guide*
jcmd command utility	See: • Troubleshoot with jcmd Utility in *Java Platform, Standard Edition Troubleshooting Guide* • jcmd in *Java Platform, Standard Edition Tools Reference*
JDK utilities	There are many useful utilities bundled with JDK. See the following topics in *Java Platform, Standard Edition Troubleshooting Guide*: • jdb • jinfo • jmap • jps • jstack • jstat • jrunscript • jsadebugd • jstatd

Table 8-1 (Cont.) Tools

Java SE Troubleshooting Tools	Notes and Resources
visualgc	See visualgc Tool in *Java Platform, Standard Edition Troubleshooting Guide*.
Native Memory Tracking Tool	See Native Memory Tracking in *Java Platform, Standard Edition Java Virtual Machine Guide*.