

# Java Platform, Standard Edition

## JRockit to HotSpot Migration Guide



Release 9  
E76287-05  
October 2017

The Oracle logo, consisting of a solid red square with the word "ORACLE" in white, uppercase, sans-serif font centered within it.

ORACLE®

E76287-05

Copyright © 1995, 2017, 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

	<b>Preface</b>	
	Audience	v
	Documentation Accessibility	v
	Related Documents	v
	Conventions	v
<b>1</b>	<b>Introduction</b>	
	Heap Sizing	1-1
<b>2</b>	<b>Garbage Collectors</b>	
	Tuning Garbage Collection	2-1
<b>3</b>	<b>Runtime</b>	
	Runtime Options	3-1
<b>4</b>	<b>Compilation Optimization</b>	
	Compiler Considerations	4-1
	Important HotSpot JIT Compiler Options	4-2
<b>5</b>	<b>Logging</b>	
	Verbose Logging	5-1
	HotSpot Logging Options	5-2
<b>6</b>	<b>Command-Line Options</b>	
	Mapping of Oracle JRockit to HotSpot Command-Line Options	6-1
	jcmm Commands	6-8

## 7 Common Migration Issues and Solutions

---

## 8 Troubleshooting Tools

---

Troubleshooting Tools Available in Java SE

8-1

# 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 9 and JDK 9. However, most of the information in this document can be applied to releases earlier than JDK 9.

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 [Oracle JDK 9 Documentation](#) for other JDK 9 guides.

## Conventions

The following text conventions are used in this document:

<b>Convention</b>	<b>Meaning</b>
<b>boldface</b>	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
<i>italic</i>	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
<code>-Xms</code>	Sets the initial and minimum size of the heap	Sets the initial and minimum size of the heap
<code>-Xmx</code>	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: <ul style="list-style-type: none"><li>• <code>-Xgc:throughput</code></li><li>• <code>-Xgc:genpar</code></li><li>• <code>-Xgc:singlepar</code></li><li>• <code>-Xgc:parallel</code></li></ul>	Throughput collector: <code>-XX:+UseParallelGC</code> - Use parallel compacting collector for both young and old generation  <code>-XX:+UseParallelGC -XX:-UseParallelOldGC</code> - 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 <code>-XX:ParallelGCThreads=n</code> .
Low latency collector set using any of the following options: <code>-Xgc:pausetime</code> <code>-Xgc:gencon</code> <code>-Xgc:singlecon</code>	<code>-XX:+UseG1GC</code> Or <code>-XX:+UseConcurrentMarkSweepGC</code>	The HotSpot offers a choice between the two mostly concurrent collectors: <ul style="list-style-type: none"><li>• 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</li><li>• 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</li></ul> 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:+UseLargePagesInMeta space.

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](#)
- [Important HotSpot JIT Compiler Options](#)

### 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 server (also known as C2).

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

HotSpot VM defaults to interpreting Java byte code. It compiles (JIT compilation) methods that runtime profiling determines to be "hot", that is, the methods that are executed for a predetermined number of times. JIT compilers 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 compiler 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 equivalent to the client VM 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, a startup that is even faster than with the client VM can be achieved, 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.

## 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>	<p>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.</p> <p>Like Oracle JRockit, HotSpot also offers ways to exclude methods from compilation or to disable specific optimizations on them.</p> <p>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.</p> <p>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.</p> <p>Equivalent HotSpot JVM options are:</p> <ul style="list-style-type: none"> <li>• -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</li> <li>• -XX:CompileCommandFile=&lt;filename&gt; 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.</li> <li>• -XX:CompileOnly=&lt;methods&gt; Sets the list of methods (separated by commas) to which compilation must be restricted.</li> <li>• -XX:CompileThreshold=&lt;invocations&gt; 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.</li> </ul>	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	Notes
-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. 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	<code>-verbose:class</code>	Displays information about the classes that are being loaded.
codegen	NA	NA
compaction	NA	NA
cpuinfo	NA	NA
exceptions	NA	NA
gc	<code>-verbose:gc</code>	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
-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 <code>-verbose:gc</code> with the time elapsed from the first GC event preceding each logged event. The <code>-Xloggc</code> option overrides the <code>-verbose:gc</code> , if both are given with the same <code>java</code> command.
-XX:LogFile=<path>	Sets the path and file name where the log data is written.
-XX:+PrintCommandLineFlags	Enables printing of the 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 <code>hotspot.log</code> in the current working directory. You can specify a different log file path and name using the <code>-XX:LogFile</code> option. The <code>-XX:+LogCompilation</code> option must be used together with the <code>-XX:UnlockDiagnosticVMOptions</code> option that unlocks diagnostic JVM options.
-XX:+PrintAssembly	Enables printing of assembly code resulting from JIT compilation of Java bytecode by using the external <code>disassembler.so</code> 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 <code>-XX:UnlockDiagnosticVMOptions</code> 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.

Table 5-2 (Cont.) Logging Options

HotSpot Logging Options	Notes
-XX:+PrintConcurrentLocks	Enables printing of <code>java.util.concurrent</code> locks after a Control+C event (SIGTERM). By default, this option is disabled.
-XX:+G1PrintHeapRegions	Enables the printing of information about the regions that are allocated and that 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 the time elapsed from the last pause (for example, a GC pause).
-XX: +PrintGCApplicationStoppedTime	Enables printing of the duration of the pause (for example, a GC pause) that lasted.
-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.
-Xlog	Enables the common logging system for all JVM components.

# 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 <a href="#">Tuning Garbage Collection</a> .
-XgcPrio (deprecated)	NA	See <a href="#">Garbage Collectors</a> .
-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 <a href="#">Compilation Optimization</a> .
-Xns	Same	NA
-XpauseTarget	-XX:MaxGCPauseMillis=n	See <a href="#">Garbage Collectors</a> .
-Xrs	Same	NA
-Xss	Same	NA
-XstrictFP	NA	NA
-Xverbose	-verbose	See <a href="#">Logging</a> .
-Xverboosedecorations	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: <ul style="list-style-type: none"> <li>• -XX:AllocateInstancePrefetchLines=&lt;lines&gt;</li> <li>• -XX:AllocatePrefetchDistance=&lt;size&gt;</li> <li>• -XX:AllocatePrefetchInstr=&lt;instruction&gt;</li> <li>• -XX:AllocatePrefetchLines=&lt;lines&gt;</li> <li>• -XX:AllocatePrefetchStepSize=&lt;size&gt;</li> <li>• -XX:AllocatePrefetchStyle=&lt;style&gt;</li> </ul>	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 <a href="#">Runtime Options</a> .
-XX:+ - CrashOnOutOfMemoryError	Same	NA
-XX:+ - DisableAttachMechanism	Same	NA
-XXdumpFullState	NA	On the HotSpot side, there is an option <code>CreateMinidumpOnCrash</code> 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 <code>-XX:+UnlockCommercialFeatures</code> 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: <ul style="list-style-type: none"> <li>-XX:+DisableExplicitGC</li> <li>-XX:+ExplicitGCInvokesConcurrent</li> <li>-XX:+ExplicitGCInvokesConcurrentAndUnloadsClasses</li> </ul>	See <a href="#">Garbage Collectors</a> .
-XXgcThreads	Related options: <ul style="list-style-type: none"> <li>-XX:ParallelGCThreads=&lt;threads&gt;</li> <li>-XX:ConcGCThreads=&lt;threads&gt;</li> </ul>	See <a href="#">Garbage Collectors</a> .
-XX:GCTimePercentage	NA	NA
-XX:GCTimeRatio	NA	NA
-XXgcTrigger	Related options: <ul style="list-style-type: none"> <li>-XX:CMSInitiatingOccupancyFraction=&lt;percent&gt;</li> <li>-XX:CMSTriggerRatio=&lt;percent&gt;</li> </ul>	See <a href="#">Garbage Collectors</a> .
-XX:+ -HeapDiagnosticsOnOutOfMemoryError	Can achieve the same by using -XX:OnOutOfMemoryError=<command>	Example: <pre>java -XX:OnOutOfMemoryError="jmap -heap %p" JavaProgram</pre>
-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 <a href="#">Compilation Optimization</a> .
-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: <ul style="list-style-type: none"> <li>• -XX:+DisableExplicitGC</li> <li>• -XX:+ExplicitGCInvokesConcurrent</li> <li>• -XX:+ExplicitGCInvokesConcurrentAndUnloadsClasses</li> </ul>	See <a href="#">Garbage Collectors</a> .
-XX:OptThreads	-XX:CICompilerCount=threads	See <a href="#">Compilation Optimization</a> .

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

Oracle JRockit	HotSpot	Notes on HotSpot Options
-XX:+ -RedoAllocPrefetch	Related options: <ul style="list-style-type: none"> <li>- XX:AllocateInstancePrefetchLines=&lt;lines&gt;</li> <li>-XX:AllocatePrefetchDistance=&lt;size&gt;</li> <li>-XX:AllocatePrefetchInstr=&lt;instruction&gt;</li> <li>-XX:AllocatePrefetchLines=&lt;lines&gt;</li> <li>-XX:AllocatePrefetchStepSize=&lt;size&gt;</li> <li>-XX:AllocatePrefetchStyle=&lt;style&gt;</li> </ul>	NA
-XX:+ -ReserveCodeMemory	-XX:ReservedCodeCacheSize=<size>	See <a href="#">Compilation Optimization</a> .
- 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: <ul style="list-style-type: none"> <li>- XX:AllocateInstancePrefetchLines=&lt;lines&gt;</li> <li>-XX:AllocatePrefetchDistance=&lt;size&gt;</li> <li>-XX:AllocatePrefetchInstr=&lt;instruction&gt;</li> <li>-XX:AllocatePrefetchLines=&lt;lines&gt;</li> <li>-XX:AllocatePrefetchStepSize=&lt;size&gt;</li> <li>-XX:AllocatePrefetchStyle=&lt;style&gt;</li> </ul>	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 <code>Xnoclassgc</code> 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]	<ul style="list-style-type: none"> <li>• -XX:+UseLargePages</li> <li>• -XX:+UseLargePagesInMetaspace</li> </ul>	See <a href="#">Runtime Options</a> .
-XX:+ -UseLazyUnlocking	-XX:+UseBiasedLocking	See <a href="#">Runtime Options</a> .
-XX:+ -UseLockProfiling	NA	NA
-XX:+ -UseLowAddressForHeap	NA	No direct corresponding option available in HotSpot but the low heap base can be specified explicitly using <code>HeapBaseMinAddress</code> 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, refer to 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
<p>Performance degradation after migrating to JDK 7.</p> <p>The issue was resolved with the use of -XX:ReservedCodeCacheSize=1g</p>	<p>-XX:+ReserveCodeMemory</p> <p>Default values:</p> <ul style="list-style-type: none"> <li>When you use -XX:+UseLargePagesForCode: 64 MB</li> <li>When you use -XX:-UseLargePagesForCode: 1024 MB</li> </ul>	<p>-</p> <p>XX:ReservedCodeCacheSize</p> <p>The default value on most of the platforms is 48 MB.</p>	<p>With HotSpot VM, it was observed that in some cases increasing the ReservedCodeCacheSize value, for example, -XX:ReservedCodeCacheSize=1g, improves the performance significantly.</p>
<p>Increased locking/unlocking events observed after switching to HotSpot.</p> <p>Disabling UseBiasedLocking helped improve the overall performance.</p>	<p>-XX:-UseLazyUnlocking (to disable)</p>	<p>-XX:-UseBiasedLocking (to disable)</p>	<p>The UseBiasedLocking option improves the performance of uncontended synchronization. This option is enabled by default.</p> <p>However, if the application has high contended synchronization, then disabling UseBiasedLocking benefits the performance.</p> <p>If you face performance issues due to locking or synchronization after migrating to HotSpot, then disabling this option might provide some performance gains.</p>

# 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 <i>Java Platform, Standard Edition Troubleshooting Guide</i> : <ul style="list-style-type: none"><li>• Java Mission Control</li><li>• What are Java Flight Recordings</li><li>• How to produce a Flight Recording</li><li>• Inspect a Flight Recording</li><li>• Debug a Memory Leak Using Java Flight Recorder</li></ul>
Serviceability Agent	See the article about the Serviceability Agent published in the Java Magazine dated July 2012: <a href="#">HotSpot's Hidden Treasure</a>
JConsole	See Troubleshoot with JConsole in <i>Java Platform, Standard Edition Troubleshooting Guide</i>
jcmm command utility	See: <ul style="list-style-type: none"><li>• Troubleshoot with jcmm Utility in <i>Java Platform, Standard Edition Troubleshooting Guide</i></li><li>• jcmm in <i>Java Platform, Standard Edition Tools Reference</i></li></ul>
JDK utilities	There are many useful utilities bundled with JDK. See the following topics in <i>Java Platform, Standard Edition Troubleshooting Guide</i> : <ul style="list-style-type: none"><li>• jdb</li><li>• jinfo</li><li>• jmap</li><li>• jps</li><li>• jstack</li><li>• jstat</li><li>• jrunscript</li><li>• jsadebugd</li><li>• jstatd</li></ul>

**Table 8-1 (Cont.) Tools**

<b>Java SE Troubleshooting Tools</b>	
<b>Notes and Resources</b>	
visualgc	See visualgc Tool in <i>Java Platform, Standard Edition Troubleshooting Guide</i> .
Native Memory Tracking Tool	See Native Memory Tracking in <i>Java Platform, Standard Edition Java Virtual Machine Guide</i> .