BEA WebLogic JRockit™ SDK

Tuning WebLogic JRockit with WebLogic Server on Linux

Version 8.1
July 2003

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Contents

3. Tuning WebLogic JRockit JVM with WebLogic Server on Linux

CHAPTER 1

Introduction

Welcome to Tuning WebLogic JRockit with WebLogic Server on Linux. This document is specifically designed for WebLogic JRockit users who are running the JVM with BEA WebLogic Server 8.1 on the Linux operating system. Much of the information in this guide is similar to that in *Tuning WebLogic JRockit 8.1 JVM*, however this document contains information and examples specific to Linux users.

This Introduction further describes the contents of this guide and answers some basic questions about tuning WebLogic JRockit. It includes information on the following subjects:

Why is Tuning Necessary?

Although WebLogic JRockit JVM automatically adapts to its underlying hardware and to the application running on it, the JVM cannot know everything about your system. For example, how much memory do you want the JVM to use? Or, how long should the maximum pauses be, to work best within the tolerances of your application? To instruct WebLogic JRockit on how to handle this critical processing functions, you can configure—or tune—many aspects of your JVM's performance by setting appropriate configuration options at startup.

How Do You Tune a JVM?

While WebLogic JRockit uses many of the standard start-up options available for other JVMs—such as logging options like `-version`, which tells WebLogic JRockit to display its product version number and `-verbose`, which tells JRockit to display verbose output—actual tuning of the JVM requires manipulating WebLogic JRockit's two main subsystems: the memory management system (often called garbage collection), and the thread system.

Tuning these subsystems requires setting non-standard, or `-X`, options at startup. This guide documents the different startup options and tells you what you need to know about these subsystems to be able to tune them successfully. By using these options and following the recommendations suggested in this guide, you can ensure that you application performs optimally. You should note that, on Linux, these options do not differ from those used with other O/Ss, such as Windows.

Migrating Applications to WebLogic JRockit

WebLogic JRockit is the default JVM shipped with BEA WebLogic Server. Although there are other JVMs available on the market today that you can use to develop Java applications, BEA Systems recommends that you use WebLogic JRockit JVM as the production JVM for any application deployed on WebLogic Server. Migrating to WebLogic JRockit with WebLogic Server on Linux describes basic environment changes necessary to migrate to WebLogic JRockit JVM from Sun Microsystems HotSpot JVM or any other third-party JVM.

What Linux Operating Systems Does WebLogic JRockit Support?

This version on WebLogic JRockit SDK supports these versions of the Linux operating system:

- For 32-bit processors (Intel Pentium II—or comparable—and higher):
  - Red Hat Enterprise Linux AS / ES / WS 2.1
  - SuSE SLES 8.0 (United Linux 1.0)
- For 64-bit processors (Intel Itanium 2 or higher):
  - Red Hat Linux Advanced Server for Itanium Processor 2.1
  - Red Hat Linux Advanced Workstation for Itanium Processor 2.1

For complete platform support details, please refer to:

`http://edocs.bea.com/wljrockit/docs81/certif.html`

Installing WebLogic JRockit with WLS on Linux

Installing WebLogic JRockit to run with WebLogic Server on a Linux machine is handled no differently that installing it in any other configuration. You can either install the JVM as part of the BEA WebLogic Platform product suite (which includes WLS) or you can install it as a standalone application. With either of these installation methods, you also have the option of installing it in either a graphic mode or from the console (command line mode), should you not be using a GUI.

To install WebLogic JRockit as part of WebLogic Platform, please refer to *Installing WebLogic Platform*, specifically:

- Installing WebLogic Platform Using Graphical-Mode Installation
- Installing WebLogic Platform Using Console-Mode Installation

To install WebLogic JRockit as a standalone application, please refer to *Installing WebLogic JRockit 8.1 SDK*, specifically:

- Installing WebLogic JRockit SDK
- Installing and Uninstalling WebLogic JRockit SDK in the Console Mode

CHAPTER 2

Running WebLogic JRockit JVM with WebLogic Server on Linux

This section describes how to start up and configure the BEA WebLogic JRockit JVM. It also provides some background on selecting and running both a thread system and a memory management system (called a "garbage collector"). This section includes information the following subjects:

Starting BEA WebLogic JRockit

You can start BEA WebLogic JRockit either from the command line or from the WebLogic Server's Node Manager. This section describes both methods.

Starting BEA WebLogic JRockit JVM from the Command Line

Before starting WebLogic JRockit JVM, ensure that you have the following directory set in your PATH environment variable:

`<jrockit-install-directory>/bin`

(where `<jrockit-install-directory>` is the folder where you installed WebLogic JRockit, such as c:`/bea/jrockit81_141_02/`.)

To start the JVM, at the command line enter on of the following:

```
java
```

and any tuning or configuration and tuning options you want to use; for example, you might start WebLogic JRockit JVM with the following command string:

```
java -verbose -Xgc:gencopy -Xms:64 -Xmx:512 -Xns:64 -Xnopt myClass
```

Here, the standard configuration parameter `-Verbose` requests verbose output from the system and the non-standard option `-Xgc:gencopy` sets the generational copying memory management system (the "garbage collector"). The tuning parameters `-Xms:64` and `-Xmx:512` set the minimum (`-Xms`) and maximum (`-Xmx`) heap sizes, while the tuning parameter `-Xns:64` set the size of the young generation (the "nursery"; required for generational collectors). `myClass` identifies the class that contains the `main()` method and is required whenever you start WebLogic JRockit.

All of the options in this example—along with others—are described in Configuring WebLogic JRockit JVM by Using Command Line Options, below, or in Tuning WebLogic JRockit JVM with WebLogic Server on Linux.

Note: You can also start JRockit from the command by specifying a fully-qualified path to the file; for example, enter:

<jrockit-install-directory>/bin/java

Starting BEA WebLogic JRockit from the Node Manager

You can also start WebLogic JRockit JVM from the WLS Node Manager. To do this, access the Node Manager and go to the Remote Start Page. In the Java Home field, enter the fully-qualifying path, as suggested in the example above. For more information on using the Node Manager, please see Overview of Node Manager in *Configuring and Managing WebLogic Server*.

Configuring WebLogic JRockit JVM by Using Command Line Options

Configure WebLogic JRockit JVM by using a number of standard and non-standard command line options that you enter at startup. These options work with WebLogic JRockit on all supported operating systems; none are specific to Linux.

If you start WebLogic JRockit JVM from the Remote Start page of the WLS Node Manager, simply enter the options, with the appropriate arguments, in the Arguments field on the Remote

Start page. For more information on using the Node Manager,please see Overview of Node Manager in *Configuring and Managing WebLogic Server*. This section describes:

- Standard Command Line Options for:
  - Presenting General Information Options
  - Setting Logging Options
- Non-standard Command Line Options for:
  - Setting Behavioral Options
  - Displaying Logging Information

Standard Command Line Options

Standard command line configuration options work the same regardless of the JVM; in other words, these options work the same whether you are running WebLogic JRockit JVM, Sun Microsystem's HotSpot JVM, or any other third-party JVM.

Presenting General Information Options

The following standard command line options set general information about WebLogic JRockit JVM:

- `-classpath` <directories and zips/jars separated by `:`>

  Tells the VM where to look for classes and resources. Alternately, you can use the option `-cp` to represent `-classpath`; for example:

  `-cp` <directories and zips/jars separated by `:`>

- `-D<name>[=<value>]`

  Tells the VM to set a Java system property. These can be read by a Java program, using the methods in `java.lang.System`.

Setting Logging Options

The following options determine if the system will provide messages to the operator and what the form and content of those messages should be.

- `-version`

  Tells JRockit to display its product version number and then exit.

- `-showversion`

Tells the VM to display its product version number and then continue.

- `-verbose[:<`components separated by `,>]`

  Tells JRockit to display verbose output. This option is used mainly for debugging purposes and causes a lot of output to the console. Supported components are `memory`, `load` and `codegen`. If no component is given, JRockit will display verbose information on everything.

- `-help`

  Tells the VM to display a short help message.

- `-X`

  Tells the VM to display a short help message on the extended options.

Non-standard Command Line Options

The non-standard, or `-X`, command line options are options that are exclusive to WebLogic JRockit JVM that change the behavior of WebLogic JRockit JVM to better suit the needs of different Java applications. Non-standard options are used extensively with thread system and memory management. These options normally won't work on other JVMs (conversely, the non-standard options used by other JVMs normally won't work with WebLogic JRockit).

Warning: Since these options are non-standard, they are subject to change at any time.

Setting Behavioral Options

The following non-standard options define general WebLogic JRockit JVM behavior:

- `-Xnoopt`

  Tells the VM not to optimize code.

- `-Xverify`

  Tells the VM to do complete bytecode verification.

Displaying Logging Information

`-Xverbose`

`-Xverbose` causes WebLogic JRockit to print to the screen specific information about the system. The information displayed depends upon the parameter specified with the option; for example, specifying the parameter `cpuinfo` displays information about your CPU and indicates

whether or not the JVM can determine if hyper threading is enabled. The valid parameters for `-Xverbose` are: .

- codegen — The names of each method that is being compiled.
- `cpuinfo` – Any interesting information about your CPUs.
- load — The names of each loaded class.
- memory; gc — Information about the memory management system, including:
  - Start time of collection (seconds since JVM start)
  - End time of collection (seconds since JVM start)
  - Memory used by objects before collection (KB)
  - Memory used by objects after collection (KB)
  - Size of heap after collection (KB)
  - Total time of collection (seconds or milliseconds)
  - Total pause time during collection (milliseconds)
  - The information displayed by -Xverbose:memory or -Xverbose:gc will vary depending upon the type of garbage collector you are using.
- opt — Information about all methods that get optimized.

For descriptions and examples of the information these parameters display, please refer to Table 3-1 in Starting and Configuring WebLogic JRockit JVM (found in *Using WebLogic JRockit 8.1 SDK*).

Using a Thread System

The thread system allows WebLogic JRockit JVM to take optimal advantage of the underlying operating system. WebLogic JRockit JVM supports two types of thread systems:

- **Native Threads**, which maps Java threads directly to the operating system threads, taking advantage of the operating system's thread scheduling and load balancing policies. Native Threads is the default thread system for WebLogic JRockit JVM.
- **Thin Threads**, wherein multiple Java threads are run on a singe operating system thread. This allows WebLogic JRockit JVM to optimize thread scheduling, thread switching, and thread synchronization, while using less memory.

Warning: Thin threads is experimental functionality in this version of WebLogic JRockit SDK and is not recommended for general use. This feature is subject to change without notice.

For more information on native and thin threads and on how to select a thread system that best meets your needs, please refer to Choosing the Thread System in *Using* WebLogic JRockit *8.1 SDK*.

To start a thread system, include one of the options listed in Table 2-1 when you start WebLogic JRockit JVM:

Table 2-1 Thread System Implementation Options

To Use...	Use this Option...
Native Threads	`-Xnativethreads` This option is the default.
Thin Threads	`-Xthinthreads` This option is not available on IA64.

Using a Memory Management System (Garbage Collection)

WebLogic JRockit JVM manages memory by employing four different garbage collectors. These collectors work during runtime to clear the memory heap of expired objects, or "garbage." The four garbage collectors are:

- **Generational Copy**, which divides the memory into two areas called "generations." Instead of allocating objects in one single space and garbage collecting that whole space when it gets full, most of the objects are allocated in the "young generation," called the nursery.
- **Single-Spaced Concurrent**, one of two types of concurrent collectors, which does its work in parallel with ordinary processing; that is, it does not stop all Java threads to do the complete garbage collection. This is designed to support garbage collection without disruption and to improve multiprocessor garbage collection performance.
- **Generational Concurrent** is the second type of concurrent collector WebLogic JRockit employs. Although very similar to a single-spaced concurrent collector, a generational concurrent garbage collector does its actual object allocation in a "nursery," reducing the need to do collection of the entire heap so often.

- **Parallel** garbage collection, which stops all Java threads and uses all CPUs to perform a complete garbage collection of the entire heap.

For information and tips on selecting a garbage collector, please refer to Choosing a Garbage Collection Method in *Using WebLogic JRockit 8.1 SDK*.

Starting a Garbage Collector

To start a garbage collector, simply include at the command line the `-Xgc` option and the type of collector you want to use, as shown in Table 2-2.

Table 2-2 Garbage Collector Implementation Options

To select...	Enter...
Generational Copying	`-Xgc:gencopy`
Single Spaced Concurrent	`-Xgc:singlecon`
Generational Concurrent	`-Xgc:gencon`
Parallel	`-Xgc:parallel`

When started, JRockit will run with the specified garbage collector.

Default

If the garbage collector has not been set and the maximum heap size (set by using `-Xmx` or by using the default, as described in Setting the Maximum Heap Size) is less than 128 MB, the default garbage collector is generational copying (`-Xgc:gencopy`); otherwise the default is parallel (`-Xgc:parallel`).

Printing a Comprehensive Report

`-Xgcreport`

`-Xgcreport` causes WebLogic JRockit JVM to print a comprehensive garbage collection report at program completion. The option `-Xgcpause` causes the VM to print a line each time Java threads are stopped for garbage collection. Combining the two is a very good way of examining the memory behavior of your application.

CHAPTER 3

Tuning WebLogic JRockit JVM with WebLogic Server on Linux

This section describes how to tune BEA WebLogic JRockit JVM by using its command line options. It also provides tips that help Linux users to exploit the tuning options to achieve optimal performance by the VM.

This section includes information on the following subjects:

- Basic JRockit Tuning
- Tuning Tips and Techniques
- Optimizing WebLogic JRockit for Use with WebLogic Server

Basic JRockit Tuning

To provide the optimal out-of-the-box experience, WebLogic JRockit JVM comes with default values that adapt automatically to the specific platform on which you are running WebLogic JRockit JVM. Depending upon the application you are running or the configuration of your machine, you might want to tune the JVM to take fullest advantage of its capabilities.

Tuning WebLogic JRockit JVM is accomplished by using non-standard—or `-x`—start-up options to change the default settings for elements such as heap size and thread stack size (for an example, see Starting BEA WebLogic JRockit). `-x` options are exclusive to WebLogic JRockit JVM.

This section descibes how to use these options to tune WebLogic JRockit. It includes information on the following subjects:

- Setting the Initial Heap Size

Setting the Initial Heap Size

`-Xms<size>`

`-Xms` sets the initial size of the heap. For this, we recommend that you set it to the same size as the maximum heap size; for example:

-java -Xgc:gencon -Xmx:64m **-Xms:64m** myClass

Default

The defalt initial heap size is 16 MB if maximum `-Xmx` is less than 128 MB; otherwise it is 25% of *available* physical memory up to, but not exceeding, 64 MB.

Setting the Maximum Heap Size

`-Xmx:<size>`

`-Xmx` sets the maximum size of the heap. Use the following guidelines to determine this value:

- On IA32 the maximum possible heap size is about 1.8GB (which is the largest contigous address space the O/S will give a process).
- Because IA64 macines have a larger address space, the 1.8GB limit does not apply.
- Typically, for any platform you don't want to use a larger maximum heap size setting than 75% of the *available* physical memory. This is because you need to leave some memory space available for internal usage in the JVM.

Note: If you encounter OutOfMemory errors, you should increase the maximum heap size according to the preceding guidelines.

Default

If you are running `-Xgc:gencopy`, the default maximum heap size is the lesser of 75% of the physical memory *and* 400 MB; otherwise it is the lesser of 75% of physical memory *and* 1536 MB.

Setting -Xmx to Avoid Fragmentation

Be aware that fragmentation can occur if you rely on the default maximum heap size (described above). Fragmentation can cause paging, which can degrade system performance. This is because WebLogic JRockit grows the heap aggressively when fragmentation occurs, potentially out-stripping the physical memory available. To avoid this situation, you should override the default and manually set `-Xmx` to 75% of the *available* physical memory, up to 1.8 GB. Note that if you have other processes running that require large amounts of the physical memory, you will have to account for their expense when calculating how much memory is available.

Setting the Size of the Nursery

`-Xns:<size>`

`-Xns` sets the size of the young generation (nursery) in generational concurrent and generational copying garbage collectors (these are the only collectors that use nurseries). Optimally, you should try to make the nursery as large as possible while still keeping the garbage collection-pause times acceptably low. This is particularly important if you are creating a lot of temporary objects.

Note: To display pause times, include the option `-Xgcpause` when you start WebLogic JRockit JVM.

The maximum size of a nursery may not exceed 25% of the total heap size if you're using `gencon` and 15% of the total heap size if you're using `gencopy`.

Default

If the nursery size (`-Xns`) has not been set the default size depends on the type of garbage collector and the number of CPUs:

- For the generational copying garbage collector (-Xgc:`gencopy`) the default nursery size is 320 KB per CPU; for example, the default for a ten CPU system using `gencopy` would be 3200 KB (3.2 GB).

- For the generational concurrent garbage collector (-Xgc:`gencon`) the default nursery size is 10 MB per CPU; for example, the default for a ten CPU system using `gencon` would be 100 MB.

Setting Thread Local Object Allocation

`-Xallocationtype:<global|local>`

`-Xallocationtype` sets the type of thread allocation, `global` or `local` as described in Table 3-1.

Table 3-1 `-Xallocationtype` Parameters

Use this type...	When...
`local`	The maximum heap size is large (more then 128 MB) or if the number of threads used by the application is low (less than several hundred).
`global`	The maximum heap size is very small (less then 128 MB) or if the number of threads used by the application is very high (several hundred). This is because every thread-local area consumes a fixed amount of memory (approximately 2 kilobytes). If the number of threads is very high or the heap size very small when using thread-local allocation the potential waste of space could cause excess fragmentation of the heap. This leads to more frequent garbage collections and may cause the application to run out of memory prematurely.

Default

If the allocation type (`-Xallocationtype`) is not set, the default is `global` for the generational copying (`-Xgc:gencopy`) garbage collector and `local` for all others (`singlecon`, `gencon`, and `parallel`).

Defining When a Memory will be Cleared

`-Xcleartype:<gc|local|alloc>`

`-Xcleartype` defines when the memory occupied by an object that has been garbage collected will be cleared. Specified parameters dictate when the memory will be cleared, as described in Table 3-2.

Table 3-2 `-Xcleartype` **Parameters**

Use this parameter...	To clear space...
`gc`	During the garbage collection
`local` This option is available only if the `-Xallocationtype` is set to local.	When a thread-local area is allocated
`alloc` This option is currently not available on IA64 systems. Additionally, it is the preferred option if the objects allocated are very large (1 to 2 kilobytes).	When that space is allocated for a new object

Note: The preferred options are `local` or `alloc`.

Default

If `-Xcleartype` is not set the default is `alloc` on IA32 systems and `gc` on IA64 systems.

Setting the Thread Stack Size

`-Xss<size>[k|K]`

`-Xss<size>[k|K]` sets the thread stack size, in kilobytes `[k|K]`.

In addition to setting the thread stack size, if the number of threads is high, you can reduce heap fragmentation by setting `-Xallocationtype:global`, as suggested in Setting the Thread Stack Size.

Default

If the thread stack size has not been set the default value depends on the threading system and the platform you are running on. When using thin threads the minimum thread stack size is 8 kilobytes and the default is 64 kilobytes. When using native threads the minimum thread stack

size is 16 kilobytes. For Linux, the default thread stack size when using native threads is 128 kilobytes.

If `-Xss` is set below the minimum value, thread stack size will default to the minimum value automatically.

Tuning Tips and Techniques

When you install WebLogic JRockit 8.1 SDK, the VM includes a number of default start-up options that ensure a satisfactory out-of-the-box experience; however, often, these options might not provide your application with the optimal performance you should experience with WebLogic JRockit 8.1 JVM. Therefore, WebLogic JRockit 8.1 JVM comes with numerous alternative options and algorithms to suit different applications. This section describes some of these alternative options and some basic tuning techniques you can use at startup. It includes information on the following subjects:

- Determine What You Want to Tune For
- Set the Heap Size
- Tune for High Responsiveness
- Tune for High Performance
- Analyze Garbage Collection and Pause Times
- Modify Threading Options When Using a Large Number of Threads
- Check for Task and Open File Allowance

Note: The tuning tips in this section create behaviors and performance that don't differ noticeably from those on other operating systems.

Determine What You Want to Tune For

Before you start WebLogic JRockit 8.1 JVM, you need to determine these two factors:

- How much of your machine memory do you want WebLogic JRockit 8.1 JVM to use?
- What do you want from WebLogic JRockit 8.1 JVM, the highest possible responsiveness or the highest possible performance?

Once you've answered these questions, use the information provided below to tune WebLogic JRockit 8.1 JVM to achieve those goals.

Set the Heap Size

Generally, you want to set the maximum heap size as high as possible, but not so high that it causes page-faults for the application or for some other application on the same computer. Set it to something less than the amount of memory in the machine. If you have multiple applications running on the computer at the same time the value could be much lower. It is recommend that you set the initial heap size (`-Xms`) the same size as the maximum heap size. Generally, you don't want the heap to exceed 75% of the available memory. However, some machines can have up to 4 GB of phsyical memory and when the heap exceeds about 1.8 GB, failure can occur. Therefore, you should limit the heap size to something less than 1.8 GB.

For specific guidelines for setting the heap size, please refer to Setting the Initial Heap Size and Setting the Maximum Heap Size.

Tune for High Responsiveness

If you want the highest responsiveness from your application and guarantee minimal pause times, set the following options at startup:

- Accept the default Generational Concurrent garbage collector (`-Xcg:gencon`).
- Set the initial (`-Xms`) and maximum (`-Xmx`) heap sizes, as described in Set the Heap Size. Since you're using a generational concurrent garbage collector, the heap size will not cause longer pauses.
- Set the size of the nursery (`-Xns`).

  If you are creating a lot of temporary objects you should have a large nursery. Larger nurseries usually result in slightly longer pauses, so, while you should try to make the nursery as large as possible, don't make it so large that pause times are unacceptable. You can see the nursery pause times in WebLogic JRockit JVM by starting the JVM with `-Xgcpause`.

Tune for High Performance

If you want the highest possible performance WebLogic JRockit 8.1 JVM can provide, set these tuning options at startup:

- Select the Parallel garbage collector (`-Xgc:parallel`). A parallel garbage collector doesn't use a nursery, so you won't need to set `-Xns`.
- Set the largest initial (`-Xms`) and maximum (`-Xmx`) heap sizes that your machine can tolerate, as described in Set the Heap Size.

Analyze Garbage Collection and Pause Times

Analyzing garbage collection and pause times together will give you a good idea of how well your application is performing while running with WebLogic JRockit JVM.

- Use the option `-Xgcreport` to generate and end-of-run report that shows the garbage collection statistics. You can use this report to determine if you're using the most effective garbage collector for your application.
- Use the option `-Xverbose:memory` (see Displaying Logging Information) to display the pause times for every garbage collection during a run. Note that this option is used mainly for debugging purposes and causes a lot of output to the console.

Modify Threading Options When Using a Large Number of Threads

If you are running with more than 100 threads and you want to improve system performance, try the following:

- Switch to thin threads by using the option `-Xthinthreads`. Thin threads are particularly effective if you're running your application on a Linux machine (please read the **Warning** about thin threads, in Using a Thread System, before implementing them).
- Turn off thread local allocation by using the option `-Xallocationtype:global`. Every thread-local area consumes a fixed amount of memory (approximately 2 kilobytes). If the number of threads is very high and you are using thread-local allocation, the potential waste of space could cause excess fragmentation of the heap. This leads to more frequent garbage collections and may cause the application to run out of memory prematurely. Using thread global allocation will result in less fragmentation, although actual allocation will be slower.

Check for Task and Open File Allowance

You should check the contents of `/proc/sys/kernel/threads-max` and `/proc/sys/fs/file-max`. The former tells you how many tasks the kernel will allow, which limits how many threads you can create at the Java level. The latter shows how many open files are allowed.

Optimizing WebLogic JRockit for Use with WebLogic Server

This section describes the tuning requirements for optimizing WebLogic JRockit on BEA WebLogoc Server.

Configuring JRockit for WebLogic Server

To use the WebLogic JRockit JVM instead of the Sun JVM, you need to increases the initial heap size to 64 MB (`-Xms:64m`)and the maximum heap size to at least 200 MB (`-Xmx:200m`). In addition, the following options are automatically set:

- `-Xnativethreads` is set as the default thread system setting.
- `-Xallocationtype:local` is set as the default thread allocation setting.

These settings are normally used for initial development. If you want to improve WebLogic JRockit performance, do any of the following:

- Increase the heap initial and maximum size (`-Xms` and `-Xmx`).
- Change the garbage collector to single space concurrent (`-Xgc:singlecon`) or parallel (`-Xgc:parallel`). Note that if you select parallel as your garbage collector, the `-Xns` setting will have no affect on processing (see Setting the Size of the Nursery).

Setting Options by Using the Node Manager

If you started the server or cluster of servers with the Node Manager and specified an absolute pathname to WebLogic JRockit JVM's top-level directory in the Java Home field on the Node Manager's Remote Start page, you can set any option from this page, too. Simply enter the option and any arguments in the Arguments field.

For more information on using the Node Manager, please refer to the Overview of Node Manager in *Configuring and Managing WebLogic Server*.

Monitoring WebLogic JRockit JVM from WebLogic Server

If you run WebLogic Server with WebLogic JRockit JVM, you can use the WebLogic Server Administration Console to view runtime data about the VM and the memory and processors on the computer hosting it. For instructions on monitoring WebLogic JRockit JVM from WebLogic Server, please see Monitoring WebLogic JRockit JVM from WebLogic Server in *Using WebLogic JRockit 8.1 SDK*.

Running JRockit with Thin Threads on WebLogic Server

Warning: Thin threads is experimental functionality in this version of JRockit, and is not recommended for general use. This feature is subject to change without notice.

The JRockit high performance thread system (Thin Threads, `-Xthinthreads`) and the native I/O system of WebLogic Server are incompatible as they both use asynchronous I/O. To avoid problems you must disable the native I/O system of WebLogic Server when running JRockit using thin threads. The native I/O is disabled automatically in WebLogic Server if JRockit is using thin threads, even if it is turned on in the corresponding WebLogic Server configuration file. In their respective default setups, WebLogic JRockit JVM does not use thin threads and WebLogic Server uses native I/O.

Switching to WebLogic JRockit JVM in WebLogic Server

When you switch to WebLogic JRockit JVM in WebLogic Server, any changes to the VM and start-up setting, should be handled by the WLS Configuration Wizard. Additionally, if any installation-wide scripts must be updated due to the switch, these will also be handled by the WLS Configuration Wizard. You will also need to restart any servers that are currently running.

For complete details on switching to WebLogic JRockit JVM from another JVM, please refer to Migrating to WebLogic JRockit with WebLogic Server on Linux.

CHAPTER 4

Migrating to WebLogic JRockit with WebLogic Server on Linux

BEA WebLogic JRockit JVM is the default JVM shipped with BEA WebLogic Server. Although there are other JVMs available on the market today that you can use to develop Java applications, BEA Systems recommends that you use WebLogic JRockit JVM as the production JVM for any application deployed on WebLogic Server.

This section describes basic environment changes necessary to migrate to WebLogic JRockit JVM from Sun Microsystems HotSpot JVM or any other third-party JVM. It includes information on the following subjects:

- Additional Migration Information
- Required Changes
- Enabling Core Dumps
- Migration Restrictions
- Migration Support

Additional Migration Information

The migration tips in this chapter are very similar to those for other operating systems, such as Windows. For more comprehensive migration information than is presented here, please refer to Migrating to WebLogic JRockit in the *Migration Guide*. Along with detailed migration information, this document also includes a discussion of best coding practices for applications designed to run on WebLogic JRockit and extensive troubleshooting information, both of which are equally germane to Linux users.

Required Changes

To migrate from HotSpot (or any third-party JVM) to WebLogic JRockit JVM, you need to make the following changes to the files specified.

Changes to Environment Variables

You need to change the environment variables in `<WEBLOGIC_HOME>\common\bin\commEnv.sh as described here:`

- Set the `JAVA_HOME` environmental variable to the appropriate path; for example:

```
# Set up JAVA HOME
   JAVA_HOME="C:\bea\jdk141_02"
```

- Set the `JAVA_VENDOR` environmental variable to `BEA`; for example:

```
# Set up JAVA VENDOR, possible values are
# BEA, HP, IBM, Sun ...
   JAVA_VENDOR=BEA
```

Changes to Start-up Scripts

If you are using start-up scripts, remove any Sun- (or other JVM provider) specific options from the start command line (for example, `-hotspot`). If possible, replace them with WebLogic JRockit-specific options; for example, `-jrockit`. Other flags that might need to be changed include `MEM_ARGS` and `JAVA_VM`.

Changes to config.xml

Change `config.xml` to point the default compiler setting(s) to the WebLogic JRockit `javac` compiler; for example:

```
<Server Name="examplesServer" ListenPort="7001" ConsoleInputEnabled=
   "false" JavaCompiler="javac" SocketReaderTimeoutMaxMillis=
   "10"OMEnabled="true">
```

Enabling Core Dumps

If you are using Red Hat Enterprise Linux AS/ES/WS 2.1 (32-bit) and want to ensure that a `core/javacore` file is created in the working directory in the event WebLogic JRockit crashes, you need to enable core dumps. To do this, set the `ulimit -c` value to something greater than zero, but no greater than a value your filesystem can accommodate; for example, `ulimit -c`

`10000000`. These values are measured in blocks, with each block equaling one kilobyte. You can set the `ulimit` value either from the command line, in the `*.profile` file, or in a shell script.

Migration Restrictions

Migration is available only for Intel-based Windows and Linux systems. For a list of supported platforms, please refer to:

`http://edocs.bea.com/wljrockit/docs81/certif.html`

Migration Support

Should you experience any problems or find any bugs with an application you have migrated to WebLogic JRockit 8.1, please report it to **`support@bea.com`**. You should provide as much information as possible about the problem, for example:

- Hardware
- Operating system and its version
- The program you are attempting to migrate
- Stack dumps (if any)
- A small code example that will reproduce the error
- Copies of any `*.dump` files