BEA WebLogic JRockit™ SDK

Tuning WebLogic JRockit 8.1 JVM

Version 8.1 Service Pack 2
December 2003

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Contents

Introduction

Tuning WebLogic JRockit JVM

Basic Tuning Tips and Techniques

Index

CHAPTER 1

Introduction

BEA WebLogic JRockit JVM automatically adapts to its underlying hardware and to the application running on it. You might wonder, why would anyone need to tune the JVM? The answer is that there are some things WebLogic JRockit JVM cannot know about your system. For example, how much memory do you want the JVM to use? You probably don't want the JVM to use most of the available memory. Or, how long should the maximum pauses be, to work best within the tolerances of your application?

WebLogic JRockit JVM has a number of non-standard startup parameters, called -X options, that allow you to better tune the JVM for your specific application. In WebLogic JRockit JVM there are two main subsystems that can be optimized separately using different startup parameters: the memory management system (including the garbage collectors), and the thread system. This guide documents the different startup parameters and what you need to know about these subsystems to be able to tune them efficiently. You will find that the memory management system is the subsystem that gives you the most tuning opportunities. By tuning these parameters you will likely find the best performance improvements for your application.

CHAPTER 2

Tuning WebLogic JRockit JVM

Have you ever seen strange pauses in your application that you haven't been able to explain? Have you seen one or all CPUs pegged on 100% utilization and all the others on 0% and still very few transactions in your system? If you answered yes to either of these two questions, your application might have been suffering from the effects of a poorly performing garbage collector. Some fairly simple tuning of the memory management system can improve performance dramatically for many applications.

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. Tuning WebLogic JRockit JVM is accomplished by using non-standard—or `-x`—command line options that you enter at startup. `-x` options are exclusive to WebLogic JRockit JVM. Use them to set the behavior of WebLogic JRockit JVM to better suit the needs of your Java applications.

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

- Setting Heap Size Parameters
- Defining When a Memory Space will be Cleared
- Setting the Type of Thread Allocation
- Setting the Thread Stack Size
- Memory Requirements and Garbage Collection Types

Note: If WebLogic JRockit behaves in some unexpected way, please consult the WebLogic JRockit FAQ. If that doesn't solve your problem, please send an e-mail to **support@bea.com**

Setting Heap Size Parameters

System performance is greatly influenced by the size of the Java heap available to the JVM. This section describes the commandline options you can use to define the initial and maximum heap sizes and the size of any nursery that might be required by generational garbage collectors.

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 default 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 1.8 GB (which is the largest contiguous address space the O/S will give a process). This is helpful because, while the O/S limitation for the maximum memory size per a process is 2 GB on IA-32 platform, you cannot fully use 2 GB for JVM heap. This is because other purpose spaces are included in that 2 GB; for example the memory used by WebLogic JRockit internally, native library space, and so on. Be aware that 1.8 GB is just a suggested heap size; you have to tune it according to your application.
- Because IA64 machines have a larger address space, the 1.8 GB 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.

Default

Generally, the default maximum heap size (`-Xmx`) is 75% of the physical memory in the machine. However, when running WebLogic JRockit with a small initial heap (that is, less than about 32MB) the default maximum heap size will depend on the initial heap size. The default maximum heap size is `-Xms`[2] (in megabytes), *up to* 75% of the physical memory; for example, if `-Xms` is 8MB, the default maximum heap size will be 8MB[2], or 64MB; if `-Xms` is 128MB, the default maximum heap size would be 128[2], or 16384MB (16 GB). Be aware that, if the machine has less physical memory than the value of `-Xms`[2], the default maximum heap will be restricted to 75% of the *physical memory*.

Note: These figures are subject to any platform limitations that determine how much contiguous memory a process can allocate.

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.

Encountering OutOfMemory Errors

Under certain circumstances, the JVM will not have enough memory to continue processing and throw an OutOfMemory error. This section describes some of the most common reason for those errors and suggests some ways to work around them.

Can't Find Memory to Allocate an Object on the Heap

Occasionally, WebLogic JRockit won't be able to find memory to allocate a Java object on the heap. If this is happening, enable `-Xverbose:gc`. If the java heap appears full, this is probably the cause. We recommend that you increase the java heap size.

OOM Errors When Compiling Large Methods

WebLogic JRockit allocates memory outside the java heap for internal structures used for thread handling, garbage collection, code generation and so on. If the java heap is taking too much of the total process memory, the JVM could run out of memory during such activities as code

generation of a large method. It will then throw an OutOfMemory error. When this happens, we suggest that you decrease the java heap size. You might also enable `-Xverbose:codegen` to detect the methods that take long time to generate. These are probably the methods consuming the most memory.

Can't Commit Reserved Memory

WebLogic JRockit might not be able to commit reserved memory needed by the system because the disk where the swap file resides is full, preventing the swap file from growing. This usually happens when the process size is not too large, but many other processes are running concurrently and the disk on which the swap file resides is almost full. We suggest that you release memory on the disk or shut down some of the other processes.

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 MB).
- 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.

Working Around Limits to Expanding Heap Size

When memory is completely allocated, WebLogic JRockit JVM cannot expand its heap size if swap space is not available. This often occurs when several applications are running simultaneously. If it happens, WebLogic JRockit will exit after throwing an OutOfMemory error. To remedy this condition, increase the available swap space by doing either or both of the following:

- Allocate more of your disk for virtual memory.
- Limit the number of applications that can run simultaneously.

You might also avoid this problem by setting the command-line flags `-Xmx` and `-Xms` to the same value. This will prevent WebLogic JRockit JVM from trying to expand the heap. If you attempt this workaround, be aware that the heap cannot expand to the size specified by `-Xmx` if the necessary physical and virtual memory is not available.

Defining When a Memory Space will be Cleared

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

`-Xcleartype` defines when the memory space occupied by an object that has been garbage collected will be cleared. When clearing is actually performed is specified by the selected parameter, as described in Table 2-1.

Table 2-1 `-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

The preferable options are either `alloc` or `local`.

Note these additional conditions:

- The `-Xcleartype:local` option is available only if the `-Xallocationtype` is set to `local`.
- The `alloc` parameter is currently not available on IA64 systems.

Default

If the clear type is not set the default is `alloc` on IA32 systems and `gc` on IA64 systems.

Note: The option `alloc` is currently not available on IA64 systems.

Setting the Type of Thread Allocation

`-Xallocationtype:<global|local>`

`-Xallocationtype` sets the type of thread allocation, `global` or `local` as described in Table 2-2.

Table 2-2 `-Xallocationtype` Parameters

Use this type...	When...
`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.
`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).

Default

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

Setting the Thread Stack Size

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

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

In addition to setting the thread stack size, if the number of threads is high, you should use `-Xallocationtype:global`, as suggested in Setting the Type of Thread Allocation to reduce heap fragmentation.

Minimum Thread Size

Minimum thread size is determined as follows:

- Thin threads—the minimum thread stack size is 8 kilobytes and the default is 64 kilobytes.
- Native threads—the minimum thread stack size is 16 kilobytes.

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

Default

If the thread stack size has not been set the default value depends on the threading system and the platform on which WebLogic JRockit is running:

32-bit Default

On either Windows or Linux IA32 machines, the default thread stack size values for native threads are:

- Win32: 64 kB
- Linux32: 128 kB

64-bit Default

On either Windows or Linux IA64 machines, the default thread stack size values for native threads are:

- Win64: 320 kB
- Linux64: 1 MB

Memory Requirements and Garbage Collection Types

You should be aware that the memory requirements described in this section are suggestions based upon out-of-the-box testing. Your actual requirements might vary, particularly as they apply to the garbage collection method you select and the application you are running. Please refer to the tuning guidelines in Basic Tuning Tips and Techniques for instructions and tips on optimally setting your memory requirements.

CHAPTER 3

Basic Tuning Tips and Techniques

When you install WebLogic JRockit 8.1 JVM, the VM includes a host 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
- Setting the Heap Size
- Tuning for High Responsiveness
- Tuning for High Performance
- Other Tuning Tips
- Analyzing and Improving Your Application

Note: The tuning settings discussed in this section refer to standard and non-standard tuning options which are not thoroughly described in the present context. For more information on these options, refer to Command Line Options by Name

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.

Setting 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.

Tuning for High Responsiveness

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

- Select the Generational Concurrent garbage collector. Since this is the default garbage collector, you don't actually need to set anything for garbage collection.
- Set the initial (`-Xms`) and maximum (`-Xmx`) heap sizes, as described in Setting the Heap Size. Since you're using a generational concurred 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`.

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

Other Tuning Tips

This section describes two other practices you can employ to improve WebLogic JRockit JVM performance.

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` 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 particulalry effective if you're running your application on a Linux machine.

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

- 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.

Analyzing and Improving Your Application

This section describes how you can improve application performance by uncovering "hotpaths," or bottlenecks in processing, and either working around those hotpaths or eliminating them completely.

Generally, analyzing and improving your application is a four-step process:

1. Find the hotpaths
2. Prioritize them
3. Fix the most important hotpaths
4. Repeat steps 1 through 3 until application performance is satisfactory

Step 1: Find the Hotpaths

Finding Hotpaths is a two-step process:

- Find the Bottleneck Methods
- Cluster the Bottleneck Methods Together into Hotpaths

Find the Bottleneck Methods

As their name implies, bottleneck methods are those methods that require excessive time and processing resources to execute. These bottlenecks can greatly affect system performance and need to be identified. To find bottleneck methods, do the following:

- Use the Intel VTune profiling tool with JRockit to analyze performance. VTune uses features on the processor to gather information about which code the processor is currently executing. This is done after a fixed number of either clock ticks (wall clock time) or after a fixed number of instructions retired (actual instructions executed in the processor). VTune then uses this data together with symbol information from Java code to present information about where the application spends most of its time.
- Use the Java Virtual Machine Profiling Interface (JVMPI) by setting the option `-Xjvmpi:allocs=off,monitors=off,entryexit=off`.

JVMPI is a two-way function call interface between the Java virtual machine and an in-process profiler agent. On one hand, the VM notifies the profiler agent of various events, corresponding to, for example, heap allocation, thread start, and so on. Concurrently, the profiler agent issues controls and requests for more information through the JVMPI.

For a list of recommended `-Xjvmpi` settings, please refer to Table 4.1 in Profiling and Debugging with WebLogic JRockit

Cluster the Bottleneck Methods Together into Hotpaths

To cluster the bottleneck methods together into hotpaths, do the following:

1. Run your favorite profiler, such as OptimizeIt or JProbe.
2. Review the call-traces produced by it.
3. Combine these call-traces with the bottleneck methods discovered in Find the Bottleneck Methods, above. This combination of bottleneck methods and call-traces will identify your hotpaths.

Step2: Prioritize the Hotpaths

To prioritize the hotpaths, do the following:

1. Sum all of the time spent in each hotpath to compute a total hotpath time.
2. Remove all individual hotpaths that represent less than a prescribed percentage of the total hotpath time; for example, a good initial percentage might be 5%. This is the hotpath threshold.
3. Ignore any hotpath that falls below the hotpath threshold. Any hotpath time above the threshold should be optimized.

Step 3: Fix the Hotpath

You need to rely on your own judgement and knowledge of the application to fix hotpaths. Once you've identified and prioritized the hotpaths, look at each one and decide if the code is really needed or if you can make some simple changes, perhaps to the coding or to an algorithm, to avoid it or eliminate it as a hotpath. If you determine that you cannot remove the hotpath, what can you do to make it faster? Rewrite the code so it's more efficient?

Also, are you sure that anything you do will actually improve performance. Any optimization you attempt should at least double performance of the hotpath or your efforts might be wasted. For

example, a performance increase of 5% or a hotpath that takes only 5% of the time is only going to improve performance .25%.

Step 4: Repeat Steps 1-3

Continue repeating the optimization process until you attain the desired system performance.

APPENDIX A

Command Line Options by Name

This appendix lists the valid WebLogic JRockit JVM command line options in alphabetical order, by name.

Note: Options that begin with `-X` are non-standard options (called "`-X` options") and are subject to change at any time.

Table A-1 Command Line Options by Name

Option	Description
`-classpath` `-cp`	Tells WebLogic JRockit JVM where to look for classes and resources.
`-D`	Tells WebLogic JRockit JVM to set a Java system property. These can be read by a Java program, using the methods in `java.lang.System`.
`-help`	Tells WebLogic JRockit JVM to display a short help message.
`-showversion`	Tells WebLogic JRockit JVM to display its product version number and then continue with startup.
`-verbose` `-Xverbose`	Tells WebLogic JRockit JVM to display verbose output.
`-version`	Tells WebLogic JRockit JVM to display its product version number and then exit.
`-X`	Tells WebLogic JRockit JVM to display a short help message on the extended options.

Table A-1 Command Line Options by Name

Option	Description
`-Xallotype` `-Xallocationtype`	Sets the type of allocation. Parameters: • `global`; global allocation • `local`; thread local allocation For details on these options, please refer to Table 2-2 in Tuning the WebLogic JRockit JVM.
`-Xbootclasspath`	Sets the search path for bootstrap classes and resources. Specify the names of the directories, `.zip`, and `.jar` files, separated by ";" (Windows) or ":" (Linux).
`-Xcleartype`	Defines when the memory space occupied by an object that has been garbage collected will be cleared. The parameters listed below determine when the space is cleared. Parameters: • `gc`; cleared during garbage collection • `local`; cleared when a thread-local area is allocated • `alloc`; cleared when that space is allocated for a new object For details on these options, please refer to Table 2-1 in Tuning the WebLogic JRockit JVM. Notes: • The `-Xcleartype:local` option is available only if the `-Xallocationtype` is set to local. • The `alloc` option is not available on IA64 systems.
`-Xgc`	Deploys the specified garbage collector. Parameters: • `gencopy`; Generational Copying • `singlecon`; Single Spaced Concurrent • `gencon`; Generational Concurrent • `parallel`; Parallel The default is `gencopy` if `-Xmx` is less than 128MB; otherwise, it's `gencon`.
`-Xgcpause`	Prints pause times caused by the garbage collector.

Table A-1 Command Line Options by Name

Option	Description
`-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.
`-Xjvmpi`	Enables and disables groups of JVMPI events when running `JVMPI`. The events are: • `entryexit` (default is ON) • `allocs` (default ON) • `monitors` (default (ON) • `arenasdelete` (default OFF) **Note:** JVMPI is an experimental feature in the Java 2 SDK and is not yet a standard profiling interface.
`-Xmanagement`	Enables the management server in the VM, which needs to be started before the Management Console can connect to WebLogic JRockit JVM.
`-Xms`	Sets the initial size of the heap. You should set the initial heap size (`-Xms`) to the same size as the maximum heap size. The default is 16 MB if maximum heap size is limited to less than 128 MB, otherwise 25% of available physical memory, but not exceeding 64 MB. This value can be specified in kilobytes (`K`,`k`), megabytes (`M`,`m`), or gigabytes (`G`,`g`).
`-Xmx`	Sets the maximum size of the heap. You should set this value 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. The default is the lesser of 75% of physical memory and 400 MB when running `gencopy`; when running another garbage collector, the default is the lesser of 75% of physical memory and 1536 MB. This value can be specified in kilobytes (`K`,`k`), megabytes (`M`,`m`), or gigabytes (`G`,`g`).

Table A-1 Command Line Options by Name

Option	Description
`-Xnativethreads`	Enables the Native Threads system. This option is the default, therefore you only need to specify it when you want to change the method from thin threads.
`-Xnoclassgc`	Disables class garbage collection.
`-Xnohup`	When specified, tells WebLogic JRockit JVM not to watch for or process `CTRL_LOGOFF_EVENT` or `SIGHUP` events.
`-Xnoopt`	Tells WebLogic JRockit JVM not to optimize code.
`-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). If you are creating a lot of temporary objects you should have a large nursery. The default is default is 10 MB per CPU with a `gencon` garbage collector and 320 KB per CPU with `gencopy`. This value can be specified in kilobytes (`K`,`k`), megabytes (`M`,`m`), or gigabytes (`G`,`g`).
`-Xss`	Sets the thread stack size; can be specified in kilobytes (`K`,`k`), megabytes (`M`,`m`), or gigabytes (`G`,`g`).
`-Xthinthreads`	Implements the WebLogic JRockit High Performance Threading System (Thin Threads). This option is not available on IA64. **Warning:** Thin threads is experimental functionality in this version of WebLogic JRockit, and is not recommended for general use. This feature is subject to change without notice.

Table A-1 Command Line Options by Name

Option	Description
`-Xverbose`	Causes WebLogic JRockit to print to the screen, upon startup, specific information about the system. The information displayed depends upon the parameter specified with the one of these options: • `codegen` • `cpuinfo` • `gc` • `load` • `memory` • `Opt` For a description of these options, please refer to Table 3-1 in Starting and Configuring WebLogic JRockit.
`-Xverify`	Tells WebLogic JRockit JVM to do complete bytecode verification.

Index

default values 2-1