DZone Refcardz

CONTENTS INCLUDE:

- Configuration
- JMX
- Available Plugins
- Java Memory Types
- JVM Settings
- and More!

Java Profiling with VisualVM

X-Ray Vision for Dramatic Performance Gains

By Mick Knutson

ABOUT VISUALVM

VisualVM is a visual tool integrating several command-line JDK tools and lightweight profiling capabilities. Designed for both production and development time use, it further enhances the capability of monitoring and performance analysis for the Java SE platform. http://visualvm.java.net/

CONFIGURATION

Repositories

Windows 7: C:\Users\[userId]\AppData\Roaming\.visualvm\[version]\repository\
oS X: /Users/[userId]/Library/Application Support/visualvm/[version]/repository/

User Directory

Windows 7: C:\Users\[userId]\AppData\Roaming\.visualvm\[version]\
oS X: /Users/[userId]/Library/Application Support/visualvm/[version]/

Message logs

Windows 7: C:\Users\[userId]\AppData\Roaming\.visualvm\[version]\var\log\message.log
oS X: /Users/[userId]/Library/Application Support/visualvm/[version]/var/log/message.log

JMX

JMX connections

Default connection:

```
<hostname>:<port>
```

Detailed connection:

```
service:jmx:<protocol>:<sap>
```

Detailed connection example:

```
service:jmx:rmi://<hostname>:<port>/jndi/rmi://<hostname>:<port>/jmxrmi
```

Container specific configuration

Configuration for exposing JMX.

Glassfish administration:

JVM options can be set via Glassfish web administration page: http://localhost:4848/common/javaConfig/serverJvmOptions.jsf?configName=server-config

Glassfish configuration file:

%GLASSFISH_HOME%\glassfish\domains\[domain]\config\domain.xml configuration file:

```
<configs> <config name="server-config">
  ... <java-config ...> ...
    <jvm-options><JMX configuration parameters></jvm-options>
  ...
```

Tomcat:

```
export CATALINA_OPTS="<JMX configuration parameters>"
```

Jetty standalone:

```
java "<JMX configuration parameters>" -jar start.jar etc/jetty-jmx.xml
```

Maven:

```
export MAVEN_OPTS="<JMX configuration parameters>"
```

Container specific configuration parameters

Name	Parameter
Access file	-Dcom.sun.management.jmxremote.access.file=<access file>
Authenticate	-Dcom.sun.management.jmxremote.authenticate=<true / false>
Debug	-Djavax.net.debug=<level>
JMX port	-Dcom.sun.management.jmxremote.port=<PortNumber>
JMX remote enabled	-Dcom.sun.management.jmxremote
* Keystore file	-Djavax.net.ssl.keyStore=<keystore file>
* Keystore password	-Djavax.net.ssl.keyStorePassword=<password>
Management level	-Djavax.management.level=<level>
Password file	-Dcom.sun.management.jmxremote.password file=<password file>
RMI server hostname	-Djava.rmi.server.hostname=<public.ip.address>
SSL enabled	-Dcom.sun.management.jmxremote.ssl=<true / false>

Name	Parameter
* Truststore file	-Djavax.net.ssl.trustStore=<truststore file>
* Truststore password	-Djavax.net.ssl.trustStorePassword=<password>

* Located in VisualVM options inside the security plugin.

AVAILABLE PLUGINS

Stable Plugins

Name	Description
MBean Browser	MBeans Browser plugin provides in general the same functionality as MBeans Browser in JConsole JDK tool: shows MBeans of an application, displays values, operations and notifications. In VisualVM, the browser will be further improved to deliver better usability and support for the latest JMX features.
Visual GC Plugin	Visual GC attaches to an application and collects and graphically displays garbage collection, class loader, and HotSpot compiler performance data.
Tracer	Framework and GUI for detailed monitoring and analyzing Java applications. Using various types of probes, the Tracer gathers metrics from an application and displays the data in a timeline. The data are displayed both graphically and in a table and can be exported to common formats for further processing in external tools.
Thread Inspector	Enables analyzing stack trace(s) of one or more selected threads directly without requiring you to take and open full thread dumps. This is extremely useful for quick and easy analyzing of various threading problems.
BTrace	BTrace is a dynamic tracing tool for Java. With this plugin you can create, deploy and save BTrace tracing scripts directly from the VisualVM.
VisualVM Extensions	The intent of this module is to add support for additional functionality (such as new JDKs, JVMs, HotSpot versions, etc.) not supported by the VisualVM core modules at the time VisualVM was released. It's always a good idea to get this plugin for a fresh VisualVM installation.
Security	The GUI for setting the keystore, truststore, protocols and ciphers for SSL/TLS connections in VisualVM. Using the plugin is equivalent to setting appropriate system properties javax.net.ssl.* and javax.rmi.ssl.client.*
Buffer Monitor	Monitors usage of direct buffers created by ByteBuffer.allocateDirect and mapped buffers created by FileChannel.map. Note that the buffers monitoring requires the monitored application to run JDK 7 starting from Build 36.
Kill Applications	Kills monitored applications that became unresponsive.
System Tray	Minimize/restore running the VisualVM instance into/from system tray. Not supported on Mac OS X (does nothing).
JVM Capabilities	Displays capabilities of monitored application's JVM.
Java ME Profiler Snapshot Viewer	Opens Java ME SDK profiler file in VisualVM.
JConsole plugins container	Provides support for using existing JConsole plugins inside VisualVM.

Tracer probes

Name	Description
Monitor Probes	For monitoring CPU & GC activity, heap & permgen usage, number of loaded classes and application threads.
JVM Probes	For monitoring various virtual machine internals, such as I/O metrics, GC metrics and HotSpot utilization.
Jvmstat Probes	For visualizing the metrics exported by the monitored JVM as jvmstat counters (including sun.gc.jvmstat, sun.perfdata.jvmstat, sun.threads.jvmstat, etc.).
Swing Probes	Provide detailed information about AWT and Swing GUI performance in terms of paints count, layout times and events utilization.
JavaFX Probes	For monitoring the performance of various logical parts of JavaFX applications.
DTrace Probes	Provides low-level system metrics, including JVM overhead, utilization of each CPU or syscalls numbers. These probes are available only for the Solaris/ OpenSolaris OS.

JConsole Plugins

Name	Description
JTop	A plugin for monitoring per-thread CPU usage and state.
Top Threads	Plugin for monitoring per-thread CPU usage and state. http://lsd.luminis.nl/top-threads-plugin-for-jconsole/
Hibernate	Monitors Hibernate via its JMX exports. The plugin displays graphs and details on queries, entities, collections and cache efficiency. http://hibernate-jcons.sourceforge.net/

Plugins in development

Name	Description
Glassfish	Provides additional information for GlassFish servers, e.g., thenumber of active sessions, processed transactions etc. It's also able to monitor each deployed web application separately.
OQL Syntax Support	Enhanced editor for OQL Console in HeapWalker providing syntax coloring and basic code completion.

Third-Party Plugins

Name	Description
Thread Dump Analyzer (TDA)	The Thread Dump Analyzer (TDA) for Java is a small GUI for analyzing Thread Dumps and Heap Information generated by the Sun Java VM. It provides statistics about thread dumps, gives information about locked monitors, waiting threads and much more. http://java.net/projects/tda
OSGI	Enables basic management of OSGi platforms via JMX. For detailed description and installation instructions visit the plugin page.
GCViewer	A new garbage collection monitoring plugin for VisualVM allows for investigating crucial GC metrics in greater detail and higher resolution. This is recommended especially for monitoring latency-constrained Java applications.

Hot Tip

GCViewer only works with OpenJDK and Hotspot by exploiting JVMStat API to access built-in HotSpot counters. For some reason the new parallel GC (-XX:+UseParNewGC) does not expose all the counters as the parallel scavenging collector does (-XX:+UseParallelGC), so only the former GC Pauses and Promoted vs Survived charts will work.

JAVA MEMORY TYPES

When a Java application is started, the Java process pre-allocates a given block of memory from the underlying operating system that is dedicated to the Java process. Addressable Java memory consists of a heap, permanent generation, and other memory spaces. The other memory space includes variables for JNI, Stack space, and the running Java Virtual Machine (JVM). A Java application can store variables in either the stack or heap spaces, depending upon the type and scope of the variable being stored.

Java stack

The Java stack space consists of local variables, method calls, arguments, reference variables, intermediate computations, and return values, if any, corresponding to the method invoked. Primitive data type variables such as int, long, float and double are also stored in the stack space.

Every thread, including the main thread and daemon threads, gets its own stack space but will share the same heap space.

The memory-for-stack space does not need to be contiguous and follows a Last In, First out (LIFO) algorithm.

Java permanent generation

The Java permanent generation space, or permGen, consists of reflective data of the virtual machine such as Class and Method objects. When new Class or Method types are created at runtime, new space is allocated in the permGen space for these types.

Java heap

The Java heap spaces consist of instances of Objects, instance variables, and instance-level references to Objects.

There are three types of heaps:

- Eden Space (young generation): pool from which memory is initially allocated for most objects.
- Survivor Spaces (young generation): two pools containing objects that have survived GC of Eden space.
- Tenured Generation (old generation): pool containing objects that have existed for some time in the survivor spaces.

JVM SETTINGS

Java JVM settings and details for tuning, tracing and debugging a Java Virtual Machine (JVM)

Categories of Java HotSpot VM Options

Standard options recognized by the Java HotSpot VM are described on the Java Application Launcher reference pages for Windows, Solaris and Linux.

- Options that begin with -X are non-standard (not guaranteed to be supported on all VM implementations), and are subject to change without notice in subsequent releases of the JDK.
- Options that are specified with -XX are unstable and are subject to change without notice.

Useful -XX Options

Some options may vary per architecture/OS/JVM version. Platforms with a differing default value are listed in the description.

- Boolean options are turned on with -XX:+<option> and turned off with -XX:-<option>.
- Numeric options are set with -XX:<option>=<number>. Numbers can include 'm' or 'M' for megabytes, 'k' or 'K' for kilobytes, and 'g' or 'G' for gigabytes (for example, 32k is the same as 32768).
- String options are set with -XX:<option>=<string>, and are usually used to specify a file, a path, or a list of commands.

Flags marked as manageable are dynamically writeable through the JDK management interface (com.sun.management.HotSpotDiagnosticMXBean API) and also through VisualVM.

The options below are loosely grouped into categories.

- Java heap options are used to specify initial and max heap size and thread stack size while running Java programs.
- Behavioral options change the basic behavior of the VM.
- Garbage First (G1) Garbage Collection Options
- Performance tuning options are knobs that can be used to tune VM performance.
- Debugging options generally enable tracing, printing, or output of VM information.

Java heap options

Option & default value	Description
-Xms<n[k|m]>	set initial Java heap size
-Xmx<n[k|m]>	set maximum Java heap size
-Xss<n[k|m]>	set java thread stack size

Behavioral Options

Option & default value	Description
[The (original) copying collector (Enabled by default)]	When this collector kicks in, all application threads are stopped, and the copying collection precedes using one thread (which means only one CPU, even if on a multi-CPU machine). This is known as a stop-the-world collection because,basically, the JVM pauses everything else until the collection is completed.
-XX:-AllowUserSignalHandlers	Do not complain if the application installs signal handlers. (Relevant to Solaris and Linux only.)
-XX:AltStackSize=<n[k|m]> [default = 16384]	Alternate signal stack size (in Kbytes). (Relevant to Solaris only, removed from 5.0.)
-XX:-DisableExplicitGC	Disable calls to System.gc(), JVM still performs garbage collection when necessary.
-XX:+FailOverToOldVerifier	Fail over to old verifier when the new type checker fails. (Introduced in 6.)
-XX:+HandlePromotionFailure	The youngest generation collection does not require a guarantee of full promotion of all live objects. (Introduced in 1.4.2 update 11) [5.0 and earlier: false.]
-XX:+MaxFDLimit	Bump the number of file descriptors to max. (Relevant to Solaris only.)
-XX:PreBlockSpin=<n> [default = 10]	Spin count variable for use with -XX:+UseSpinning. Controls the maximum spin iterations allowed before entering operating system thread synchronization code. (Introduced in 1.4.2.)
-XX:-RelaxAccessControlCheck	Relax the access control checks in the verifier. (Introduced in 6.)
-XX:+ScavengeBeforeFullGC	Do young generation GC prior to a full GC. (Introduced in 1.4.1.)
-XX:+UseAltSigs	Use alternate signals instead of SIGUSR1 and SIGUSR2 for VM internal signals. (Introduced in 1.3.1 update 9, 1.4.1. Relevant to Solaris only.)
-XX:+UseBoundThreads	Bind user level threads to kernel threads. (Relevant to Solaris only.)
-XX:-UseConcMarkSweepGC	Use concurrent mark-sweep collection for the old generation. (Introduced in 1.4.1)
-XX:+UseGCOverheadLimit	Use a policy that limits the proportion of the VM's time that is spent in GC before an OutOfMemory error is thrown. (Introduced in 6.)
-XX:+UseLWPSynchronization	Use LWP-based instead of thread based synchronization. (Introduced in 1.4.0. Relevant to Solaris only.)
-XX:-UseParallelGC	Use parallel garbage collection for scavenges. (Introduced in 1.4.1)

Option & default value	Description
-XX:-UseParallelOldGC	Use parallel garbage collection for the full collections. Enabling this option automatically sets -XX:+UseParallelGC. (Introduced in 5.0 update 6.)
-XX:-UseSerialGC	Use serial garbage collection. (Introduced in 5.0.)
-XX:-UseSpinning	Enable naive spinning on Java monitor before entering operating system thread synchronization code. (Relevant to 1.4.2 and 5.0 only.) [1.4.2, multi-processor Windows platforms: true]
-XX:+UseTLAB	Use thread-local object allocation (Introduced in 1.4.0, known as UseTLE prior to that.) [1.4.2 and earlier, x86 or with -client: false]
-XX:+UseSplitVerifier	Use the new type checker with StackMapTable attributes. (Introduced in 5.0.)[5.0: false]
-XX:+UseThreadPriorities	Use native thread priorities.
-XX:+UseVMInterruptibleIO	Thread interrupt before or with EINTR for I/O operations results in OS_INTRPT. (Introduced in 6. Relevant to Solaris only.)
-XX:-UseParNewGC	The parallel copying collector, like the original copying collector, this is a stop-the-world collector.
-Xincgc	The incremental collector uses a "train" algorithm to collect small portions of the old generation at a time.

Garbage First (G1) Garbage Collection Options

Option & default value	Description
-XX:+UseG1GC	Use the Garbage First (G1) Collector
-XX:MaxGCPauseMillis=<n>	Sets a target for the maximum GC pause time. This is a soft goal, and the JVM will make its best effort to achieve it.
-XX:InitiatingHeapOccupancyPercent=<n> [default = 45] [A value of 0 denotes 'do constant GC cycles'.]	Percentage of the (entire) heap occupancy to start a concurrent GC cycle. It is used by GCs that trigger a concurrent GC cycle based on the occupancy of the entire heap, not just one of the generations (e.g., G1).
-XX:NewRatio=<n> [default = 2]	Ratio of new/old generation sizes.
-XX:SurvivorRatio=<n> [default = 8]	Ratio of eden/survivor space size.
-XX:MaxTenuringThreshold=<n> [default = 15]	Maximum value for tenuring threshold.
-XX:ParallelGCThreads=<n> [The default value varies with the platform on which the JVM is running.]	Sets the number of threads used during parallel phases of the garbage collectors.
-XX:ConcGCThreads=<n>	Number of threads concurrent garbage collectors will use. The default value varies with the platform on which the JVM is running.
-XX:G1ReservePercent=<n> [default = 10]	Sets the amount of heap that is reserved as a false ceiling to reduce the possibility of promotion failure.
-XX:G1HeapRegionSize=<n[k|m]> [min = 1mb, max = 32mb] [The default value of this parameter is determined ergonomically based upon heap size.]	With G1 the Java heap is subdivided into uniformly sized regions. This sets the size of the individual sub-divisions.
-XX:+AggressiveHeap	Aggressively tune the parameters of its tuning algorithm based on using all the resources of the operating system on which the JVM is running.

Performance Options

Option & default value	Description
-XX:+AggressiveOpts	Turn on point performance compiler optimizations that are expected to be default in upcoming releases. (Introduced in 5.0 update 6.)
-XX:CompileThreshold=<n> [default = 10000] [-client: 1,500]	Number of method invocations/ branches before compiling.
-XX:LargePageSizeInBytes=<n[k|m]> [default = 4m] [amd64: 2m.]	Sets the large page size used for the Java heap. (Introduced in 1.4.0 update 1.)
-XX:MaxHeapFreeRatio=<n> [default = 70]	Maximum percentage of heap free after GC to avoid shrinking.
-XX:MaxNewSize=<n> [1.3.1 Sparc: 32m; 1.3.1 x86: 2.5m.]	Maximum size of new generation (in bytes). Since 1.4, MaxNewSize is computed as a function of NewRatio.
-XX:PermSize=<n[k|m]> [default = 64mb]	Size of the Permanent Generation.
-XX:MaxPermSize=<n[k|m]> [default = 64m] [5.0 and newer: 64 bit VMs are scaled 30% larger; 1.4 amd64: 96m; 1.3.1 -client: 32m.]	Size of the Permanent Generation.
-XX:MinHeapFreeRatio=<n> [default = 40]	Minimum percentage of heap free after GC to avoid expansion.
-XX:NewRatio=<n> [default = 2] [Sparc -client: 8; x86 -server: 8; x86 -client: 12.]-client: 4 (1.3) 8 (1.3.1+), x86: 12]	Ratio of new/old generation sizes.
-XX:NewSize=<n[k|m]> [default = 2.125m] [5.0 and newer: 64 bit VMs are scaled 30% larger; x86: 1m; x86, 5.0 and older: 640k]	Default size of new generation (in bytes)
-XX:ReservedCodeCacheSize=<n[k|m]> [default = 32m] [Solaris 64-bit, amd64, and -server x86: 48m; in 1.5.0_06 and earlier, Solaris 64-bit and and64: 1024m.]	Reserved code cache size (in bytes) - maximum code cache size.
-XX:SurvivorRatio=<n> [default = 8] [Solaris amd64: 6; Sparc in 1.3.1: 25; other Solaris platforms in 5.0 and earlier: 32]	Ratio of eden / survivor space size
-XX:TargetSurvivorRatio=<n> [default = 50]	Desired percentage of survivor space used after scavenge.

Option & default value	Description
-XX:ThreadStackSize=<n> [default = 512] [Sparc: 512; Solaris x86: 320 (was 256 prior in 5.0 and earlier); Sparc 64 bit: 1024; Linux amd64: 1024 (was 0 in 5.0 and earlier); all others 0.]	Thread Stack Size (in Kbytes). (0 means use default stack size)
-XX:+UseBiasedLocking [5.0: false]	Enable biased locking. For more details, see this tuning example. (Introduced in 5.0 update 6.)
-XX:+UseFastAccessorMethods	Use optimized versions of Get<Primitive>Field.
-XX:-UseISM [Not accepted for non-Solaris platforms.]	Use Intimate Shared Memory.
-XX:+UseLargePages	Use large page memory. (Introduced in 5.0 update 5.)
-XX:+UseMPSS [1.4.1 and earlier: false]	Use Multiple Page Size Support w/4mb pages for the heap. Do not use with ISM as this replaces the need for ISM. (Introduced in 1.4.0 update 1, Relevant to Solaris 9 and newer.)
-XX:+UseStringCache	Enables caching of commonly allocated strings.
-XX:AllocatePrefetchLines=<n> [Default values are 1 if the last allocated object was an instance and 3 if it was an array.]	Number of cache lines to load after the last object allocation using prefetch instructions generated in JIT compiled code.
-XX:AllocatePrefetchStyle=<[0|1|2]> [0 - no prefetch instructions are generate*d*, 1 - execute prefetch instructions after each allocation, 2 - use TLAB allocation watermark pointer to gate when prefetch instructions are executed.]	Generated code style for prefetch instructions.
-XX:+UseCompressedStrings	Use a byte[] for Strings that can be represented as pure ASCII. (Introduced in Java 6 Update 21 Performance Release)
-XX:+OptimizeStringConcat	Optimize String concatenation operations where possible. (Introduced in Java 6 Update 20)

Debugging Options

Option & default value	Description
-XX:-CITime	Prints time spent in JIT Compiler. (Introduced in 1.4.0.)
-XX:ErrorFile=./hs_err_pid<pid>.log	If an error occurs, save the error data to this file. (Introduced in 6.)
-XX:-ExtendedDTraceProbes	Enable performance-impacting dtrace probes. (Introduced in 6. Relevant to Solaris only.)
-XX:HeapDumpPath=./java_pid<pid>.hprof	Path to directory or filename for heap dump. Manageable. (Introduced in 1.4.2 update 12, 5.0 update 7.)
-XX:-HeapDumpOnOutOfMemoryError	Dump heap to file when java.lang.OutOfMemoryError is thrown. Manageable. (Introduced in 1.4.2 update 12, 5.0 update 7.)
-XX:OnError="<cmd args>;<cmd args>"	Run user-defined commands on fatal error. (Introduced in 1.4.2 update 9.)
-XX:OnOutOfMemoryError="<cmd args>;<cmd args>"	Run user-defined commands when an OutOfMemoryError is first thrown. (Introduced in 1.4.2 update 12, 6)
-XX:-PrintClassHistogram	Print a histogram of class instances on Ctrl-Break. Manageable. (Introduced in 1.4.2.) The jmap -histo command provides equivalent functionality.
-XX:-PrintConcurrentLocks	Print java.util.concurrent locks in Ctrl-Break thread dump. Manageable. (Introduced in 6.) The jstack -l command provides equivalent functionality.
-XX:-PrintCommandLineFlags	Print flags that appeared on the command line. (Introduced in 5.0.)
-XX:-PrintCompilation	Print message when a method is compiled.
-XX:-PrintGC	Print messages at garbage collection. Manageable.
-XX:-PrintGCDetails	Print more details at garbage collection. Manageable. (Introduced in 1.4.0.)
-XX:-PrintGCTimeStamps	Print timestamps at garbage collection. Manageable (Introduced in 1.4.0.).
-XX:-PrintTenuringDistribution	Print tenuring age information.
-XX:-TraceClassLoading	Trace loading of classes.
-XX:-TraceClassLoadingPreorder	Trace all classes loaded in the order referenced (not loaded). (Introduced in 1.4.2.)
-XX:-TraceClassResolution	Trace constant pool resolutions. (Introduced in 1.4.2.)
-XX:-TraceClassUnloading	Trace unloading of classes.
-XX:-TraceLoaderConstraints	Trace recording of loader constraints. (Introduced in 6.)
-XX:+PerfSaveDataToFile	Saves jvmstat binary data on exit.
-XX:ParallelGCThreads=<n> [The default value varies with the platform on which the JVM is running.]	Sets the number of garbage collection threads in the young and old parallel garbage collectors.
-XX:+UseCompressedOops	Enables the use of compressed pointers (object references represented as 32-bit offsets instead of 64-bit pointers) for optimized 64-bit performance with Java heap sizes less than 32gb.
-XX:+AlwaysPreTouch	Pre-touch the Java heap during JVM initialization. Every page of the heap is thus demand-zeroed during initialization rather than incrementally during application execution.
-XX:AllocatePrefetchDistance=<n> [The default value varies with the platform on which the JVM is running.]	Sets the prefetch distance for object allocation. Memory about to be written with the value of new objects is prefetched into cache at this distance (in bytes) beyond the address of the last allocated object. Each Java thread has its own allocation point.
-XX:InlineSmallCode=<n> [The default value varies with the platform on which the JVM is running.]	Inline a previously compiled method only if its generated native code size is less than this.

Option & default value	Description
-XX:MaxInlineSize=<n> [default = 35]	Maximum bytecode size of a method to be inlined.
-XX:FreqInlineSize=<n> [The default value varies with the platform on which the JVM is running.]	Maximum bytecode size of a frequently executed method to be inlined.
-XX:LoopUnrollLimit=<n> [The default value varies with the platform on which the JVM is running.]	Unroll loop bodies with server compiler intermediate representation node count less than this value. The limit used by the server compiler is a function of this value, not the actual value.
-XX:InitialTenuringThreshold=<n> [default = 7]	Sets the initial tenuring threshold for use in adaptive GC sizing in the parallel young collector. The tenuring threshold is the number of times an object survives a young collection before being promoted to the old, or tenured, generation.
-XX:MaxTenuringThreshold=<n> [The default value is 15 for the parallel collector and is 4 for CMS. The current largest value is 15.]	Sets the maximum tenuring threshold for use in adaptive GC sizing.
-verbose:gc	Logs garbage collector runs and how long they're taking. I generally use this as my first tool to investigate if GC is a bottleneck for a given application.

Option & default value	Description
-XX:+PrintGCApplicationConcurrentTime	Time the applications run between collection pauses.
-XX:+PrintGCApplicationStoppedTime	Length of the collection pauses.
-Xdebug -Xnoagent	Enables debugging support in the VM.
-Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=<port>	Loads in-process debugging libraries and specifies what kind of connection will be made.
-XX:HeapDumpPath=./java_pid.hprof	Path to directory or file name for heap dump.
-XX:-PrintCommandLineFlags	Print flags that appeared on the command line.
-XX:-PrintConcurrentLocks	Print java.util.concurrent locks in Ctrl-Break thread dump.

PrintGC Details Output

Hot Tip

The collection output for the PrintGC*.
Syntax
[GC [<collector>: <starting occupancy1> -> <ending occupancy1>, <pause time1> secs] <starting occupancy3> -> <ending occupancy3>, <pause time3> secs]

Additional Resources

http://java.dzone.com/articles/visualvm-refcard-additional

ABOUT THE AUTHORS

Mick Knutson, Mick Knutson has been a Enterprise technology consultant, Java Architect, project leader, Engineer, Designer and Developer, and has gained experience in disciplines including Java EE, Web Services, Mobile Computing, and Enterprise Integration Solutions. He has led training courses and book publishing engagements, authored technical white papers, and presented at seminars worldwide. As an active blogger and Tweeter, Mr. Knutson has also been inducted into the prestigious DZone.com "Most Valuable Blogger" (MVB) group, and can be followed at http://baselogic.com, http://dzone.com/users/mickknutson and @mickknutson. Mr. Knutson is the President of BASE Logic, Inc.

RECOMMENDED BOOK

Java EE6 Cookbook for securing, tuning, and extending enterprise applications: This book covers exciting recipes on securing, tuning and extending enterprise applications using a Java EE 6 implementation. http://packtpub.com/java-ee6-securing-tuning-extending-enterprise-applications-cookbook/book

DZone

DZone communities deliver over 6 million pages each month to more than 3.3 million software developers, architects and decision makers. DZone offers something for everyone, including news, tutorials, cheat sheets, blogs, feature articles, source code and more. "DZone is a developer's dream," says PC Magazine.

DZone, Inc.
150 Preston Executive Dr.
Suite 201
Cary, NC 27513

888.678.0399
919.678.0300

Refcardz Feedback Welcome
refcardz@dzone.com

Sponsorship Opportunities
sales@dzone.com

ISBN-13: 978-1-936502-57-8
ISBN-10: 1-936502-57-7
50795
9 781936 502578
$7.95

Version 1.0