

Maximizing the benefit of using IBM Monitoring and Diagnostic Tools for Java

Speakers : Toby Corbin and Julie Stalley



Agenda

- Why use IBM Monitoring and Diagnostic Tools for Java™?
- Tooling overview
 - Garbage Collection and Memory Visualiser
 - Dump Analyzer
 - Memory Analyzer
 - Diagnostics Collector
 - Health Center
- Scenario Demonstration
 - Health Center
 - Memory Analyzer
- Q&A

Why use IBM Monitoring and Diagnostic Tools for Java™ ?

- Provide a unified suite of tools to understand different aspects of Java applications.
- Provide more than visualizations – also provide observations and recommendations.
- Fully IBM supported
- Available in the IBM Support Assistant
 - Provides the “toolbox” in which analysis tools reside
 - Assist with opening PMRs and working with IBM Support
 - Product information to locate APARs and fixes
 - Search feature to query IBM and non-IBM knowledge banks

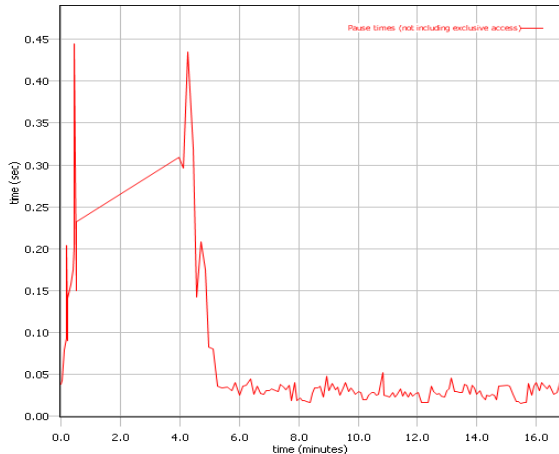
Fixing problems ...

... is much easier with the right tool for the job!

Garbage Collector and Memory Visualizer (GCMV) overview

- What problem am I solving?
 - How is the Garbage Collector (GC) behaving? Can I do better?
 - How much time is GC taking?
 - How much free memory does my JVM have?
- Tool to analyze Java verbose GC logs
- Graphs to show garbage collection and Java heap statistics over time
- Not only for memory errors, very good for performance tuning
- Recommendations use heuristics to guide you towards issues that may be limiting performance

Garbage Collector and Memory Visualizer (GCMV) overview cont...



Graphical Display of Data

- Allows graphing of all available data: pause times, heap size etc
- Allows zoom, cropping and change of axes value and units
- Allows comparison of multiple files

Tuning recommendation

⚠ The garbage collector seems to be compacting excessively. On average 45% of each pause was spent compacting the heap. Compaction occurred on 40% of collections. Possible causes of excessive compaction include the heap size being too small or the application allocating objects that are larger than any contiguous block of free space on the heap.

⚠ The garbage collector is performing system (forced) GCs. 5 out of 145 collections (3.448%) were triggered by System.gc() calls. The use of System.gc() is generally not recommended since they can cause long pauses and do not allow the garbage collection algorithms to optimise themselves. Consider inspecting your code for occurrences of System.gc().

✅ The mean occupancy in the nursery is 7%. This is low, so the gencon policy is probably an optimal policy for this workload.

ℹ The mean occupancy in the tenured area is 14%. This is low, so you have some room to shrink the heap if required.

Summary

Allocation failure count	140
Concurrent collection count	0
Forced collection count	5
GC Mode	gencon
Global collections - Mean garbage collection pause (ms)	185
Global collections - Mean interval between collections (minutes)	0.13
Global collections - Number of collections	5
Global collections - Total amount tenured (MB)	93.1
Largest memory request (bytes)	127784
Minor collections - Mean garbage collection pause (ms)	48.2
Minor collections - Mean interval between collections (ms)	7193
Minor collections - Number of collections	140
Minor collections - Total amount flipped (MB)	668
Minor collections - Total amount tenured (MB)	38.8
Proportion of time spent in garbage collection pauses (%)	0.76
Proportion of time spent unpaused (%)	99.24
Rate of garbage collection (MB/minutes)	874

Analysis and Recommendations

- Provides tuning recommendations based on data and flags errors.
- Analysis can be limited using cropping.
- Values and units used in analysis can be changed by changing axes values and units

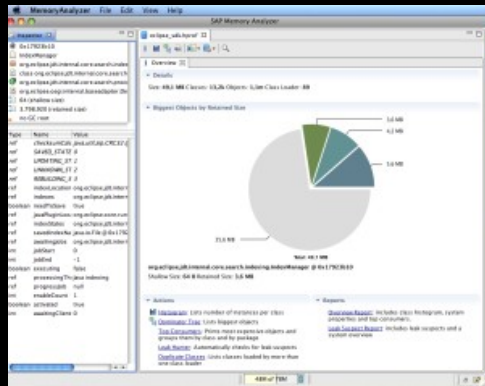
Dump Analyzer overview

- What problem am I solving?
 - What was happening when the JVM aborted?
 - What was the native memory situation in my JVM?
 - Why did my JVM deadlock? (show me the locks/threads)
- Provides general analysis of formatted system dumps
- Provides a number of specific analysis modules
- Extracts useful information
- Searches for known problems

Memory Analyzer overview

- What problem am I solving?
 - Why did I run out of Java memory?
 - What's in my Java heap? How can I explore it and get new insights?
- Tool for analyzing heap dumps and identifying memory leaks from JVMs
- Works with IBM system dumps, heapdumps and Sun HPROF binary dumps
- Provides memory leak detection and footprint analysis
 - Objects by Class, Dominator Tree Analysis, Path to GC Roots, Dominator Tree by Class Loader
- Shows areas of memory wastage
 - Collections, duplicate strings, substring/char arrays, constant value primitives
- Displays Stack trace with object references
- Provides SQL like object query language (OQL)
- Provides extension points to write analysis plugins

Memory Analyzer overview cont...

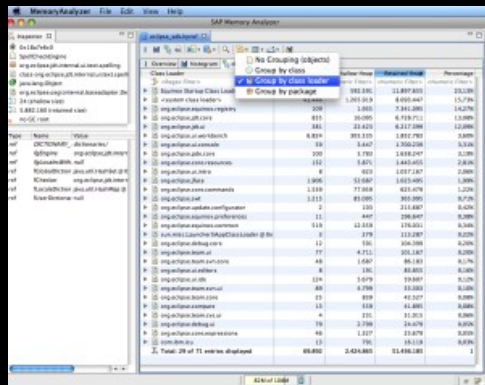
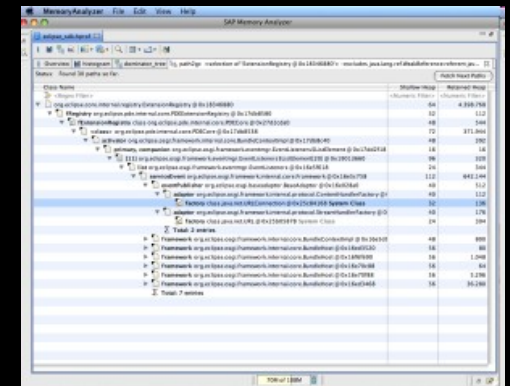


Overview:

- Overview of the heapdump including size and total number of objects.
- Provides links to continued analysis

Path to GC Roots:

- Provides the reference chain that prevents an object being garbage collected.



Dominator Tree grouped by Class Loader:

- Lists the biggest objects using a “keep alive tree”. Grouping by Class
- Loader limits the analysis to a single application in a JEE environment

Diagnostic Collector overview

- What problem am I solving?
 - Something has gone wrong!
 - JVM has detected an error and captured diagnostic information
 - Need to get this diagnostic information to the right people
- Runs as a separate process when the JVM detects a 'dump event'
 - GPF
 - Java heap OutOfMemoryError
 - Unexpected signal received
 - (optionally) JVM start, JVM stop
- Knows all possible dump locations and searches to gather all dumps into a single zip file
- Collects system dumps, Java dumps, heap dumps, verbose GC logs
- If system dump found jextract runs automatically
- At JVM start it runs a diagnostic configuration check
- Requires IBM SDK for Java version 5.0 or above

Health Center overview

- What problem am I solving?
 - What is my JVM doing? Is everything ok?
 - Why is my application running slowly? Why is it not scaling?
 - Am I using the right options?
- Live monitoring tool with very low overhead
- Understand how your application is behaving
 - Monitor Class loading, File I/O, Environment settings, Garbage Collection, Method Profiling, Locking, Native memory use
- Diagnose potential problems, with recommendations
- Works at the JVM level – no domain-specific (e.g. J2EE) information
- Suitable for all Java applications
- Available in ISA

Scenario Demonstration

- Demonstration of Health Center and Memory Analyzer running against the WebSphere plants application

Where to find more information

- IBM Monitoring and Diagnostic Tools for Java™ on developerWorks
<http://www.ibm.com/developerworks/java/jdk/tools/>
- IBM Support Assistant (ISA)
<http://www.ibm.com/software/support/isa>
- Email javatool@uk.ibm.com

Any Questions?