



Java Technologies

IBM Monitoring and Diagnostic Tools for Java™ ...

Brian Peacock
IBM Java Technology Centre

Agenda

- **Introduce the family of Java consumability tools**
- **Usage scenarios**
- **Health Center**
 - Overview
 - Demo

The IBM Monitoring and Diagnostic Tools for Java™ - family

- **Diagnostics Collector**
- **Health Center**
- **Dump Analyzer**
- **Garbage Collection and Memory Visualizer (GCMV)**

Plus ...

Close relatives

- **IBM Diagnostics Guide**
- **IBM Guided Activity Assistant (IGAA)**
- **Eclipse Memory Analyzer Tool (MAT)**
- **Diagnostics Tool Framework for Java (DTFJ)**
- **IBM Support Assistant (ISA)**

Scenarios - “How do I ... ?”

- **Diagnostics Guide**

- The manual you all know and love
- Tells you about Java options
- Explains diagnosis procedures

- **IGAA**

- Guides you through problem scenarios
 - eg. I have OutOfMemory ... what do I do now ?
- Also available in InfoCenter format

Scenarios – Artifact Analysis

■ **Diagnostics Collector**

- Where is that dump/trace ?
- What do you mean I forgot to send you xxxx ?

■ **Dump Analyzer**

- Err ... what does this dump actually show me ?

■ **GCMV**

- Garbage Collection ... that's an internal JVM thing that I don't need to worry about ... isn't it ?

■ **MAT**

- OpenSource heap analyzer
- IBM provide adapters to read IBM Java dumps

Scenarios – misc items

- **DTFJ**
 - Common API to read Java dumps
- **IBM Support Assistant**
 - You want an IBM tool ... this is the place to go
 - You have a software problem ... this is the place to go

Scenarios – Live Monitoring and Analysis

- **Health Center**

Typical customer questions

- **What is my Java application doing ?**
- **Why is it doing that ?**
- **Why is my application going so slowly ?**
- **How can I make it go faster ?**
- **Is my application scaling well ?**
- **Is our algorithm sensible ?**
- **Do we need to tune the JVM ?**



... and some pretty tricky questions ...

- **Is my configuration sensible ?**
- **Is the system stable ?**
- **Have I got a memory leak ?**
- **Is the application about to crash ... leading to a very awkward conversation with my boss ?**

... which boil down to ...

- **Is my application healthy ?**
- **If not, how can I fix it ?**



do **you** have any similar questions that you want our tooling to answer ?

Health Center

- **New tooling from the IBM Java Technology Center**
 - Currently available in beta
- **Provides a view inside a running Java application**
 - What the application is doing
 - Performance analysis
 - Configuration recommendations
 - Configuration recommendations
 - (so important it's worth mentioning it twice)

Target Environments

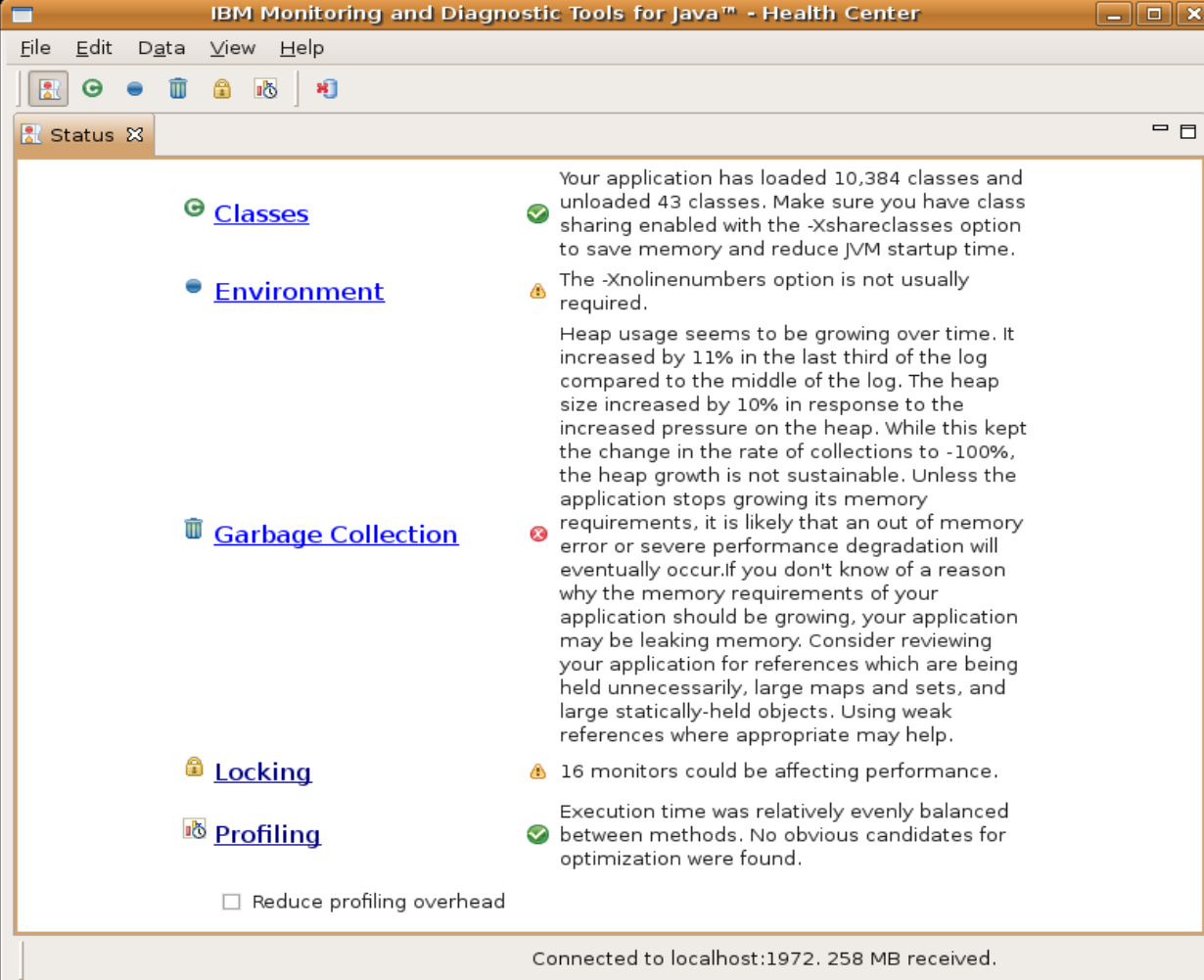
- **Use during the development phase to detect and resolve:**
 - Performance problems
 - Functional issues
- **Use in production to detect and resolve:**
 - Configuration problems
 - Stability issues

How does it work ?

- **Small agent runs on the target JVM**
 - Minimal overhead (circa 3%)
 - Supports all IBM Java platforms
 - Requires Java 5 onwards
- **Data is fed to client which does the analysis**
 - Windows or Linux based client
 - Analyzes/monitors various parts of Java subsystem
 - Makes recommendations based on data received
 - Dynamically updated as more data is received

Initial screen – overview and recommendations






- Recommendation engine provides
 - Visual indicator of status
 - Explanation of potential problems and solutions



IBM Monitoring and Diagnostic Tools for Java™ - Health Center

File Edit Data View Help

Status


-  [Classes](#)
-  [Environment](#)
-  [Garbage Collection](#)
-  [Locking](#)
-  [Profiling](#)

Reduce profiling overhead

Your application has loaded 10,384 classes and unloaded 43 classes. Make sure you have class sharing enabled with the -Xshareclasses option to save memory and reduce JVM startup time.

The -Xnolinenumbers option is not usually required.

Heap usage seems to be growing over time. It increased by 11% in the last third of the log compared to the middle of the log. The heap size increased by 10% in response to the increased pressure on the heap. While this kept the change in the rate of collections to -100%, the heap growth is not sustainable. Unless the application stops growing its memory requirements, it is likely that an out of memory error or severe performance degradation will eventually occur. If you don't know of a reason why the memory requirements of your application should be growing, your application may be leaking memory. Consider reviewing your application for references which are being held unnecessarily, large maps and sets, and large statically-held objects. Using weak references where appropriate may help.

 16 monitors could be affecting performance.

Execution time was relatively evenly balanced between methods. No obvious candidates for optimization were found.

Connected to localhost:1972. 258 MB received.

Classloading

- Shows all loaded classes
- Shows load time
- Visualizes classloading activity
- Identifies shared classes
- Makes recommendations

The screenshot displays the IBM Monitoring and Diagnostic Tools for Java - Health Center interface. The window title is "IBM Monitoring and Diagnostic Tools for Java™ - Health Center". The interface includes a menu bar (File, Edit, Data, View, Help) and a toolbar with various icons. The main content area is divided into several panels:

- Status Panel:** Contains links for [Classes](#), [Environment](#), [Garbage Collection](#), [Locking](#), and [Profiling](#). There is also a checkbox for "Reduce profiling overhead".
- Class loading timeline Panel:** Shows a bar chart titled "Loaded classes" with the x-axis labeled "time (minutes)" ranging from 0:00 to 212:27. The chart displays vertical green bars representing class loading events.
- Classes loaded Panel:** Includes a "Filter classes:" input field with "Apply" and "Clear" buttons. Below it is a table with columns "Time loaded", "Shared cache", and "Classname".
- Recommendation Panel:** Displays a message: "Your application has loaded 10,385 classes and unloaded 43 classes. Make sure you have class sharing enabled with the Xshareclasses option to save memory and reduce JVM startup time."

At the bottom of the window, it shows "Connected to localhost:1972. 258 MB received."

Time loaded	Shared cache	Classname
0:00	No	java/lang/Object
0:00	No	java/lang/9VMInternals
0:00	No	java/lang/Class
0:00	No	java/io/Serializable
0:00	No	java/lang/reflect/GenericDeclaration
0:00	No	java/lang/reflect/Type
0:00	No	java/lang/reflect/AnnotatedElement
0:00	No	java/lang/Cloneable
0:00	No	java/lang/String
0:00	No	java/lang/Comparable

Environment analysis

- Detects unsupported Java options
- Detects options which may hurt performance or serviceability
- Useful for remote diagnosis of configuration-related problems

The screenshot displays the 'IBM Monitoring and Diagnostic Tools for Java™ - Health Center' application. The main window is titled 'Configuration' and shows a list of system properties and their values. A warning message is visible in the bottom-left pane, stating: 'The -Xnolinelnumbers option is not usually required.' The 'Java Virtual Machine' pane shows the following properties:

Property	Value
Full version	J2RE 1.6.
Java home	/opt/ibm
Java vendor	IBM Corp
Java virtual machine name	IBM J9 VM
Process id	17834
Version	1.6

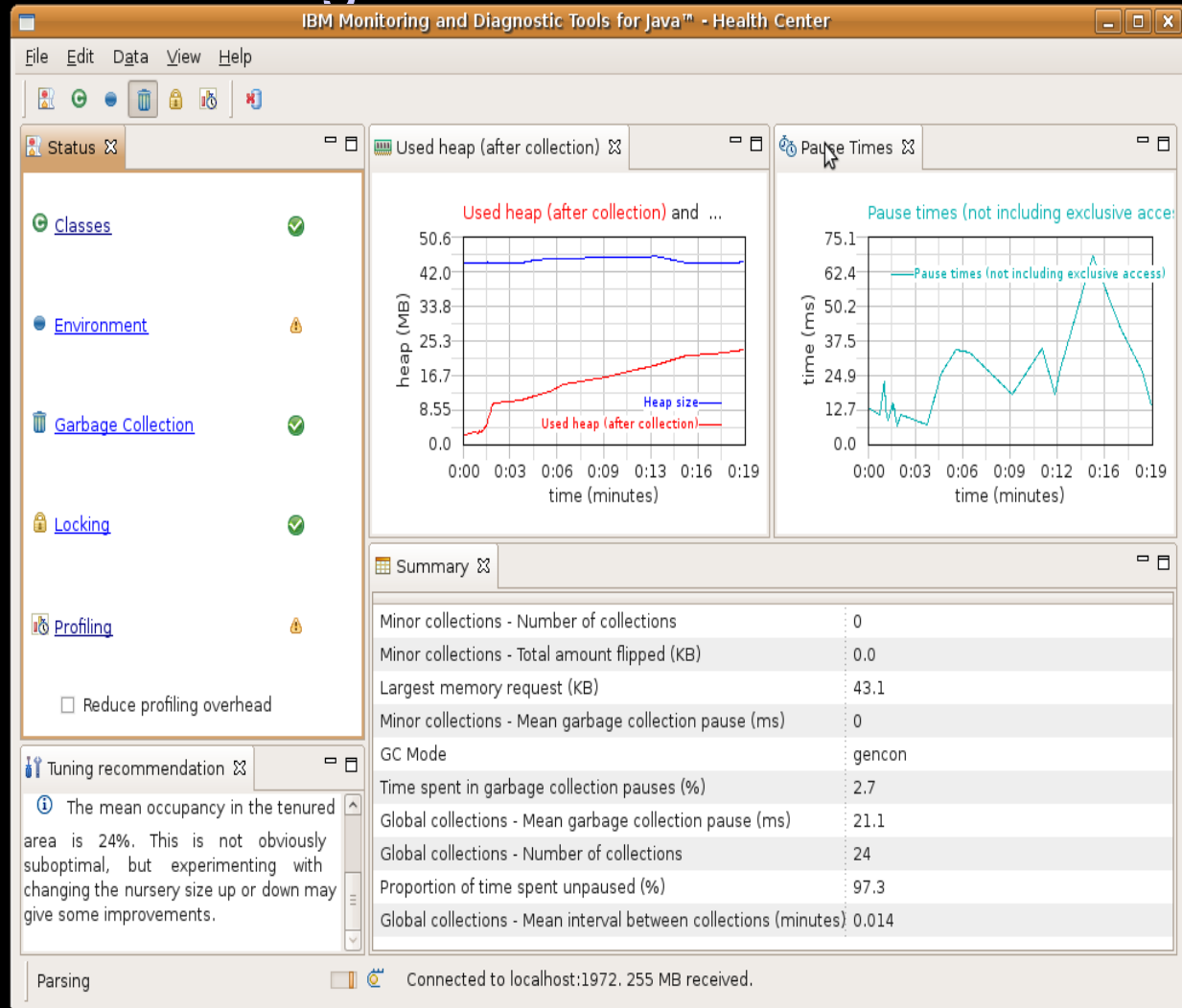
The 'System' pane shows the following properties:

Property	Value
Architecture	x86
Number of available processors	2
Operating system	Lin
Operating system version	2.6

At the bottom of the window, it indicates 'Connected to localhost:1973. 9.16 MB received.'

Garbage collection monitoring

- Visualizes heap usage and gc pause times over time
- Identifies memory leaks
- Suggests command-line and tuning parameters
- Same recommendation logic as GCMV



Lock analysis

- Always-on lock monitoring
- Quickly allows the usage of all locks to be profiled
- Helps to identify points of contention in the application that are preventing scaling

IBM Monitoring and Diagnostic Tools for Java™ - Health Center

File Edit Data View Help

Status

Monitors bar chart

Inflated Java Monitors

Slow (height) and % miss (colour)

Total Slow lock count (number)

Monitor

Inflated Java m...

Monitors

Inflated Java Monitors

% miss	Gets	Slow	Recursive	% util	Average hold time	Name
1	246918	2613	3610	0	31637	[0909DCCC] or
12	78451	4080	44701	0	137200	[8F591658] jav
0	37880	5	0	0	726	[966FB3A8] org
17	19701	1667	9843	0	190267	[8F5915B0] jav

Connected to localhost:1972. 258 MB received.

Reduce profiling overh...

lang/Object@A7C8EE68 (Object)" had a high miss percentage indicating that 33% of attempts by a thread to own the lock (when the requesting thread did not already own it) required the thread to

Method profiling

- Always-on profiling offers insight into application activity
- Identifies the hottest methods in an application
- Full call stacks to identify where methods are being called from and what methods they call
- No bytecode instrumentation, no recompiling

The screenshot displays the 'Method profile' window in the IBM Monitoring and Diagnostic Tools for Java - Health Center. The window title is 'IBM Monitoring and Diagnostic Tools for Java™ - Health Center'. The menu bar includes 'File', 'Edit', 'Data', 'View', and 'Help'. Below the menu bar is a toolbar with various icons. The main content area shows a table of method profiles with columns for 'Samples', 'Self (%)', 'Self', 'Tree (%)', 'Tree', and 'Method'. The top row is highlighted in yellow, representing the method 'java/lang/ClassLoader.defineClassImpl(Ljava/lang/String;[BII)'. Below the table, there are tabs for 'Invocation paths' and 'Called methods'. The 'Called methods' tab is active, showing a call stack for 'defineClassImpl' that includes 'defineClass' (86.3%) and another 'defineClass' (12.9%). At the bottom of the window, there is a status bar showing 'Parsing' and 'Connected to localhost:1972. 257 MB received.'

▼ Samples	Self (%)	Self	Tree (%)	Tree	Method
751	16.2		19.4		java/lang/ClassLoader.defineClassImpl(Ljava/lang/String;[BII)
222	4.79		26.0		java/util/regex/Pattern\$Branch.match(Ljava/util/regex/Matche
159	3.43		26.0		java/util/regex/Pattern\$Curly.match1(Ljava/util/regex/Matche
143	3.09		21.1		org/eclipse/osgi/baseadaptor/loader/ClasspathManager.defi
135	2.92		26.0		java/util/regex/Pattern\$GroupTail.match(Ljava/util/regex/Mat
122	2.63		3.39		java/util/regex/Pattern\$8.isSatisfiedBy(I)
114	2.46		25.9		java/util/regex/Pattern\$BranchConn.match(Ljava/util/regex/I
111	2.4		2.4		java/lang/String.lastIndexOf(I)
99	2.14		26.0		java/util/regex/Pattern\$BmpCharProperty.match(Ljava/util/re
82	1.77		26.0		java/util/regex/Pattern\$GroupHead.match(Ljava/util/regex/M

Methods that call ClassLoader.defineClassImpl()

- ▼ M defineClassImpl
 - ▼ M defineClass
 - ▼ M defineClass (86.3%)
 - ▶ M defineClass
 - ▼ M defineClass (12.9%)

Availability

- **Requires Java 5 or Java 6**
- **GA-level (suitable for production systems) from Java 5 SR10 or Java 6 SR5**
- **Agent ships with VM from Java 5 SR9 or Java 6 SR3**
- **Beta-level function available from Java 5 SR8 and Java 6 SR1**

Useful links

- **Direct link**

- <http://www.ibm.com/developerworks/java/jdk/tools/index.html>

- **Or ...**

- Use your favourite search tool to look for ...

- IBM Java

- Take first link which will be to IBM DeveloperWorks

- Select

- “Monitoring and diagnostic tools”

Health Center demo