Quick and Easy Profiling With Integrated Tools

Jaroslav Bachorík
Jiří Sedláček
Gregg Sporar

Sun Microsystems, Inc.

TS-9555

Goal

Learn how easy it is to use powerful profiling tools within an integrated development environment (IDE).

Agenda

What Sorts of Problems Can Be Solved?

Advantages of Integrated Profiling Tools

Case Study: Application With a Memory Leak

The Next Step: NetBeans™ IDE 6.0

Case Study: Roller Performance Problems

Resources

Q&A

Agenda

What Sorts of Problems Can Be Solved?

Advantages of Integrated Profiling Tools

Case Study: Application With a Memory Leak

The Next Step: NetBeans™ IDE 6.0

Case Study: Roller Performance Problems

Resources

Q&A

What Sorts of Problems Can Be Solved?

Three things

- Threading problems
- CPU bottlenecks
- Memory usage problems/memory leaks

DEMO

Examining CPU Usage

Agenda

What Sorts of Problems Can Be Solved?

Advantages of Integrated Profiling Tools

Case Study: Application With a Memory Leak

The Next Step: NetBeans IDE 6.0

Case Study: Roller Performance Problems

Resources

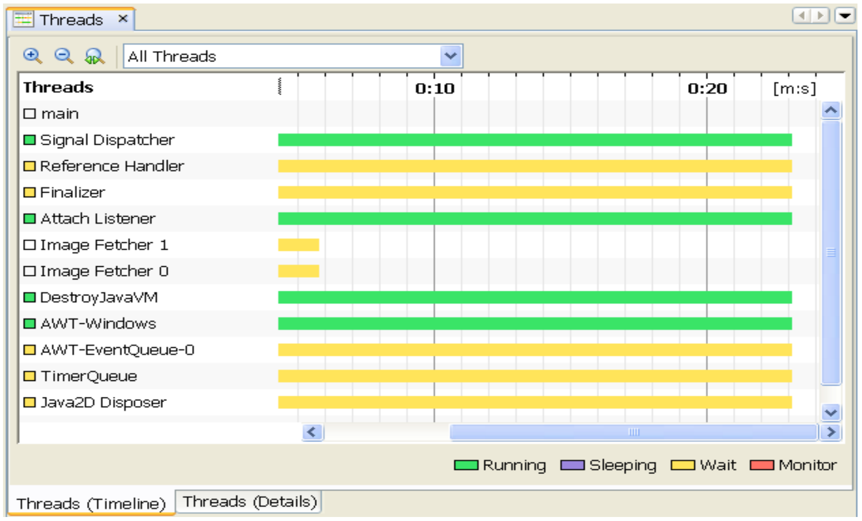
Q&A

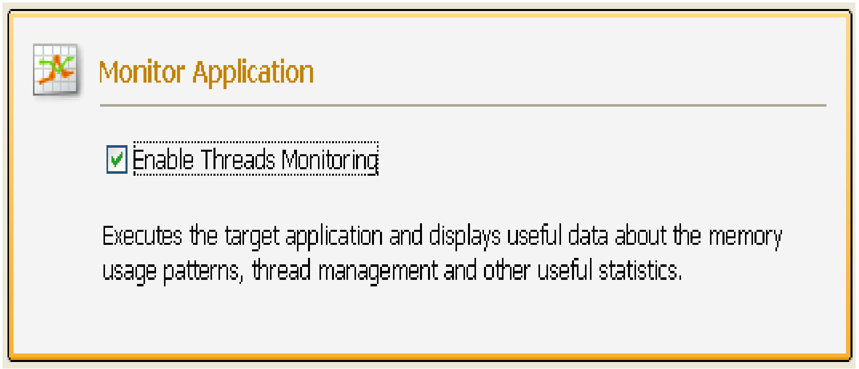
Advantages of Integrated Profiling Tools

NetBeans IDE 5.5

- No need to start an additional tool
- Already understands your project structure
- Task-oriented UI
- Work flow changes from: edit/compile/test/debug to edit/compile/test/debug/**profile**

Goal: fix performance problems before releasing the software

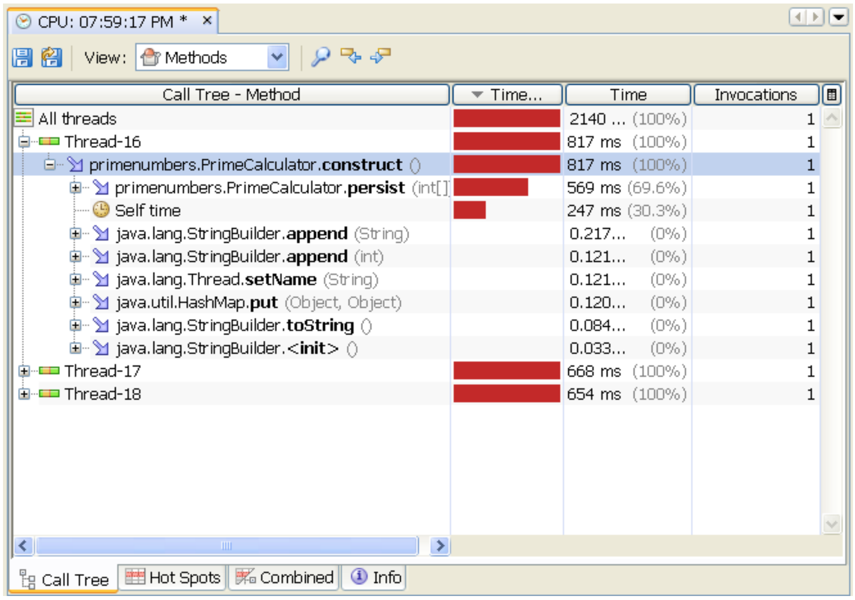




Monitor Application

[✓] Enable Threads Monitoring

Executes the target application and displays useful data about the memory usage patterns, thread management and other useful statistics.

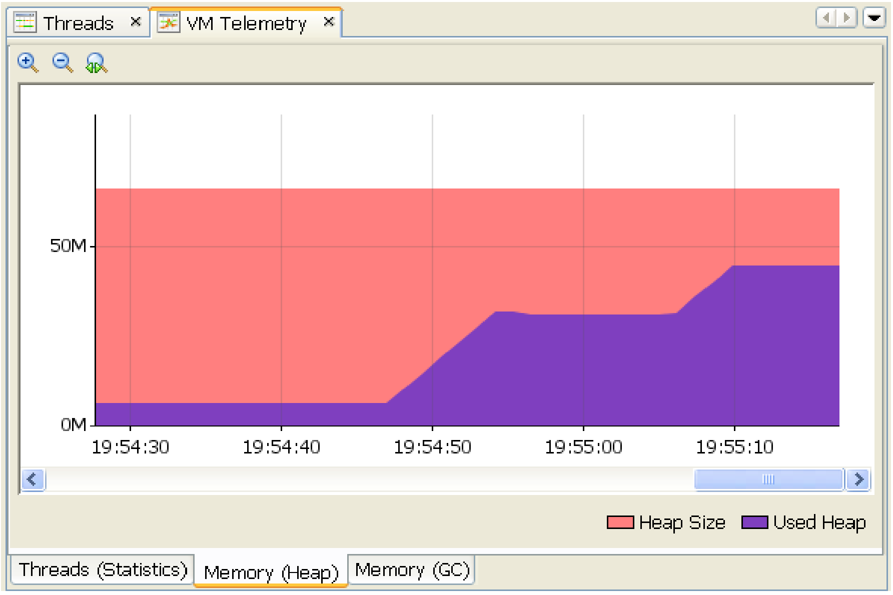


CPU: 07:59:17 PM * ×

View: Methods

Call Tree - Method	Time...	Time	Invocations
All threads		2140 ... (100%)	1
Thread-16		817 ms (100%)	1
primenumbers.PrimeCalculator.**construct** ()		817 ms (100%)	1
primenumbers.PrimeCalculator.**persist** (int[])		569 ms (69.6%)	1
Self time		247 ms (30.3%)	1
java.lang.StringBuilder.**append** (String)		0.217... (0%)	1
java.lang.StringBuilder.**append** (int)		0.121... (0%)	1
java.lang.Thread.**setName** (String)		0.121... (0%)	1
java.util.HashMap.**put** (Object, Object)		0.120... (0%)	1
java.lang.StringBuilder.**toString** ()		0.084... (0%)	1
java.lang.StringBuilder.**<init>** ()		0.033... (0%)	1
Thread-17		668 ms (100%)	1
Thread-18		654 ms (100%)	1

Call Tree | Hot Spots | Combined | Info



Agenda

What Sorts of Problems Can Be Solved?

Advantages of Integrated Profiling Tools

Case Study: Application With a Memory Leak

The Next Step: NetBeans IDE 6.0

Case Study: Roller Performance Problems

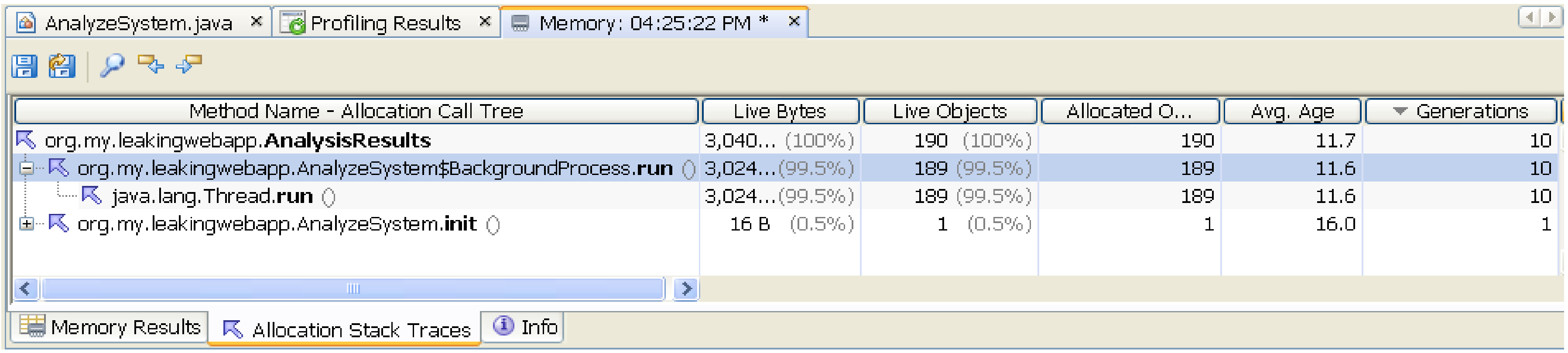
Resources

Q&A

Case Study: Application With a Memory Leak

- Analyzes hardware/software configuration problems for Sun's customers
- Developed during 2000–2004
- JDK™ 1.? release (Later moved to JDK™ 1.4 release)
- ~150,000 LOCs, which does not include
  - JavaServer Pages™ (JSP™) technology
  - A subsystem written in Perl
- Memory leak found in live, production system
- Hard to reproduce the problem—seemed to occur somewhat randomly

JDK™ = Java Development Kit


AnalyzeSystem.java | Profiling Results | Memory: 04:25:22 PM *

Method Name - Allocation Call Tree	Live Bytes	Live Objects	Allocated O...	Avg. Age	Generations
org.my.leakingwebapp.**AnalysisResults**	3,040... (100%)	190 (100%)	190	11.7	10
org.my.leakingwebapp.AnalyzeSystem$BackgroundProcess.**run** ()	3,024...(99.5%)	189 (99.5%)	189	11.6	10
java.lang.Thread.**run** ()	3,024...(99.5%)	189 (99.5%)	189	11.6	10
org.my.leakingwebapp.AnalyzeSystem.**init** ()	16 B (0.5%)	1 (0.5%)	1	16.0	1

Memory Results | Allocation Stack Traces | Info

Agenda

What Sorts of Problems Can Be Solved?

Advantages of Integrated Profiling Tools

Case Study: Application With a Memory Leak

The Next Step: NetBeans IDE 6.0

Case Study: Roller Performance Problems

Resources

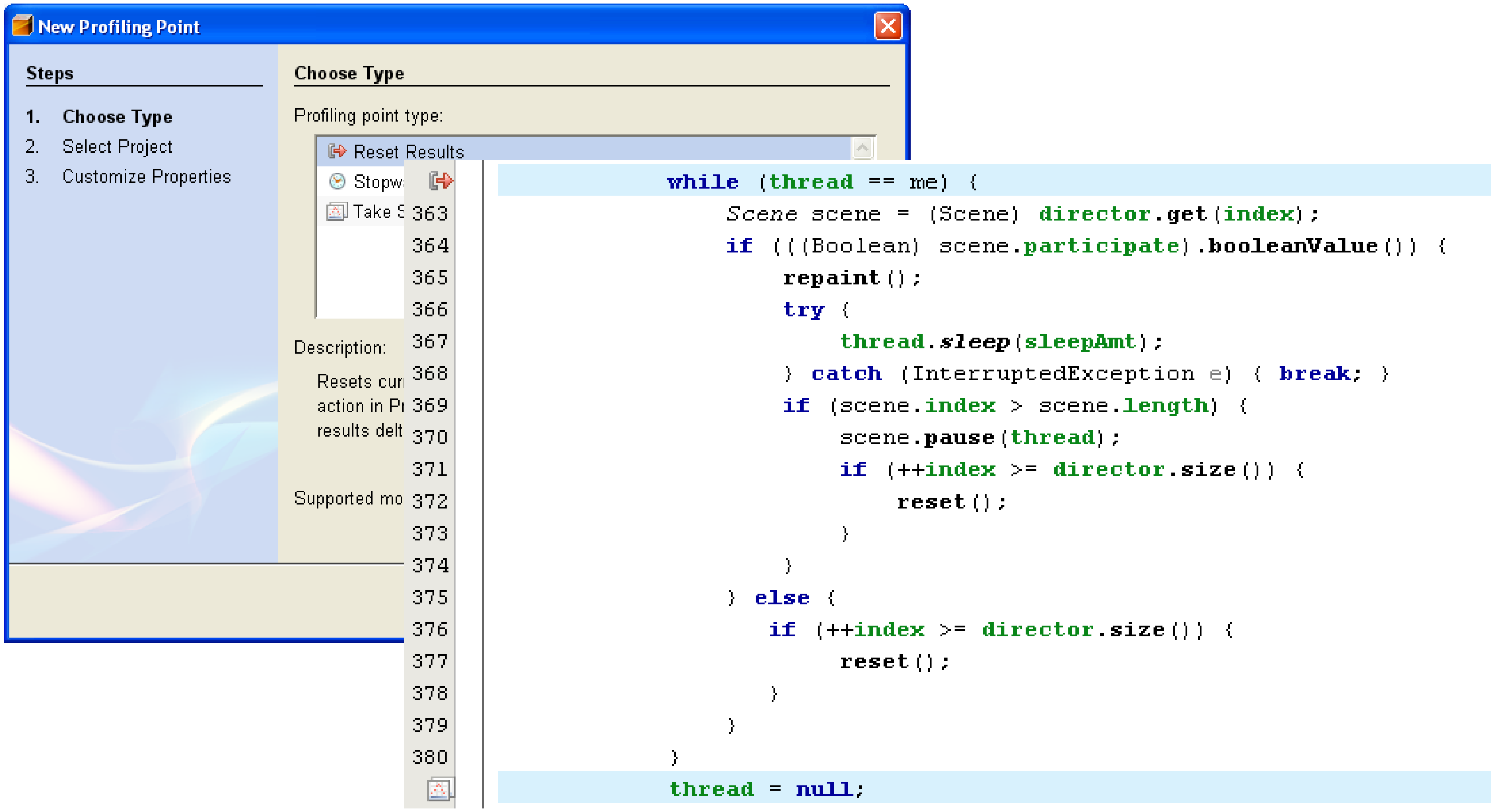
Q&A

The Next Step: NetBeans IDE 6.0

- Profiling points
- JMeter integration
- HeapWalker
- Areas of interest
- Dynamic attach
- Included in standard distribution

The Next Step: NetBeans IDE 6.0 (Cont.)

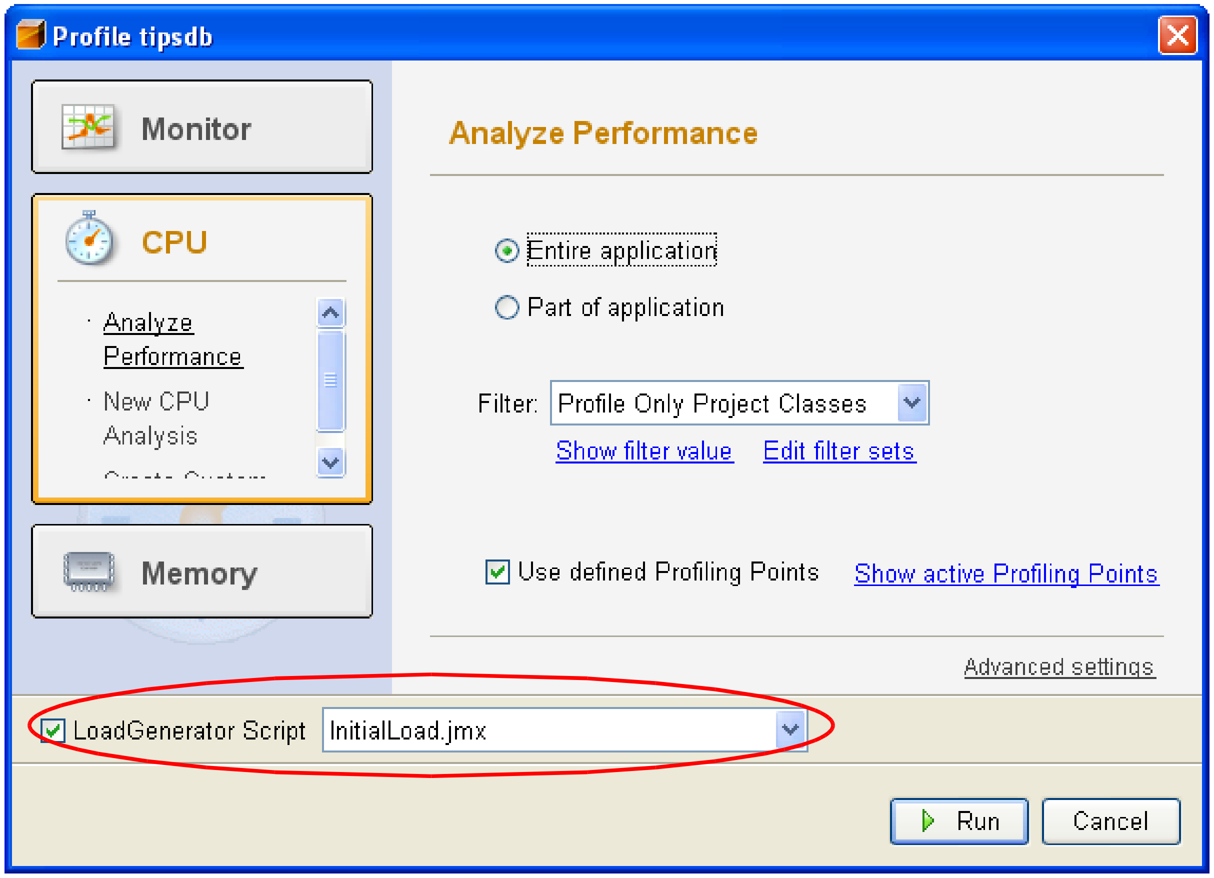
Profiling points



```
            while (thread == me) {
                Scene scene = (Scene) director.get(index);
                if (((Boolean) scene.participate).booleanValue()) {
                    repaint();
                    try {
                        thread.sleep(sleepAmt);
                    } catch (InterruptedException e) { break; }
                    if (scene.index > scene.length) {
                        scene.pause(thread);
                        if (++index >= director.size()) {
                            reset();
                        }
                    }
                } else {
                    if (++index >= director.size()) {
                        reset();
                    }
                }
            }
            thread = null;
```

The Next Step: NetBeans IDE 6.0 (Cont.)

JMeter integration


Profile tipsdb

Analyze Performance

(●) Entire application
( ) Part of application

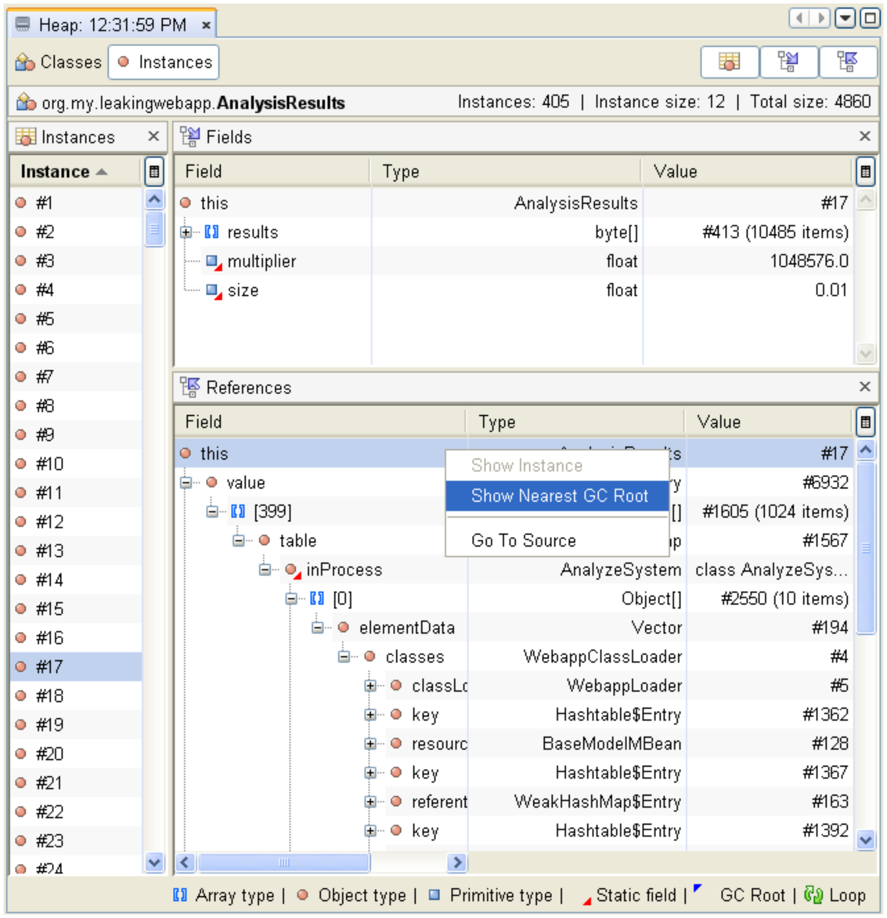
Filter: Profile Only Project Classes

[✓] Use defined Profiling Points

[✓] LoadGenerator Script InitialLoad.jmx

The Next Step: NetBeans IDE 6.0 (Cont.)

HeapWalker



Heap: 12:31:59 PM

Classes | Instances

org.my.leakingwebapp.**AnalysisResults** Instances: 405 | Instance size: 12 | Total size: 4860

Instances

Instance
#1
#2
#3
#4
#5
#6
#7
#8
#9
#10
#11
#12
#13
#14
#15
#16
#17
#18
#19
#20
#21
#22
#23
#24

Fields

Field	Type	Value
this	AnalysisResults	#17
results	byte[]	#413 (10485 items)
multiplier	float	1048576.0
size	float	0.01

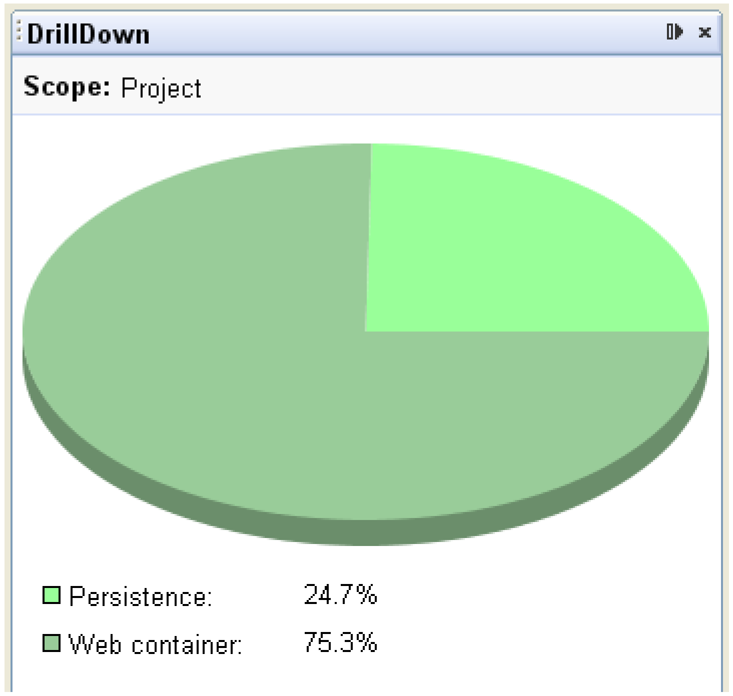
References

Field	Type	Value
this	AnalysisResults	#17
value		#6932
[399]	[]	#1605 (1024 items)
table		#1567
inProcess	AnalyzeSystem	class AnalyzeSys...
[0]	Object[]	#2550 (10 items)
elementData	Vector	#194
classes	WebappClassLoader	#4
classLc	WebappLoader	#5
key	Hashtable$Entry	#1362
resourc	BaseModelMBean	#128
key	Hashtable$Entry	#1367
referent	WeakHashMap$Entry	#163
key	Hashtable$Entry	#1392

Array type | Object type | Primitive type | Static field | GC Root | Loop

The Next Step: NetBeans IDE 6.0 (Cont.)

Areas of interest

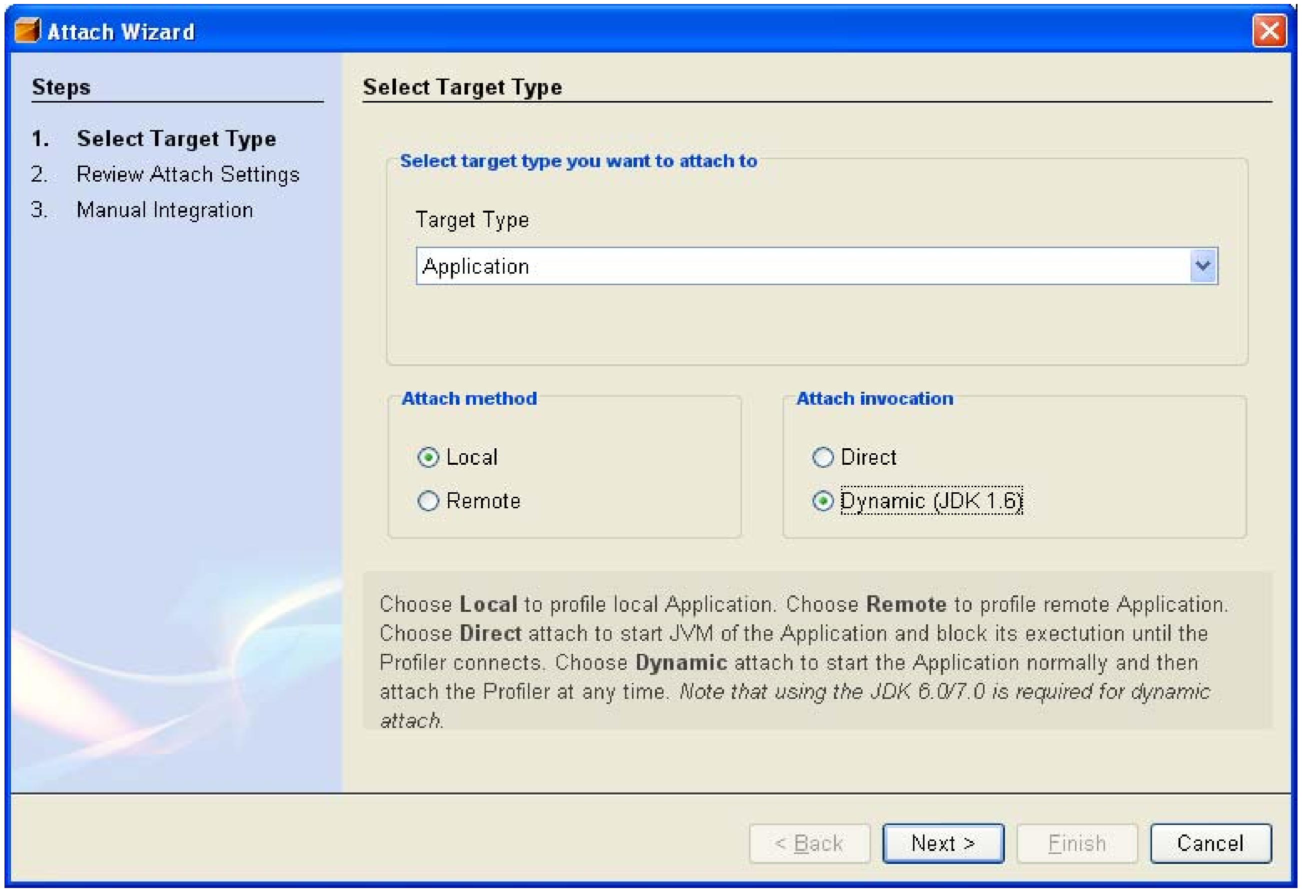


Scope: Project

🟩 Persistence:	24.7%
🟩 Web container:	75.3%

The Next Step: NetBeans IDE 6.0 (Cont.)

Dynamic attach



Select Target Type

Select target type you want to attach to

Target Type: Application

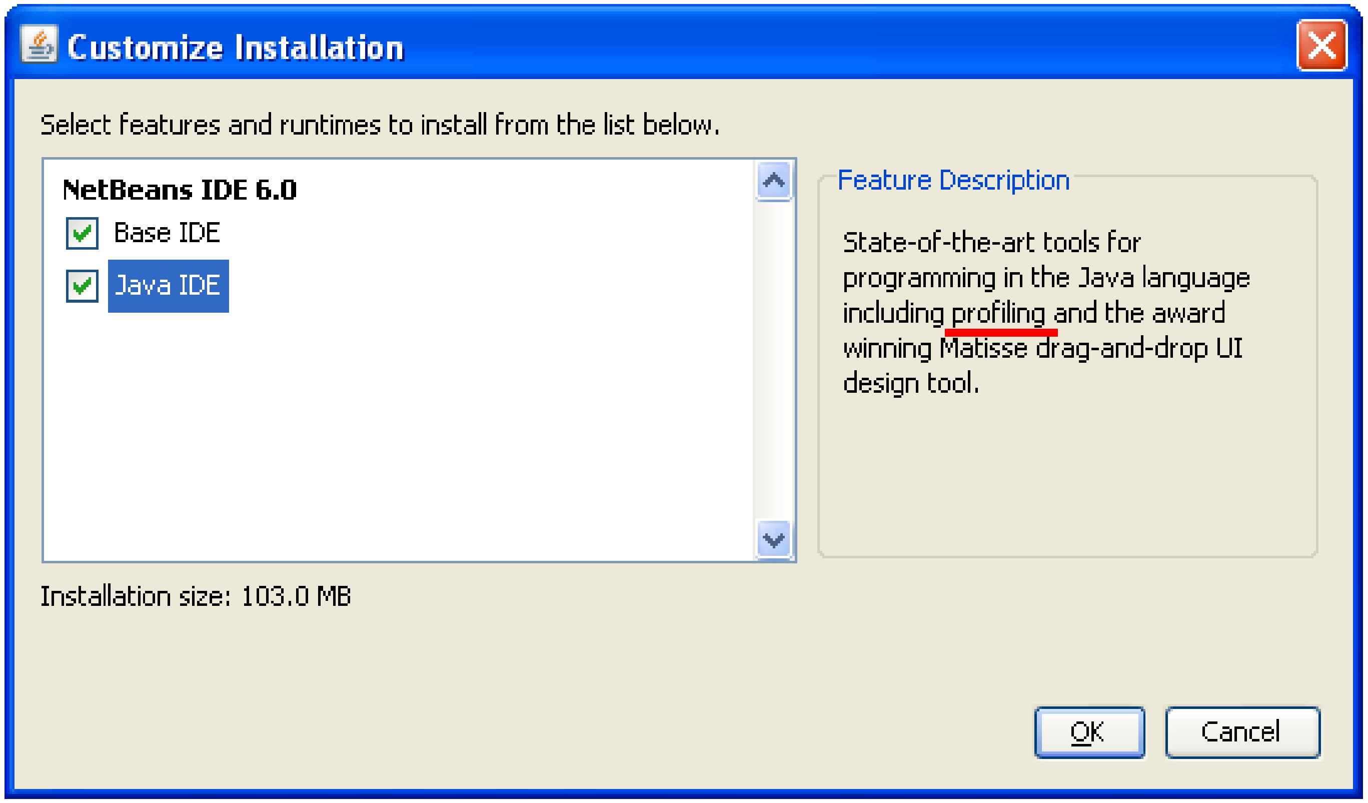
Attach method: [✓ Local] Remote

Attach invocation: Direct [✓ Dynamic (JDK 1.6)]

Choose **Local** to profile local Application. Choose **Remote** to profile remote Application. Choose **Direct** attach to start JVM of the Application and block its exectution until the Profiler connects. Choose **Dynamic** attach to start the Application normally and then attach the Profiler at any time. *Note that using the JDK 6.0/7.0 is required for dynamic attach.*

The Next Step: NetBeans IDE 6.0 (Cont.)

Included in standard distribution



Select features and runtimes to install from the list below.

NetBeans IDE 6.0

- [✓] Base IDE
- [✓] Java IDE

Feature Description

State-of-the-art tools for programming in the Java language including profiling and the award winning Matisse drag-and-drop UI design tool.

Installation size: 103.0 MB

Agenda

What Sorts of Problems Can Be Solved?

Advantages of Integrated Profiling Tools

Case Study: Application With a Memory Leak

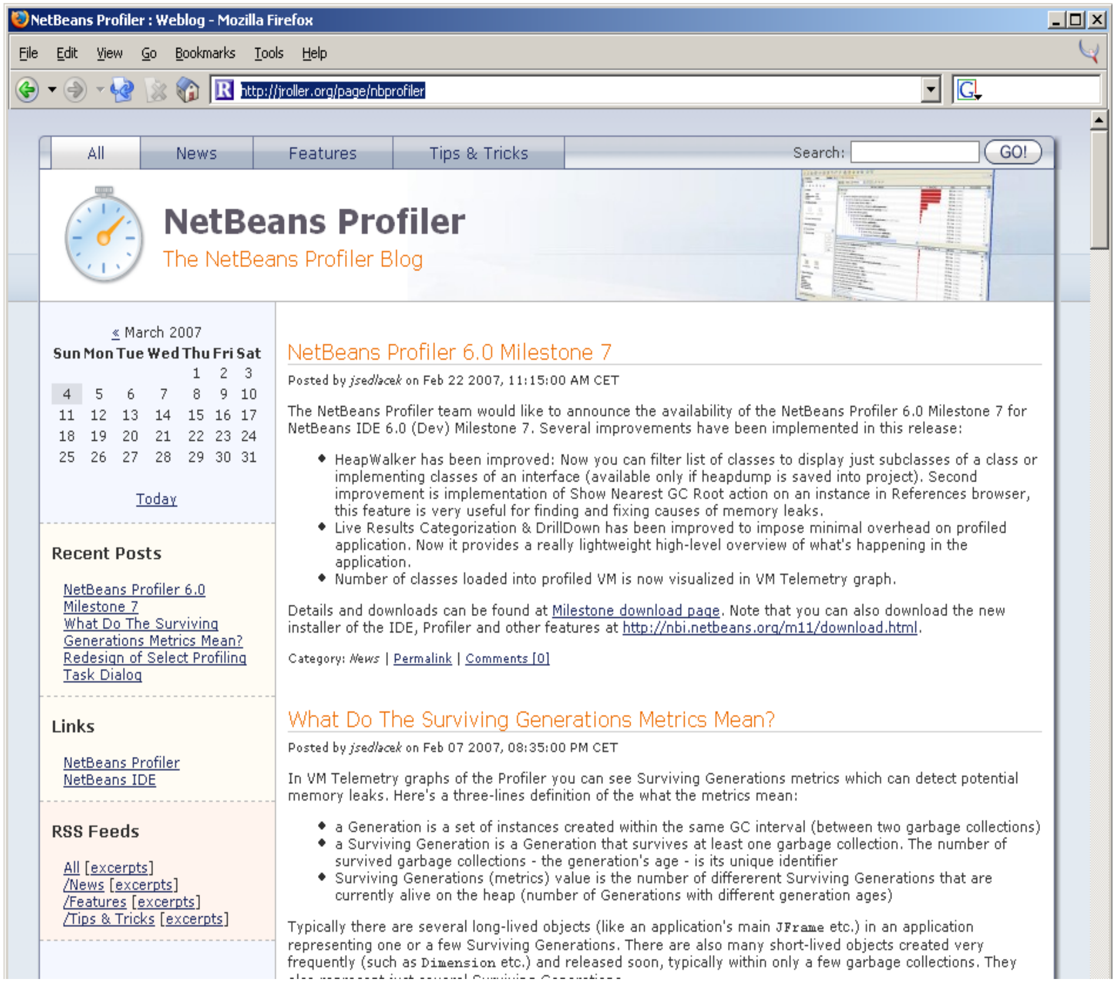
The Next Step: NetBeans IDE 6.0

Case Study: Roller Performance Problems

Resources

Q&A

Case Study: Roller Performance Problems


NetBeans Profiler : Weblog - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://jroller.org/page/nbprofiler
All
News
Features
Tips & Tricks
Search:
GO!

NetBeans Profiler

The NetBeans Profiler Blog

« March 2007

Sun	Mon	Tue	Wed	Thu	Fri	Sat
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

Today

Recent Posts

NetBeans Profiler 6.0 Milestone 7
What Do The Surviving Generations Metrics Mean?
Redesign of Select Profiling Task Dialog

Links

NetBeans Profiler
NetBeans IDE

RSS Feeds

All [excerpts]
/News [excerpts]
/Features [excerpts]
/Tips & Tricks [excerpts]

NetBeans Profiler 6.0 Milestone 7

Posted by *jsedlacek* on Feb 22 2007, 11:15:00 AM CET

The NetBeans Profiler team would like to announce the availability of the NetBeans Profiler 6.0 Milestone 7 for NetBeans IDE 6.0 (Dev) Milestone 7. Several improvements have been implemented in this release:

- HeapWalker has been improved: Now you can filter list of classes to display just subclasses of a class or implementing classes of an interface (available only if heapdump is saved into project). Second improvement is implementation of Show Nearest GC Root action on an instance in References browser, this feature is very useful for finding and fixing causes of memory leaks.
- Live Results Categorization & DrillDown has been improved to impose minimal overhead on profiled application. Now it provides a really lightweight high-level overview of what's happening in the application.
- Number of classes loaded into profiled VM is now visualized in VM Telemetry graph.

Details and downloads can be found at Milestone download page. Note that you can also download the new installer of the IDE, Profiler and other features at http://nbi.netbeans.org/m11/download.html.

Category: *News* | Permalink | Comments [0]

What Do The Surviving Generations Metrics Mean?

Posted by *jsedlacek* on Feb 07 2007, 08:35:00 PM CET

In VM Telemetry graphs of the Profiler you can see Surviving Generations metrics which can detect potential memory leaks. Here's a three-lines definition of the what the metrics mean:

- a Generation is a set of instances created within the same GC interval (between two garbage collections)
- a Surviving Generation is a Generation that survives at least one garbage collection. The number of survived garbage collections - the generation's age - is its unique identifier
- Surviving Generations (metrics) value is the number of differerent Surviving Generations that are currently alive on the heap (number of Generations with different generation ages)

Typically there are several long-lived objects (like an application's main `JFrame` etc.) in an application representing one or a few Surviving Generations. There are also many short-lived objects created very frequently (such as `Dimension` etc.) and released soon, typically within only a few garbage collections. They

Case Study: Roller Performance Problems (Cont.)

- Open source blog server (http://rollerweblogger.org/project/)
- Version 2.3 (June, 2006)
- Struts based
- >400 Classes
- >190 JSP pages
- Almost 40,000 LOCs
- Saw a problem with slow performance

DEMO

Roller Performance Problems

Agenda

What Sorts of Problems Can Be Solved?

Advantages of Integrated Profiling Tools

Case Study: Application With a Memory Leak

The Next Step: NetBeans IDE 6.0

Case Study: Roller Performance Problems

Resources

Q&A

Resources

- NetBeans Profiler Team Home Page (http://profiler.netbeans.org)
  - Latest bits
  - Documentation
  - The team blog
  - Tutorials and articles
- More information on memory leak detection: *Software Test & Performance* magazine, April, 2007 issue (http://stpmag.com/retrieve/stp-0704.htm)

Resources (Cont.)

- New Hands On Lab that uses the NetBeans IDE Profiler
  - LAB-4410: Benchmarking Web 2.0 Applications for Performance
  - Pick up a DVD in the lab room: 130/131
- **BOF-9123—Visualize Runtime Problems: A New All-in-One JDK Software Troubleshooting Tool**
  - Tonight at 9:55 PM in Esplanade 303

Agenda

What Sorts of Problems Can Be Solved?

Advantages of Integrated Profiling Tools

Case Study: Application With a Memory Leak

The Next Step: NetBeans IDE 6.0

Case Study: Roller Performance Problems

Resources

Q&A

Q&A

jaroslav.bachorik@sun.com

jiri.sedlacek@sun.com

gregg.sporar@sun.com

Quick and Easy Profiling With Integrated Tools

Jaroslav Bachorík
Jiří Sedláček
Gregg Sporar

Sun Microsystems, Inc.

TS-9555