

Performance Profiling Using NetBeans Profiler

Sang Shin
JPassion.com
“Learn with Passion!”



Topics

- CPU Profiling
- NetBeans Profiler
 - > Profiling schemes
 - > Profiling points
 - > Monitoring threads status
 - > Identifying performance bottlenecks

CPU Profiling

What is CPU Profiling?

- CPU profiling provides information about where an application is spending most of its time
 - > Execution performance profiling
- When is CPU profiling needed or beneficial?
 - > Poor application throughput measured against a predetermined target
 - > Saturated cpu utilization
 - > High sys or kernel cpu utilization
 - > High lock contention
 - > To a lesser extent, poor application scalability

CPU Profiling Strategies

- Some profilers such as NetBeans Profiler allows you profile a subset of an application.
 - > Approach can be useful when profiling the entire application is very intrusive or severely disturbs application's performance
 - > Profile from a large suspected area of your application then drill down
- DTrace scripts can also be effective
 - > Supported on Solaris, some Linux, Mac OS 10.5+ "Leopard"

NetBeans Profiler

Profiling Schemes

- Two schemes
 - > Sampling
 - > Instrumentation – “all methods” vs “root methods”
- Sampling
 - > Lower overhead
 - > Profiler performs sampling periodically
 - > Less precise than instrumentation
- Instrumentation
 - > Methods of the profiled application are instrumented
 - > “All methods” - all methods are instrumented
 - > “Root method” - only selected root methods are instrumented – less overhead than “All methods”

Profiling Filter

- You can also set a filter to limit the classes that are profiled and control the overhead
- Built-in filters
 - > Profile all classes (higher overhead, too much info)
 - > Profile only project classes (use root method)
 - > Profile project and subproject classes
 - > Quick filter – create your own filter
 - > Exclude Java Core Classes

Recommendation

- Detect suspected area using “Sampling” first
- Then use Instrumentation scheme in smaller area
- Use root method and/or filter to further refine the profiling

Lab:

Exercise 1: Profiling Schemes 5114_javaperf_nbprofiler.zip



Profiling Points

- You can set profiling points in your source code to more precisely control the collection of profiling results
 - > Similar to debugger breakpoints, Profiling points trigger actions when certain trigger conditions occur
- Trigger conditions
 - > Execution of a line of code
 - > Time elapsed
 - > Memory used
- Trigger actions
 - > Create heap dumps
 - > Reset results collected
 - > Run load generator script
 - > Take results snapshot

Lab:

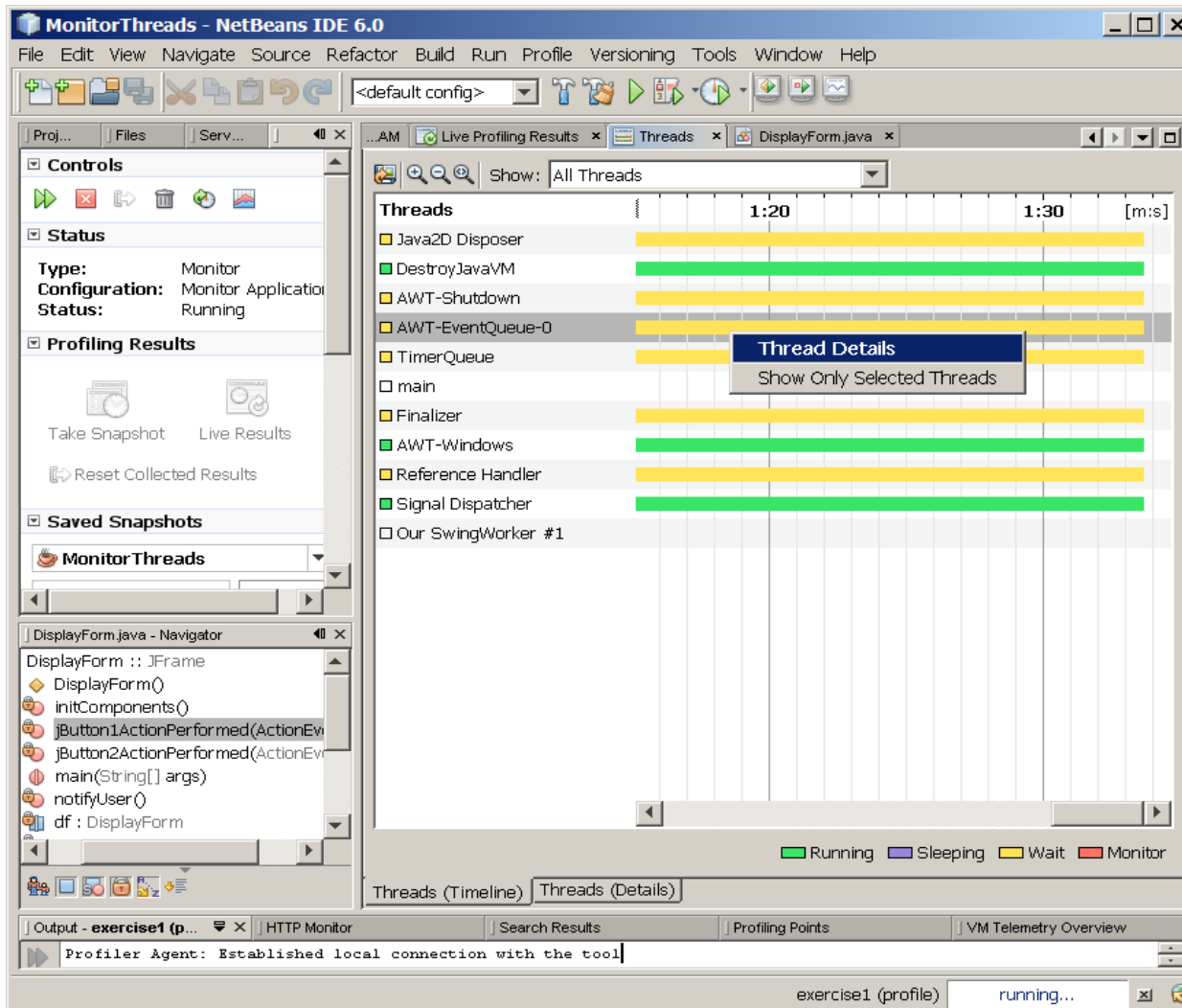
Exercise 2: Profiling Points
5114_javaperf_nbprofiler.zip



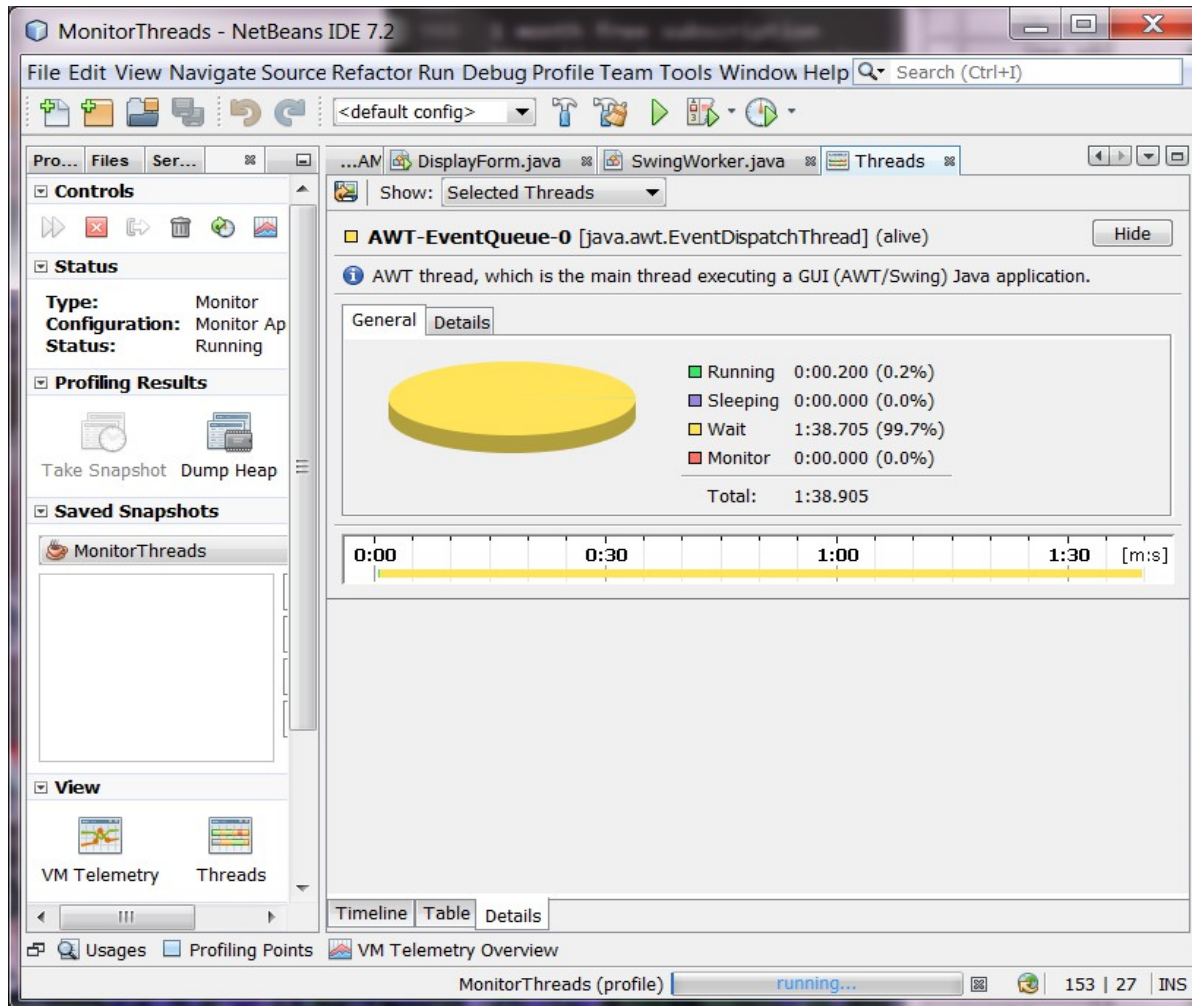
Monitoring Threads Status

- On the Threads Timeline, the NetBeans profiler displays the current and past status of all threads in an application.
- Thread Details give you detailed information about one or more selected threads, including a list of all state changes in the thread life.

Threads Timeline



Thread Detail



Lab:

Exercise 3: Monitoring Thread Status 5114_javaperf_nbprofiler.zip



Identifying CPU Bottlenecks

- Can be set up to report only on particular methods or the entire application.
- A graph categorizes where CPU time has been spent.
- Click on the graph sections to drill down from high-level categories to more detailed profiling information.
- You can do load testing by starting JMeter scripts at the beginning of a profiling session.

CPU Profiling

NetBeans IDE 7.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help Search (Ctrl+I)

Pro... Files Ser... %

...AM cpu: 9:17:21 AM * % cpu: 9:19:07 AM * %

View: Methods

Controls

Status

Type: CPU
Configuration: Analyze P
Status: Running

Profiling Results

Take Snapshot Dump Heap

Saved Snapshots

ComputePrimeNumberV

View

VM Telemetry Threads

Call Tree - Method

Method	Time...	Time	Invocat...
http-bio-8084-exec-13	...	(100%)	1
demo.Performance.processRequest (javax.s...	...	(100%)	1
demo.Performance.doCountPrimeNumbers	...	(99.7%)	1
org.netbeans.modules.web.monitor.server.Mo	...	(0.2%)	2
org.apache.catalina.connector.ResponseFacad	...	(0%)	1
org.apache.catalina.connector.CoyoteWriter.p	...	(0%)	23
org.apache.catalina.connector.CoyoteWriter.d	...	(0%)	1
Self time	...	(0%)	1
java.lang.Integer.parseInt (String)	...	(0%)	1
org.apache.catalina.loader.WebappClassLoad	...	(0%)	2
java.lang.StringBuilder.append (long)	...	(0%)	2
java.lang.StringBuilder.append (String)	...	(0%)	11
org.apache.catalina.connector.ResponseFacad	...	(0%)	1
java.lang.StringBuilder.append (int)	...	(0%)	3
java.lang.StringBuilder.toString ()	...	(0%)	5
org.apache.catalina.connector.CoyoteWriter.p	...	(0%)	2
java.util.HashMap.put (Object, Object)	...	(0%)	2
javax.servlet.http.HttpServletRequestWrapper.	...	(0%)	1
java.lang.StringBuilder.<init> ()	...	(0%)	5
java.lang.System.getSecurityManager ()	...	(0%)	2
http-bio-8084-exec-1	...	(100%)	1

Method Name Filter (Contains)

Call Tree Hot Spots Combined Info

Output Usages Profiling Points VM Telemetry Overview HTTP Server Monitor

143 | 6

Lab:

**Exercise 4: Identifying
Performance Bottleneck
5114_javaperf_nbprofiler.zip**



Learn with Passion!
JPassion.com

