Software Profiling

Roman Matveev

What is profiling?

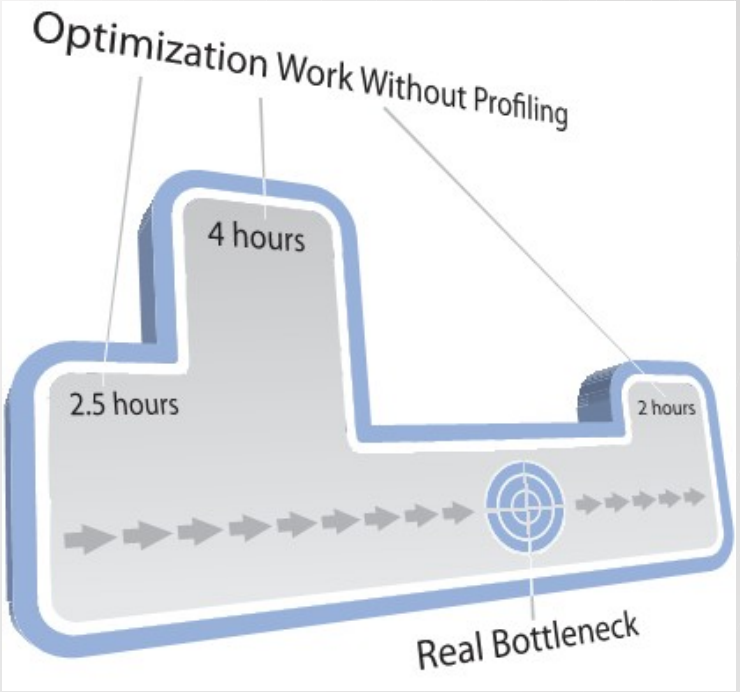
- Presents diagnostic information about applications
- Details on runtime duration, threads, methods
- Identifies inefficiencies and bottlenecks
- Allows for analysis between different snapshots per each routine instrumentation
- Incremental performance improvements amount over long timescales

What use is profiling to me?

- Code telemetry allows you to pinpoint memory leaks
  - As the amount of instances of a class increases, the garbage collector is unable to clean up the code
- Tools allow for exploration of individual objects
  - Traversing references to objects in heap allows profiler to locate paths to garbage collector roots
  - Tracing allocations of methods allows developer to analyze the profile instance to find inefficiencies within the call stacks
- Makes life easier
  - Like using version control tools, continuous integration tools, testing frameworks, profiling code is a major benefit

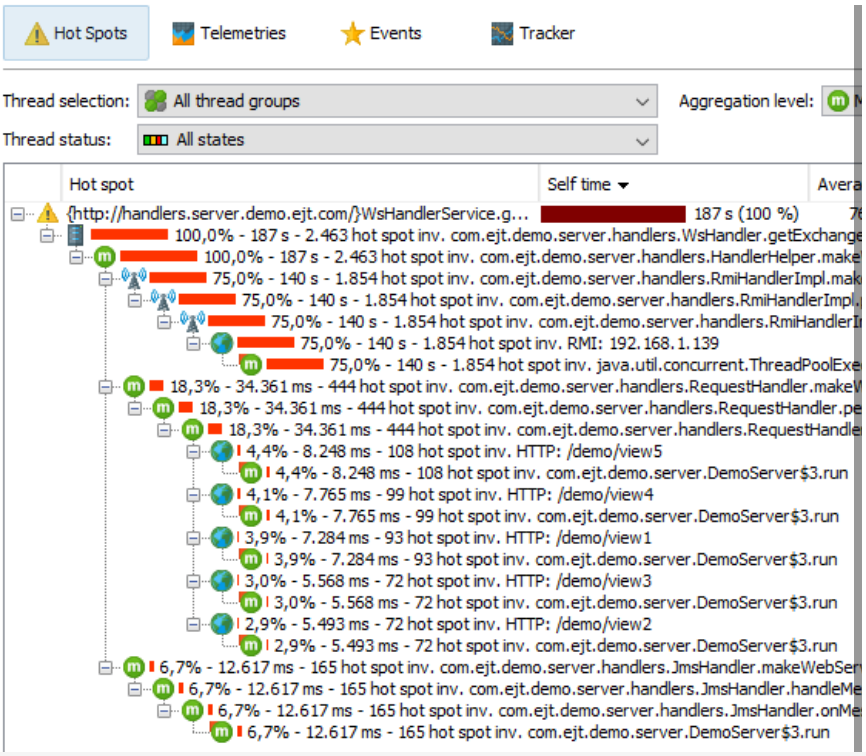
When is it most useful?

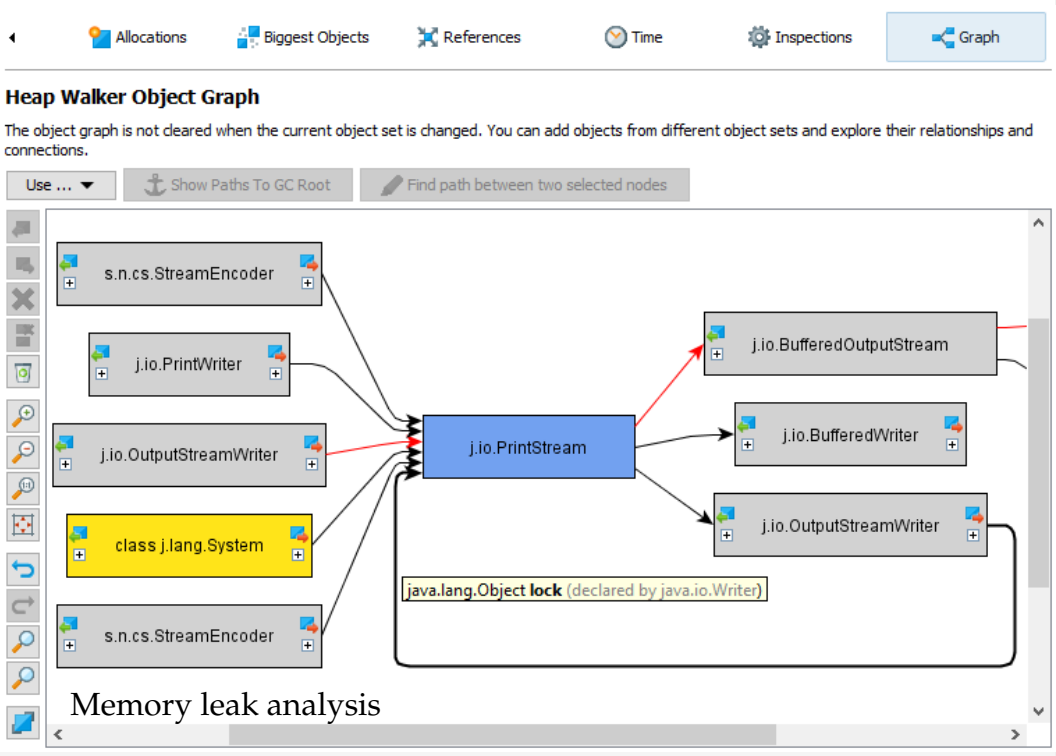
- Streamline agile workflows processes
- Fast-growing applications and services need to optimize in order to keep up performance

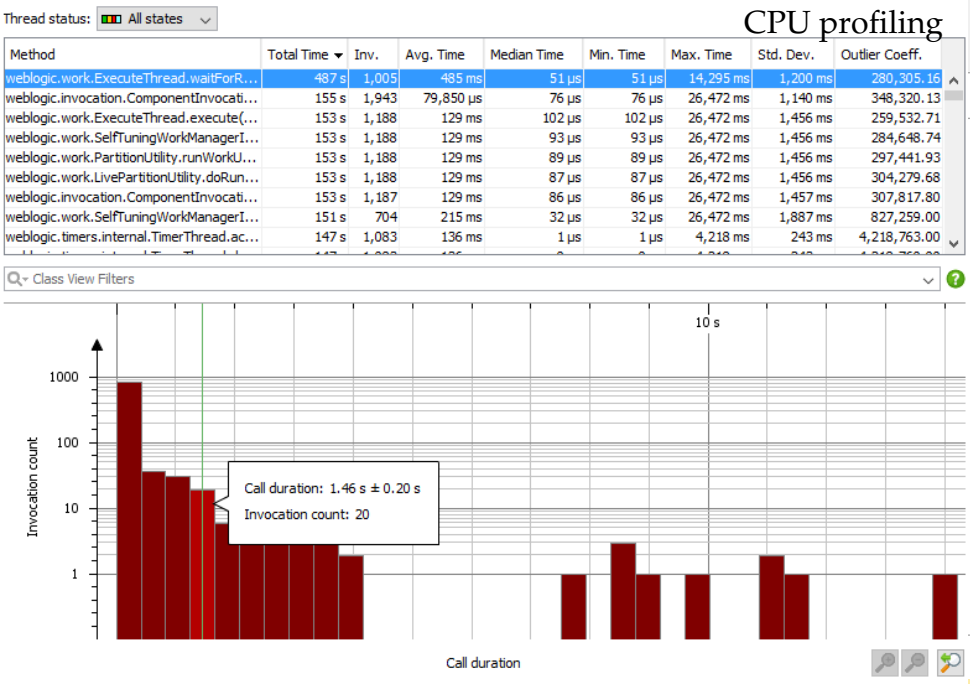


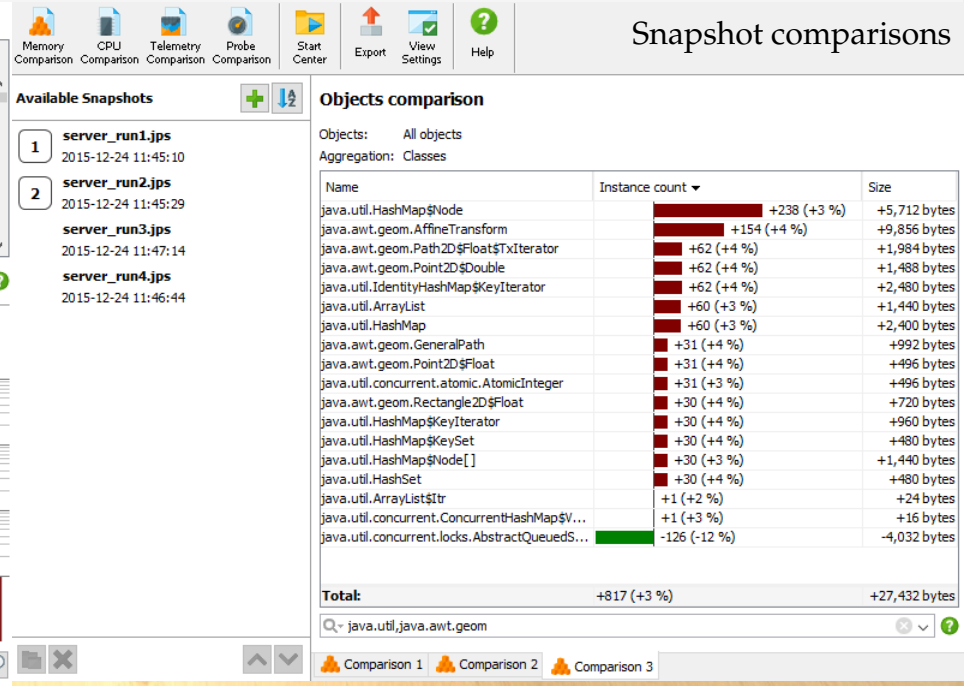
Examples of profiling tools

- VisualVM (Java) and dotTrace (.NET) for JetBrains IDE's
- JProfiler for Java Virtual Machine (JVM)
  - Integrates into Eclipse, IntelliJ, NetBeans, etc.
- YourKit profiler for .NET framework
- Stackify's Prefix (for Java and .NET)
- Profiler in NetBeans 8.1+
- Diagnostic Tools in Visual Studio
- Hotshot (deprecated), now Coldshot
- PyPy for Python (replaces CPython)


Hot Spots
Telemetries
Events
Tracker
Thread selection:
All thread groups
Aggregation level:
Thread status:
All states
Hot spot
Self time
{http://handlers.server.demo.ejt.com/}WsHandlerService.g...
187 s (100 %)
100,0% - 187 s - 2.463 hot spot inv. com.ejt.demo.server.handlers.WsHandler.getExchange
100,0% - 187 s - 2.463 hot spot inv. com.ejt.demo.server.handlers.HandlerHelper.make
75,0% - 140 s - 1.854 hot spot inv. com.ejt.demo.server.handlers.RmiHandlerImpl.mak
75,0% - 140 s - 1.854 hot spot inv. com.ejt.demo.server.handlers.RmiHandlerImpl.
75,0% - 140 s - 1.854 hot spot inv. com.ejt.demo.server.handlers.RmiHandlerI
75,0% - 140 s - 1.854 hot spot inv. RMI: 192.168.1.139
75,0% - 140 s - 1.854 hot spot inv. java.util.concurrent.ThreadPoolExe
18,3% - 34.361 ms - 444 hot spot inv. com.ejt.demo.server.handlers.RequestHandler.makeW
18,3% - 34.361 ms - 444 hot spot inv. com.ejt.demo.server.handlers.RequestHandler.pe
18,3% - 34.361 ms - 444 hot spot inv. com.ejt.demo.server.handlers.RequestHandle
4,4% - 8.248 ms - 108 hot spot inv. HTTP: /demo/view5
4,4% - 8.248 ms - 108 hot spot inv. com.ejt.demo.server.DemoServer$3.run
4,1% - 7.765 ms - 99 hot spot inv. HTTP: /demo/view4
4,1% - 7.765 ms - 99 hot spot inv. com.ejt.demo.server.DemoServer$3.run
3,9% - 7.284 ms - 93 hot spot inv. HTTP: /demo/view1
3,9% - 7.284 ms - 93 hot spot inv. com.ejt.demo.server.DemoServer$3.run
3,0% - 5.568 ms - 72 hot spot inv. HTTP: /demo/view3
3,0% - 5.568 ms - 72 hot spot inv. com.ejt.demo.server.DemoServer$3.run
2,9% - 5.493 ms - 72 hot spot inv. HTTP: /demo/view2
2,9% - 5.493 ms - 72 hot spot inv. com.ejt.demo.server.DemoServer$3.run
6,7% - 12.617 ms - 165 hot spot inv. com.ejt.demo.server.handlers.JmsHandler.makeWebSer
6,7% - 12.617 ms - 165 hot spot inv. com.ejt.demo.server.handlers.JmsHandler.handleMe
6,7% - 12.617 ms - 165 hot spot inv. com.ejt.demo.server.handlers.JmsHandler.onMe
6,7% - 12.617 ms - 165 hot spot inv. com.ejt.demo.server.DemoServer$3.run


Allocations
Biggest Objects
References
Time
Inspections
Graph
Heap Walker Object Graph
The object graph is not cleared when the current object set is changed. You can add objects from different object sets and explore their relationships and connections.
Use ...
Show Paths To GC Root
Find path between two selected nodes
s.n.cs.StreamEncoder
j.io.PrintWriter
j.io.OutputStreamWriter
class j.lang.System
s.n.cs.StreamEncoder
j.io.PrintStream
j.io.BufferedOutputStream
j.io.BufferedWriter
j.io.OutputStreamWriter
java.lang.Object lock (declared by java.io.Writer)
Memory leak analysis


Thread status:
All states
CPU profiling
Method
Total Time
Inv.
Avg. Time
Median Time
Min. Time
Max. Time
Std. Dev.
Outlier Coeff.
weblogic.work.ExecuteThread.waitForR...
487 s
1,005
485 ms
51 µs
51 µs
14,295 ms
1,200 ms
280,305.16
weblogic.invocation.ComponentInvocati...
155 s
1,943
79,850 µs
76 µs
76 µs
26,472 ms
1,140 ms
348,320.13
weblogic.work.ExecuteThread.execute(...
153 s
1,188
129 ms
102 µs
102 µs
26,472 ms
1,456 ms
259,532.71
weblogic.work.SelfTuningWorkManagerI...
153 s
1,188
129 ms
93 µs
93 µs
26,472 ms
1,456 ms
284,648.74
weblogic.work.PartitionUtility.runWorkU...
153 s
1,188
129 ms
89 µs
89 µs
26,472 ms
1,456 ms
297,441.93
weblogic.work.LivePartitionUtility.doRun...
153 s
1,188
129 ms
87 µs
87 µs
26,472 ms
1,456 ms
304,279.68
weblogic.invocation.ComponentInvocati...
153 s
1,187
129 ms
86 µs
86 µs
26,472 ms
1,457 ms
307,817.80
weblogic.work.SelfTuningWorkManagerI...
151 s
704
215 ms
32 µs
32 µs
26,472 ms
1,887 ms
827,259.00
weblogic.timers.internal.TimerThread.ac...
147 s
1,083
136 ms
1 µs
1 µs
4,218 ms
243 ms
4,218,763.00
Class View Filters
10 s
1000
100
10
1
Invocation count
Call duration: 1.46 s ± 0.20 s
Invocation count: 20
Call duration


Memory Comparison
CPU Comparison
Telemetry Comparison
Probe Comparison
Start Center
Export
View Settings
Help
Snapshot comparisons
Available Snapshots
1
server_run1.jps
2015-12-24 11:45:10
2
server_run2.jps
2015-12-24 11:45:29
server_run3.jps
2015-12-24 11:47:14
server_run4.jps
2015-12-24 11:46:44
Objects comparison
Objects: All objects
Aggregation: Classes
Name
Instance count
Size
java.util.HashMap$Node +238 (+3 %) +5,712 bytes
java.awt.geom.AffineTransform +154 (+4 %) +9,856 bytes
java.awt.geom.Path2D$Float$TxIterator +62 (+4 %) +1,984 bytes
java.awt.geom.Point2D$Double +62 (+4 %) +1,488 bytes
java.util.IdentityHashMap$KeyIterator +62 (+4 %) +2,480 bytes
java.util.ArrayList +60 (+3 %) +1,440 bytes
java.util.HashMap +60 (+3 %) +2,400 bytes
java.awt.geom.GeneralPath +31 (+4 %) +992 bytes
java.awt.geom.Point2D$Float +31 (+4 %) +496 bytes
java.util.concurrent.atomic.AtomicInteger +31 (+3 %) +496 bytes
java.awt.geom.Rectangle2D$Float +30 (+4 %) +720 bytes
java.util.HashMap$KeyIterator +30 (+4 %) +960 bytes
java.util.HashMap$KeySet +30 (+4 %) +480 bytes
java.util.HashMap$Node[] +30 (+3 %) +1,440 bytes
java.util.HashSet +30 (+4 %) +480 bytes
java.util.ArrayList$Itr +1 (+2 %) +24 bytes
java.util.concurrent.ConcurrentHashMap$V... +1 (+3 %) +16 bytes
java.util.concurrent.locks.AbstractQueuedS... -126 (-12 %) -4,032 bytes
Total: +817 (+3 %) +27,432 bytes
java.util,java.awt.geom
Comparison 1
Comparison 2
Comparison 3

References

- https://en.wikipedia.org/wiki/Profiling_(computer_programming)
- https://www.youtube.com/watch?v=032aTGa-1XM
- https://www.ej-technologies.com/products/jprofiler/overview.html
- https://msdn.microsoft.com/en-us/library/mt210448.aspx
- https://smartbear.com/learn/code-profiling/how-to-choose-profiling-tools/
- https://stackify.com/what-is-code-profiling/