JBossProfiler – JVMTI Module – Memory and Method execution - Index

Author: Clebert Suconic @ jboss.com

Concepts:

Runtime Execution- Use JVMPI version if you need

This stills under construction. If you need to capture methods callings, please use the JVMPI version for now.

Memory Profiler

The memory module at JBossProfiler uses heap navigations functions presented in JVMTI.
JBossProfiler memory module captures into three plain text files:

<Chosen prefix >_classes.<chosen suffix>
<Chosen prefix >_objects.<chosen suffix>
<Chosen prefix >_references.<chosen suffix>

We will refer to these classes as just classes, objects and references classes.

File Formats:

All the files are text based separated by comma, CVS like files. We made these files easily understandable as you might create external tools to analyze it, like uploading values to the database.
As a CSV file, the first line always contains the names of the columns.

Classes File

Three fields used:

- tagClass – An unique count id for the class
- signature – The JNI signature for the class
- tagClassLoader – The unique object id for the classLoader

Example:

```
tagClass,signature,tagClassLoader
      1,Ljava/io/BufferedWriter;,0
      2,Ljava/util/Collections$ReverseComparator;,0
      3,Ljava/lang/StringCoding$StringDecoder;,0
```

Objects File

Three fields used:

ObjectTag – The Unique Count Id of an object
ClassTag – The Unique Count Id of the declaring class for this object
Size – The size in bytes for this object

Example:

```
objectTag,classTag,size
1338,59,480
1339,106,88
1340,106,88
1341,106,88
```

References Files:

Three fields used:

tagReferrer – Unique CountId for the object holding a reference
tagReferred – Unique CountId for the object referred
index – JVMTI sends this count that I didn't find any usage for this. It's here for future usage.

Example:

```
tagReferrer, tagReferee,index
0,134,0
0,168,0
0,16,0
0,270,0
```

Controllers

This topic explains the purpose of each controller existent in JBossProfile

MBean - JVMTIClass

It controls the life cycle of the profiler.
For example, to extract a snapshot you have a method heapSnapsho(prefix, suffix)

The methods existent at this Bean can be also used directly without the use of JMX.
Refer to org.jboss.profiler.jvmti.JVMTIInterface and you have all the available methods.

org.jboss.profiler.memoryprofiler.engine.MemorySnapshotEngine

This class process the text files, generating a POJO model in memory defined by the package org.jboss.profiler.memoryprofiler.model.

If you want to create a tool to deal with files directly, use this class.

org.jboss.profiler.memoryprofiler.engine.MemorySnapshotController

As a general rule I would say to never solve a reference between entities directly in the POJO model.
For example don't look for the classLoder responsible for the loading of any specific class. For this use solveClassLoaderReference(MemoryClass) from this class.

I'm imposing this caveat as there is a possibility of stop using in memory model to use an in file model by using org.jboss.profiler.util.FileCollection, and I can't have these references serialized directly into those objects and the model might change based on that.

There are two particular functions in this controller:

- filterRoots
  - Maybe we should change this name ☺ But basically what it doest is the main view of loaded classes. It shows a reference between classes and number of objects
- summarizeReferenceByPath(boolean forward,String [] path)
  - You can tell the direction you want to navigate. Forward means from referencer to referencee. If you want to look at who is holding a reference you should use false (backwards).
  - In the path you can use class references (C<ObjectId>) if you want to know all the references, or object reference (O<ObjectId>) if you want to look at references in a specific object. (References to a classLoader for example)

Using the Tool

Installing the Agent

First thing you need to use Java 5 on this as JVMTI is only present on Java 5.

I - You have to modify you Java argument to load the agent. You can do this by:
set JAVA_OPTS=%JAVA_OPTS% -agentlib:jbossAgent

II – You have to install the SAR for JVMTI to have access to the MBean controller.
Do that by copying jboss-profiler-jvmti.sar into your deployment dir.

Using the Tool

III – Use the MBean. Look for:

jboss.profiler

- mbean=JVMTIClass

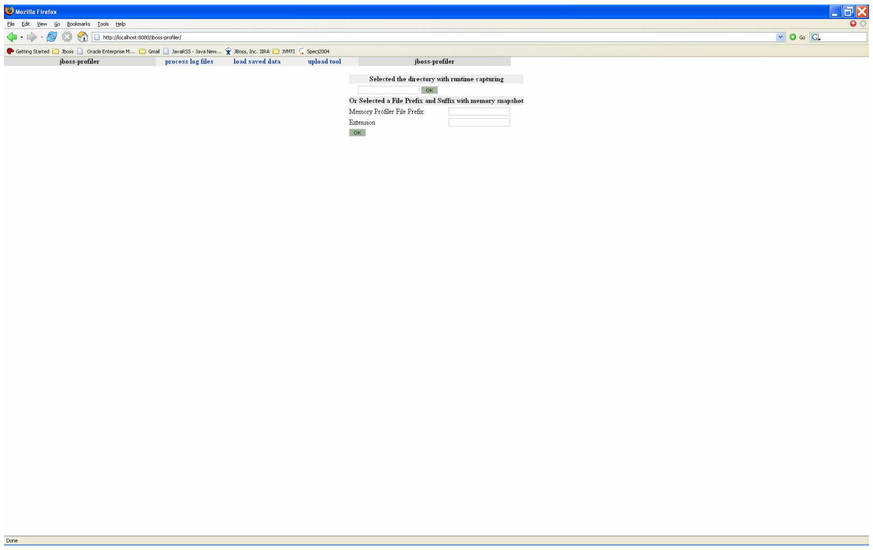
IV – Create the memory snapshot by defining the prefix directory and the suffix.
For example "C:\temp\snap", and "shot" for the suffix (or extension).

Analyze data

You can either use the WEB application or to write a tool that uses MemorySnapshotEngine or MemorySnapshotController.

Important: I don't recommend using the same JVM you are capturing the snapshot to analyze it, as for now we are processing everything in memory and by my short experience this produces three times memory as you had in your VM. So, if you execute a snapshot after analyzed a snapshot you will have a mirror like loop, although if you don't capture snapshots after analyzed snapshots you will be okay using the same VM.

V – If you want to use the WEB application, install jboss-profiler.war in your deploy dir, then:


Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:8080/jboss-profiler/
Getting Started Jboss Oracle Enterprise M... Gmail JavaRSS - Java New... JBoss, Inc. JIRA JVMTI Specj2004
jboss-profiler
process log files
load saved data
upload tool
jboss-profiler
OK
OK
Done

Selected the directory with runtime capturing

Or Selected a File Prefix and Suffix with memory snapshot

Memory Profiler File Prefix

Extension

Use the second form, proving the prefix and suffix for your snapshot.

All you have to do after that is navigate in the model.

Notice that the front-end is really really simple. The intention here was to prove the model's capability and we are looking for contributors (preferably with design skills) to contribute with a better UI.

Features:

- The capability of trace references to classes or objects is a pretty nice feature, hard to find in any other profiler.
- You can introspect references to ClassLoaders through Classes analysis.
- The snapshot capability without a need of any software besides what’s in the VM is also good, you can retrieve a snapshot easily for both runtime (throught JVMPI and TI in the future) and memory.