**Working Log**

**Test Environment:** JDK8 + VisualVM1.4 + JMC5.5+ IntelliJ2017.3

**IntelliJ plugin:** JMH plugin + VisualVM Launcher

**JVisualVM plugin:** Startup Profiler + BTrace Workbench + Visual GC

**JMC(Java mission control):** JFR(Java Flight Recorder)

1. **JVisualVM**

**What is VisualVM?**

It is a visual tool integrating several command line JDK tools and lightweight profiling capabilities.

**How to install JVisualVM?**

Go to <https://visualvm.github.io/> and download version VisualVM 1.4

**What is profiling?**

Gather statistics from your application throughout normal application execution (via byte-code instrumentation of your application’s source). The statistics you’ll get from this profiler will be more accurate and more frequently updated than the data gathered by the sampler.

The flipside you must consider though is that the profiler is a “brute-force” profiler of sorts. It’s instrumentation approach will essentially redefine most of the classes and methods that your application is executing, and as a result is likely to slow down your application significantly. For example, running part of a normal analysis with the application used above, the application finished in about 35 seconds. Turning on VisualVM memory profiler caused the application to finish the same analysis in 31 minutes.

**How to do profiling in VisualVM?**

Step1: Running java code with VisualVM in IntelliJ and VisualVM is opened automatically.

Step2: Click Profiler->CPU

Step3: Go to source code and keep the code running.

Step4: Go back to VisualVM and click “Hot spots” icon.

**How to export csv files?**

Step1: Click “Snapshot” when the code is running.

Step2: Choose “Export Forward Calls” option.

Step3: Name this file and export it, file formate is \*.csv as default.

**What is sampling?**

Sample application periodically for CPU and Memory usage.

1. **JMC**

**What is JFR?**

It is used to produces detailed recordings about the JVM and the application it is running. It is low performance impact, less than 2% for typical Java applications, when collecting data and events.

**How to use JFR?**

Step1: Modify environmental variables and then enter “jmc” in command line to launcher JMC.

Step2: Double click “Flight Recorder” after running java code in IntelliJ.

Step3: Choose “Time fixed recording” or “Continuous recording” mode to sampler data.

**How to analysis sampling results?**

Code: overview | Hot Methods | Call Tree | Exceptions | Compilation | Class Loading

Expand subtree of the method which takes the most percentages of memories.

General

Memory

Threads

I/O

System

Events

1. **JMH**

**What is JMH?**

**How to install JMH in IntelliJ?**

**Test JMH on demo code.**

**Test JMH on the code of awele.**

**Compte rendu:**

**20180315**

1. Save the result of performance of original code as file1 which record running time and rates of memories of functions.

2. Analysis the function which occupied most of the memories and modify it. And then, run it again. Save the new result as file2.

3. Compare the differences between them. Mark functions with green icons if its’ performances are better, mark functions with red icons if it takes much more time than before and mark functions with yellow icons if nothing changed.