Paul English

CS 2420-003

Robert Baird

April 8, 2013

Reflection - VisualVM

Using VisualVM can offer some insights and information that can help you optimize and speed up your Java application. It can be used to find memory leaks, pinpoint bottlenecks in performance, and in general learn about what your application might be doing under the scenes.

When running the application on with the ‘SortTestRun‘ example code we can see some obvious patterns which we might expect. We can see the sleep and run cycles where our code performs a sort, then sleeps for a half-second. We can watch our heap usage over time to view how memory regularly gets used up, and summarily garbage collected in order to prevent errors. We can even see what functions are being invoked, how many times they are invoked, and how much time they’re taking in general. While observing a program using VisualVM there can be noticeable slowdown of your application when enabling profiling of either the CPU performance or the Memory. As noted by VisualVM your code is actually being modified, instrumented, so that the profiling can take place. For me this noticeably increased some of the speeds and memory values that were being observed. The test reported longer times, for either profiling tool.

When running with VisualVM on a different piece of code I found out some interesting things, that may be useful to look into. I have a Java application that scrapes a series of pages pulling out specific data and writing it into database records. It’s long-running since there are a lot of pages that it can scrape, and each page builds several database records before moving onto the next page. While profiling it for CPU performance I found that there was an obvious bottleneck in

a factory function that provides methods with a database connection. It's getting invoked quite a bit and taking a significant amount of time in general. The longest running methods are predictably the methods that make an HTTP connection and download HTML content for scraping. While viewing the monitor pane, I can see that the heap is staying relatively managed, meaning I probably don't have any memory leaks. Quite a bit of my memory is being used by raw byte arrays which are probably being created to house the data being streamed from the HTTP connection. One other useful thing to take not of is the lack of thread activity in this application, which could easily benefit from concurrent activity. The application runs entirely on one thread, though it wouldn't matter if many threads were run at the same time since none of the scraped data relies on any of the other scraped pages.