Monitoring Nashorn Performance

Running The Octane microbenchmarks with Nashorn.

1. Pointing out the right Nashorn jar

As of b82, nashorn is integrated into the JDK, which means that the nashorn.jar that ships with the JDK is picked unless you do one of the following, which you should do before running any nashorn benchmarks if you want to test nashorn code that you have built yourself. If you just want to benchmark the JDK nashorn.jar in your JAVA_HOME, you can omit this step

1) put the built nashorn.jar in $JAVA_HOME/jre/lib/ext, replacing the one that came with the JDK.

or

2) add -Djava.ext.dir=path_to_directory_with_nashorn:jar:$JAVA_HOME/jre/lib/ext to your java command line after "java"

2. Running the octane benchmarks

You have three ways you can run the benchmark suite.

2.1 Using JMH

The best way to run the octane benchmarks with Nashorn is to use the JMH microbenchmark suites where they are already integrated.

2.1.1 Install JMH

Make sure you are on the VPN.

Make sure your maven settings contain proxy information:

```
cat ~/.m2/settings.xml
<settings>
  <proxies>
    <proxy>
      <active>true</active>
      <protocol>http</protocol>
      <host>www-proxy.us.oracle.com</host>
      <port>80</port>
    </proxy>
  </proxies>
</settings>

hg clone http://default:default@sthserver02.se.oracle.com:5000/microbenchmarks
cd microbenchmarks/micros-jdk8
mvn clean install
```

You only need to do this once. Now you can go off the VPN and copy the benchmark directory anywhere.

2.1.2 Running JMH

To run the octane benchmarks and get a long stable run, do:

```
java -jar target/microbenchmarks.jar ".*octane.*" -wi 20 -i 100 -f 5
```

(you can also replace the wildcard with individual octane benchmarks, like java -jar target/microbenchmarks.jar ".*octane.*Box2D.*" -wi 20 -i 100 -f 5)

-wi means warmup iterations. 20 is more than enough, this gives hotspot time to optimize
-i means real iterations.
-f means how many JVM forks should be done, i.e. should each benchmark be started in a new JVM

Low iteration scores when stable are good. They show time per iteration in milliseconds

Modify rerun flags as needed

2.2 Using the "ant octane" job in the Nashorn source

Go to the nashorn make directory

prerequisite: run "ant externals" to download benchmarks and all other third party dependencies

Then run "ant octane" from the nashorn make directory

(you can run individual benchmarks e.g. with ant octane-crypto, ant octane-box2d) too.

The nashorn used will be the one you build from source in the ../dist/ directory from make.

High score means good. (iterations / time unit)

2.3 Using the run-octane.js script, part of the Nashorn sanity tests

Go to the nashorn make directory

prerequisite: run "ant externals" to download benchmarks and all other third party dependencies

```
sh ../bin/jjs ../test/script/basic/run-octane.js -- --verbose --iterations 5
```

(runs five iterations of each benchmark)

individual benchmarks can be run by adding them on the command line, e.g.

```
sh ../bin/jjs ../test/script/basic/run-octane.js -- crypto --verbose --iterations 5
```

or

```
sh ../bin/jjs ../test/script/basic/run-octane.js -- crypto splay deltablue --verbose --iterations 5
```

High score means good. (iterations / time unit)

Please note that 2) and 3) use the octane base.js harness, which has a very primitive and deviation prone way of measuring benchmark time. Benchmarks like splay that have very short iterations will be extremely badly affected and show widely diverging results.