(http://baeldung.com)

Microbenchmarking with Java

Last modified: January 15, 2018

by baeldung (http://www.baeldung.com/author/baeldung/)

Java (http://www.baeldung.com/category/java/)

1. Introduction

This quick article is focused on JMH (the Java Microbenchmark Harness) – which is scheduled to become a part of JVM in the upcoming Java 9 release.

Simply put, JMH takes care of the things like JVM warm-up and code-optimization paths, making benchmarking as simple as possible.

2. Getting Started

To get started, we can actually keep working with Java 8 and simply define the dependencies:

```
<dependency>
    <groupId>org.openjdk.jmh</groupId>
    <artifactId>jmh-core</artifactId>
    <version>1.19</version>
</dependency>
<dependency>
    <groupId>org.openjdk.jmh</groupId>
    <artifactId>jmh-generator-annprocess</artifactId>
    <version>1.19</version>
</dependency>
```

The latest versions of the JMH Core (https://search.maven.org/#artifactdetails%7Corg.openjdk.jmh%7Cjmh-core%7C1.19%7Cjar) and JMH Annotation Processor (https://search.maven.org/#artifactdetails%7Corg.openjdk.jmh%7Cjmh-generator-annprocess%7C1.19%7Cjar) can be found in Maven Central.

Next, create a simple benchmark by utilizing *@Benchmark* annotation (in any public class):

```
@Benchmark
public void init() {
    // Do nothing
}
```

Then we add the main class that starts the benchmarking process:

```
public class BenchmarkRunner {
    public static void main(String[] args) throws Exception {
        org.openjdk.jmh.Main.main(args);
    }
}
```

Now running *BenchmarkRunner* will execute our arguably somewhat useless benchmark. Once the run is complete, a summary table is presented:

```
# Run complete. Total time: 00:06:45
Benchmark      Mode  Cnt Score            Error        Units
BenchMark.init thrpt 200 3099210741.962 ± 17510507.589 ops/s
```

3. Types of Benchmarks

JMH supports some possible benchmarks: *Throughput, AverageTime, SampleTime*, and *SingleShotTime*. These can be configured via *@BenchmarkMode* annotation:

```
@Benchmark
@BenchmarkMode(Mode.AverageTime)
public void init() {
    // Do nothing
}
```

The resulting table will have an average time metric (instead of throughput):

```
# Run complete. Total time: 00:00:40
Benchmark Mode Cnt  Score Error Units
BenchMark.init avgt 20 ≈ 10⁻⁹ s/op
```

4. Configuring Warmup and Execution

By using the *@Fork* annotation, we can set up how benchmark execution happens: the *value* parameter controls how many times the benchmark will be executed, and the *warmup* parameter controls how many times a benchmark will dry run before results are collected, for example:

```
@Benchmark
@Fork(value = 1, warmups = 2)
@BenchmarkMode(Mode.Throughput)
public void init() {
    // Do nothing
}
```

This instructs JMH to run two warm-up forks and discard results before moving onto real timed benchmarking.

Also, the *@Warmup* annotation can be used to control the number of warmup iterations. For example, *@Warmup(iterations = 5)* tells JMH that five warm-up iterations will suffice, as opposed to the default 20.

5. State

Let's now examine how a less trivial and more indicative task of benchmarking a hashing algorithm can be performed by utilizing *State*. Suppose we decide to add extra protection from dictionary attacks on a password database by hashing the password a few hundred times.

We can explore performance impact by using a *State* object:

```
@State(Scope.Benchmark)
public class ExecutionPlan {

    @Param({ "100", "200", "300", "500", "1000" })
    public int iterations;

    public Hasher murmur3;

    public String password = "4v3rys3kur3p455w0rd";

    @Setup(Level.Invocation)
    public void setUp() {
        murmur3 = Hashing.murmur3_128().newHasher();
    }
}
```

Our benchmark method then will look like:

```
@Fork(value = 1, warmups = 1)
@Benchmark
@BenchmarkMode(Mode.Throughput)
public void benchMurmur3_128(ExecutionPlan plan) {

    for (int i = plan.iterations; i > 0; i--) {
        plan.murmur3.putString(plan.password, Charset.defaultCharset());
    }

    plan.murmur3.hash();
}
```

Here, the field *iterations* will be populated with appropriate values from the *@Param* annotation by the JMH when it is passed to the benchmark method. The *@Setup* annotated method is invoked before each invocation of the benchmark and creates a new *Hasher* ensuring isolation.

When the execution is finished, we'll get a result similar to the one below:

```
# Run complete. Total time: 00:06:47

Benchmark                   (iterations)   Mode  Cnt      Score      Error  Units
BenchMark.benchMurmur3_128           100  thrpt   20  92463.622 ± 1672.227  ops/s
BenchMark.benchMurmur3_128           200  thrpt   20  39737.532 ± 5294.200  ops/s
BenchMark.benchMurmur3_128           300  thrpt   20  30381.144 ±  614.500  ops/s
BenchMark.benchMurmur3_128           500  thrpt   20  18315.211 ±  222.534  ops/s
BenchMark.benchMurmur3_128          1000  thrpt   20   8960.008 ±  658.524  ops/s
```

5. Conclusion

This tutorial focused on and showcased Java's micro benchmarking harness.

As always, code examples can be found on GitHub (https://github.com/eugenp/tutorials/tree/master/jmh).

CATEGORIES

SPRING (HTTP://WWW.BAELDUNG.COM/CATEGORY/SPRING/)
REST (HTTP://WWW.BAELDUNG.COM/CATEGORY/REST/)