JMH + perf

Tran Tuan Hiep

Microbenchmark

Benchmark of a small isolated component or just a method

Similar to unit test

Benchmarking in Java

- Benchmarking is complicated
  - compiler optimizations, hw optimizations
- In java is even more complicate due to GC and JIT compilation
  - JVM can perform optimisations on the benchmark, that cannot be applied in production

JMH

Java Microbenchmark Harness

- Java framework for writing microbenchmarks
- Spares the pain of writing the whole benchmark
  - Generates the code doing the measuring around the code you want to measure
- Part of JDK
- Pluggable profiler architecture
- Various output format, support for JSON and CSV
- Compatible with java build system such as maven and gradle,
- Written by JVM developers

Writing benchmark in JMH

- Annotate the code with @Benchmark
- Further customisation with following annotations
  - @BenchmarkMode
    - 4 supported modes
  - @OutputTimeUnit, @Measurement, @Fork ..

Code example - logarithm computation

https://github.com/honzatran/jmh-presentation-examples

Running JMH benchmarks

- Project with multiple JMH benchmarks
- Build it with gradle “./gradlew jmhJar”
- The output of the build is jar
- You can execute the benchmark using

“java -jar build/libs/benchmark_jmh.jar [arguments]”

Internals of benchmark execution

- main process forks a benchmarking process, which runs the single benchmark in a specified benchmark mode
- main process repeats this forking n times
- number of forks can be specified by
  - @Fork
  - -f option when executing the jar

Forked benchmark process

- Runs a Trial in multiple threads
- Trial is a sequence of warmup and measurement iterations
- Iteration
    - Invokes a benchmark method (invocation) for the iteration period of time

- **Warmup**
  - used for warming up the code
  - e.g. let JIT optimizations kick in, warmup instruction and data cache
  - use @Warmup annotation to set up the iteration count and iteration period
  - alternative use -wi to set up warmup iteration and -w iteration period time

- **Measurement**
  - The real performance measurement
  - use @Measurement annotation to set up
  - alternatively use -i to set the number of iteration and -r to set the iteration time

Benchmark modes

- annotation @BenchmarkMode(Mode) sets the default mode
  - AverageTime
  - Throughput
  - SampleTime
  - SingleShotTime

Benchmark state

- Class that is passed to the benchmark method as an argument
- Annotation @State(Scope)
- Different level of scope - Sharing instances
  - Benchmark, Group, Thread

Set up and tear down method of a State

- Like before and after method in unit test for states
- Annotations @Setup(Level), @TearDown(Level)
- Level of method = when the method is invoked
  - Trial, Iteration, Invocation
  - Pass as an argument to annotations

Parameter

- Benchmark method depends on a parameter
- Use @Param annotation to inject a value into a non-final public member in state
    - Benchmark can use this member of the state
- Must be primitive, Enum or String
- Must have a default value

Code example - hash maps

java -jar build/libs/benchmark_jmh.jar .*Hashmap.*

Concurrent benchmarks

- JMH simplifies writing of concurrent benchmarks
- @Group annotation
  - every benchmark annotated with the same group is run simultaneously
- @GroupThread
  - number of threads executing the benchmark

Lock free queue benchmark

java -jar build/libs/benchmark_jmh.jar .*Multithreaded.*

Benchmarking pitfalls

- JMH is not a silver bullet
- Can’t fix badly written benchmark
  - method annotated with @Benchmark
- But has tools, that help writing these methods
  - Blackhole, CompilerControl

Dead code elimination example

java -jar build/libs/benchmark_jmh.jar .*BadlyWritten.*

Dead code elimination

- return value from benchmark or use blackhole

Profilling jmh benchmarks

JMH profilers

- pluggable profiler architecture
- use -lprof to get the list of profilers
- use -prof to run the benchmark with a profiler
- particularly useful integration with perf,
  - available only on linux

Perf with JMH

- JMH has 3 perf profilers currently
  - perf
    - hw counters
  - perfnorm
    - normalised hw counters per invocation(operation)
  - perfasm
    - assembly level profiler

Hashmap example perf hw counters and gc

java -jar build/libs/benchmark_jmh.jar .*Hashmap.* -prof perform -p size=1000000

Perfasm

- It’s necessary to install hsdis at first, which is part of jdk
- use following git repository
  - https://github.com/jkubrynski/profiling
  - copy the hsdis library to $JAVA_HOME/jre/lib/<architecture>/server/

Atomic and ArrayCopy example perfasm

java -jar build/libs/benchmark_jmh.jar .*Atomic.* -prof perform

java -jar build/libs/benchmark_jmh.jar .*Array.* -prof perform

End