JMH Cheatsheet

Java Microbenchmark Harness

`@Benchmark:`
Annotates the benchmark method. JMH will produce the benchmark code for this method during compilation. demarcates the benchmark payload, and JMH treats it specifically as the wrapper which contains the benchmark code.

`@BenchmarkMode:`
Defines the mode in which this benchmark will run.

`Mode.Throughput: (ops/time)`
Counts the total throughput over all worker threads. This mode is time-based, and it will run until the iteration time expires.

`Mode.AverageTime: (time/op)`
Counts the average time to call over all worker threads. This is the inverse of Mode.Throughput.

`Mode.SampleTime:`
Runs by continuously calling the benchmark methods, and randomly samples the time needed for the call. It will run until the iteration time expires and will provide a detailed view on percentiles at the end.

`Mode.SingleShotTime:`
Measures the time for a single operation. Useful to estimate the "cold" performance, when you don't want to hide the warmup invocations. Warning: Timers overhead might be significant, if benchmarks are small; switch to SampleTime in that case.

`@State:`
State objects encapsulate the state on which the benchmark is working on. The Scope of state object defines to which extent it is shared among the worker threads. States can be injected into benchmarks methods as arguments, and also on @Setup and @TearDown methods.

`Scope.Benchmark:`
When scope is benchmark, all instances of the same type will be shared across all worker threads.

`Scope.Thread:`
When scope is thread, all instances of the same type are distinct, even if multiple state objects are injected in the same benchmark.

`Scope.Group:`
When scope is group, all instances of the same type will be shared across all threads within the same group. Each thread group will be supplied with its own state object.

`@Setup:`
Fixture to run *before* the benchmark. Can only be declared in classes annotated with @State. You may optionally provide a Level.

`@TearDown:`
Fixture to run *after* the benchmark. Same properties as the @Setup annotation.

`Level.Trial:`
Trial is the set of benchmark iterations.

`Level.Iteration:`
Iterations is the set of benchmark invocations.

`Level.Invocation: (Dangerous)`
A benchmark method execution. Only usable for benchmarks taking more than 1 millisecond per single benchmark method invocation. Check the JavaDocs for more details.

`@Threads:`
Default number of threads to run. Can be put at a benchmark method to have effect on that method only, or at class level to have effect on all benchmarks of that class. May be overridden with runtime options.

`@Warmup, @Measurement:`
Allow to define the number of iterations, time, time unit and batch size via annotations. These can also be set via command line or in the main method via an Options object.

`@Fork:`
Allows to set forking parameters for the benchmark.

- `value` = # of times to fork;
- `jvmArgs` = JVM args to replace in the command line. Useful for external profilers;
- `jvm` = JVM executable to run with. Useful for testing with different JVMs.

Check the JavaDoc for more details.