Java Microbenchmarking Harness

- bierzemy z mavena

- mozna oznaczac metody do pomiaru

- robimy project gdzie wywoływane wykonania

- tworzymy artefakt, można też do proj istniejącego – tu opis

<http://java-performance.info/jmh/>

- odpalamy –

java -jar target/benchmarks.jar

- jest skanowanie przez klasy i odpalane z adnotacjami

- trzeba powyłączać inne aplikacje na czas odpalenia, co cpu zajmują, by były dobre/równe warunki

- rozne mode

|  |  |
| --- | --- |
| Throughput | Measures the number of operations per second, meaning the number of times per second your benchmark method could be executed. |
| Average Time | Measures the average time it takes for the benchmark method to execute (a single execution). |
| Sample Time | Measures how long time it takes for the benchmark method to execute, including max, min time etc. |
| Single Shot Time | Measures how long time a single benchmark method execution takes to run. This is good to test how it performs under a cold start (no JVM warm up). |
| All | Measures all of the above. |

* Pierwszy to default
* Okreslamy @BenchmarkMode(Mode.Throughput)
* Możliwe okreslenie unit czas wypisu

@OutputTimeUnit(TimeUnit.MINUTES)

NANOSECONDS

MICROSECONDS

MILLISECONDS

SECONDS

* Scope - @State(Scope.Thread) – kod dodatkowy dla JMH

To może być parametr początkowy na wszystkie testy

@State(Scope.Thread)

public static class MyState {

public int a = 1;

public int b = 2;

public int sum ;

}

@Benchmark @BenchmarkMode(Mode.Throughput) @OutputTimeUnit(TimeUnit.MINUTES)

public void testMethod(MyState state) {

state.sum = state.a + state.b;

}

* State object moze być reuzyty między kolejne wywołania, okreslamy scope

|  |  |
| --- | --- |
| Thread | Each thread running the benchmark will create its own instance of the state object. |
| Group | Each thread group running the benchmark will create its own instance of the state object. |
| Benchmark | All threads running the benchmark share the same state object. |

* A JMH state class must obey the following rules:
* The class must be declared public
* If the class is a nested class, it must be declared static (e.g. public static class ...)
* The class must have a public no-arg constructor (no parameters to the constructor).

**State Object @Setup and @TearDown**

* Metody które przygotowują I czyszczą parametry dla wywołania – przed/po
* - ich czas nie jest liczony
* Metody umieszczane są w obiekcie state
* @Setup(Level.Trial)
* public void doSetup() {
* sum = 0;
* System.out.println("Do Setup");
* }
* @TearDown(Level.Trial)
* public void doTearDown() {
* System.out.println("Do TearDown");
* }
* Są levele mówiące kiedy metoda ma być wywołana

|  |  |
| --- | --- |
| Level.Trial | The method is called once for each time for each full run of the benchmark. A full run means a full "fork" including all warmup and benchmark iterations. |
| Level.Iteration | The method is called once for each iteration of the benchmark. |
| Level.Invocation | The method is called once for each call to the benchmark method. |

* Jak nie wiemy kiedy wywołane, to println i widać po wypisie

Problemy:

- może być dodatkowa optymalizacja dla wykonania, co może nie nastapić w docelowej aplikacji

- unikanie petli w metodach benchmark

- może być przy optymalizacji usuwany kod ktory nie jest używany, np. Wyliczenia/zmienne nigdy nie uzyte, co może spowodować bezuzytecznośc testu – dead code removed

- konstant holding – jak w kodzi coś co zawsze daje to samo na ustawionych to może zo w wynik dac by przyspieszyć

<http://javapapers.com/java/java-micro-benchmark-with-jmh/>

Beanchmark do porównywania, micro – kawałki kodu w systemie, macro – miedzy systemami

- Stage are, benchmark design, benchmark code implementation, benchmark test execution and results interpretation

-  [Caliper](https://github.com/google/caliper) by Google and [JMH](http://openjdk.java.net/projects/code-tools/jmh/) by OpneJDK

- w setup mozna np. We tablice uzupełniać

- wazne ustawienie ile rozgrzeń i ile odpaleń

<http://java-performance.info/jmh/>

@BenchmarkMode(Mode.Throughput)

@OutputTimeUnit(TimeUnit.SECONDS)

@Warmup(iterations = 10, time = 1, timeUnit = TimeUnit.SECONDS)

@Measurement(iterations = 10, time = 1, timeUnit = TimeUnit.SECONDS)

@State(Scope.Thread)

@Fork(2)

- mozna też threads

@Threads(1) // doczytać

---

- mozna zaznaczyc co chcemy by było robione przez kompilator z nie uzywanymi metodami - CompilerControl.Mode

- parametry

All you need to do is:

1. Define a @State object
2. Define all your parameters fields in it
3. Annotate each of these fields with @Param annotation

- dla multithread - @State(Scope.Benchmark)

- grupy wywolań

In order to setup a group of tests, you need:

1. Mark all your test methods with @Group(name) annotation, providing the same string name for all tests in a group (otherwise these tests will be run independently – no warning will be given!).
2. Annotate each of your tests with @GroupThreads(threadsNumber) annotation, specifying a number of threads which will run the given method.

-

To see all the options you have available use the command:  
  
java - jar benchmark.jar -h  
  
One of the most useful options is to run with the benchmarks with a profiler.  To list the available profilers on your system (for example perf is only available on Unix) use the command.  
  
java -jar benchmark.jar -lprof  
  
To run the test with a simple stack profiler (available on all systems) use  
  
java -jar benchmark.jar -prof stack

Benchmark w repo

https://cedt-icg-bitbucket.nam.nsroot.net/bitbucket/projects/ECOMER/repos/mercury-benchmarks/browse/universal-translator

przykład:

https://cedt-icg-bitbucket.nam.nsroot.net/bitbucket/projects/ECOMER/repos/mercury-benchmarks/browse/universal-translator/fix-to-json/src/test/java/com/citi/mercury/benchmark/TranslationBenchmark.java

example :

**import** org.apache.commons.io.IOUtils;  
**import** org.openjdk.jmh.annotations.\*;  
**import** org.openjdk.jmh.runner.Runner;  
**import** org.openjdk.jmh.runner.RunnerException;  
**import** org.openjdk.jmh.runner.options.Options;  
**import** org.openjdk.jmh.runner.options.OptionsBuilder;  
**import** org.springframework.core.io.ClassPathResource;  
**import** org.springframework.core.io.Resource;  
**import** org.springframework.messaging.support.MessageBuilder;  
  
**import** java.util.Map;  
**import** java.util.concurrent.TimeUnit;  
  
@BenchmarkMode(Mode.***Throughput***)  
@OutputTimeUnit(TimeUnit.***SECONDS***)  
@Warmup(iterations = 10, time = 1, timeUnit = TimeUnit.***SECONDS***)  
@Measurement(iterations = 10, time = 1, timeUnit = TimeUnit.***SECONDS***)  
@State(Scope.***Thread***)  
@Fork(2)  
**public class** TranslationBenchmark {  
  
 **private** QuickfixjMessageMapTransformer **quickfixjMessageMapTransformer**;  
  
 **private** SelectAllValuesFromMapWithSplitTransformer **selectAllValuesFromMapWithSplitTransformer**;  
  
 **private** String **message**;  
  
  
 **public static void** main(String[] args) **throws** RunnerException {  
 Options opt = **new** OptionsBuilder()  
 .include(TranslationBenchmark.**class**.getSimpleName())  
 .jvmArgs(**"-ea"**)  
 .build();  
  
 **new** Runner(opt).run();  
 }  
  
 @Setup  
 **public void** setup() **throws** Exception {  
 **quickfixjMessageMapTransformer** = **new** QuickfixjMessageMapTransformer(**"FIX50SP2.xml"**);  
 **selectAllValuesFromMapWithSplitTransformer** = **new** SelectAllValuesFromMapWithSplitTransformer();  
 Resource resource = **new** ClassPathResource(**"com/citi/mercury/fix/transformer/FixMessage.txt"**);  
 **message** = IOUtils.*toString*(resource.getURI(), **"UTF-8"**);  
 }  
  
 @Benchmark  
 @Threads(1)  
 **public** Map<String, Map<String, Object>> transormToMap\_1Thread() **throws** Exception {  
 **return quickfixjMessageMapTransformer**.transformMap(**message**);  
 }  
  
 @Benchmark  
 @Threads(1)  
 **public** Object transormToMapWithSelectorAndSplitter\_1Thread() **throws** Exception {  
 **return selectAllValuesFromMapWithSplitTransformer**.splitMessage(MessageBuilder.*withPayload*(**quickfixjMessageMapTransformer**.transformMap(**message**)).build());  
 }

}

Maven

*<!-- https://mvnrepository.com/artifact/org.openjdk.jmh/jmh-core -->*<**dependency**>  
 <**groupId**>org.openjdk.jmh</**groupId**>  
 <**artifactId**>jmh-core</**artifactId**>  
 <**version**>1.17.5</**version**>  
</**dependency**>  
<**dependency**>  
 <**groupId**>org.openjdk.jmh</**groupId**>  
 <**artifactId**>jmh-generator-annprocess</**artifactId**>  
 <**version**>1.17.5</**version**>  
</**dependency**>

<**build**>  
 <**plugins**>  
 <**plugin**>  
 <**groupId**>org.codehaus.mojo</**groupId**>  
 <**artifactId**>exec-maven-plugin</**artifactId**>  
 <**executions**>  
 <**execution**>  
 <**id**>run-benchmarks</**id**>  
 <**phase**>integration-test</**phase**>  
 <**goals**>  
 <**goal**>exec</**goal**>  
 </**goals**>  
 <**configuration**>  
 <**classpathScope**>test</**classpathScope**>  
 <**executable**>java</**executable**>  
 <**arguments**>  
 <**argument**>-classpath</**argument**>  
 <**classpath** />  
 <**argument**>org.openjdk.jmh.Main</**argument**>  
 <**argument**>.\*</**argument**>  
 </**arguments**>  
 </**configuration**>  
 </**execution**>  
 </**executions**>  
 </**plugin**>  
 </**plugins**>  
</**build**>