Java Performance - TP2

October 23, 2018

Thomas COUCHOUD
thomas.couchoud@etu.univ-tours.fr
Victor COLEAU
victor.coleau@etu.univ-tours.fr

Chapter 1

Etude de JMH

1.1 Annotations

JMH fonctionne principalement avec des annotations. Parmi celles-ci nous retrouvons notamment:

- **Benchmark**: Permet de définir une méthode comme étant une méthode à tester
- **BenchmarkMode**: Permet d'indiquer quelles métriques nous voulons étudier, nous avons:
  - Throughput: Mesure le nombre d'opérations par seconde
  - Average Time: Mesure le temps moyen d'exécution de la méthode
  - Sample Time: Mesure les temps d'exécution de la méthode: min, max, ...
  - Single Shot Time: Mesure le temps d'exécution pour un run unique du benchmark. Cela peut être utile pour voir la performance de la méthode sans le hotspot.
- **OutputTimeUnit**: Permet de définir les unités de temps utilisées pour les rapports
- **Fork**: Permet de définir le nombre de JVM qui sont forkées. En effet par défaut la JVM est forkée à chaque "trial" de la méthode à benchmarker, cependant dans certains cas il peut être utile de changer ce comportement. Il faut tout de même faire attention lors de l'utilisation de cette annotation car des comportements anormaux peuvent apparaître (si l'un a deux classes implémentant la même interface par exemple, en effet dans ce cas la première sera plus rapide car la JVM remplace les appels méthodes).
- **Group**: Cette annotation permet de définir à quel groupe appartient le test. Cela permet de lancer tous les tests d'un même groupe en même temps (voir annotation suivante). Par défaut chaque méthode est dans un groupe unique.
- **GroupThread**: Définit le nombre de threads qui sera utilisé pour lancer la méthode. Dans le cas d'un groupe il y aura un nombre de threads égal à la somme de tous les GroupThreads du groupe.
- **Measurement**: Permet de définir les valeurs par défaut pour le benchmark de la méthode:
  - Nombre d'itérations
  - Temps de mesure
  - Taille d'un batch
- **State**: Annotation à mettre sur une classe afin de définir dans quel état cette dernière sera utilisée:
  - Thread: Valeur par défaut. Une instance de cette classe sera utilisée pour chaque thread qui exécute le benchmark.
  - Benchmark: Une instance unique sera partagée entre tous les threads qui exécutent le même benchmark. Peut être utile dans le cas de méthodes exécutées en parrallèle.
  - Group: Une instance sera allouée pour chaque groupe.
- **TearDown / Setup**: Cette annotation permet de marquer les méthodes à lancer avant et après les benchmark. L'appel à ces méthodes peuvent être précisés grâce a une paramètre de l'annotation:
  - Trial: Valeur par défaut. La méthode est appelée après/avant un benchmark entier.
  - Iteration: La méthode est appelée avant/après une itération (plusieurs invocations).

  - Invocation: La méthode est appelée avant/après une invocation (chaque run).

- **Timeout**: Définit le temps maximum que le benchmark peut prendre.
- **Warmup**: Définit les paramètres pour la phase de chauffe. Similaire à Measurement mais concerne juste la phase de chauffe.
- **CompilerControl**: Définit les options de compilation lors d'un fork de la JVM pour le benchmark de cette méthode. Cependant il faut faire attention car le compilateur peut ignorer ces derniers.

1.2 BalckHole

Le BlackHole permet de consommer, sans donner d'informations si la valeur est réellement utilisée ou non à JIT (compilation à la volée), des variables ou bien du temps CPU.

Dans le cas de variables cela est utile car cela évite que le compilateur réalise des optimisations où la variable ne serait pas calculée.

1.3 Résultats

Voici la sortie pour un benchmark:

Listing 1.1 – out.log

```
# JMH version: 1.21
# VM version: JDK 1.8.0_171, OpenJDK 64-Bit Server VM, 25.171-b11
# VM invoker: /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java
# VM options: -Dfile.encoding=UTF-8
# Warmup: 5 iterations, 10 s each
# Measurement: 5 iterations, 10 s each
# Timeout: 10 min per iteration
# Threads: 1 thread, will synchronize iterations
# Benchmark mode: Throughput, ops/time
# Benchmark: fr.polytechtours.javaperformance.tp2.MyBenchmark.fact

# Run progress: 0.00% complete, ETA 00:16:40
# Fork: 1 of 5
# Warmup Iteration 1: 84603912.755 ops/s
# Warmup Iteration 2: 82896037.779 ops/s
# Warmup Iteration 3: 97368528.704 ops/s
# Warmup Iteration 4: 97875393.382 ops/s
# Warmup Iteration 5: 97872525.925 ops/s
Iteration  1: 97819325.439 ops/s
Iteration  2: 98012719.194 ops/s
Iteration  3: 97591403.605 ops/s
Iteration  4: 97814943.829 ops/s
Iteration  5: 96757314.458 ops/s

# Run progress: 10.00% complete, ETA 00:15:06
# Fork: 2 of 5
# Warmup Iteration 1: 85290497.139 ops/s
# Warmup Iteration 2: 82777393.661 ops/s
# Warmup Iteration 3: 97806785.254 ops/s
# Warmup Iteration 4: 97951394.470 ops/s
# Warmup Iteration 5: 97921213.373 ops/s
Iteration  1: 97939927.352 ops/s
Iteration  2: 97868584.771 ops/s
Iteration  3: 98095783.580 ops/s
Iteration  4: 98099712.187 ops/s
Iteration  5: 97887580.667 ops/s

# Run progress: 20.00% complete, ETA 00:13:24
# Fork: 3 of 5
# Warmup Iteration 1: 85419408.056 ops/s
# Warmup Iteration 2: 83225598.451 ops/s
# Warmup Iteration 3: 97896644.463 ops/s
# Warmup Iteration 4: 97752422.955 ops/s
# Warmup Iteration 5: 97583200.816 ops/s
```

```
45 Iteration  1: 97456878.760 ops/s
46 Iteration  2: 97658458.826 ops/s
47 Iteration  3: 97869433.122 ops/s
48 Iteration  4: 97825488.380 ops/s
49 Iteration  5: 97631526.478 ops/s
50
51 # Run progress: 30.00% complete, ETA 00:11:43
52 # Fork: 4 of 5
53 # Warmup Iteration 1: 85280330.363 ops/s
54 # Warmup Iteration 2: 83234718.351 ops/s
55 # Warmup Iteration 3: 97957582.498 ops/s
56 # Warmup Iteration 4: 97359323.794 ops/s
57 # Warmup Iteration 5: 97928819.893 ops/s
58 Iteration  1: 97942649.273 ops/s
59 Iteration  2: 97913140.638 ops/s
60 Iteration  3: 98081409.370 ops/s
61 Iteration  4: 97774009.651 ops/s
62 Iteration  5: 97438788.773 ops/s
63
64 # Run progress: 40.00% complete, ETA 00:10:03
65 # Fork: 5 of 5
66 # Warmup Iteration 1: 85336199.983 ops/s
67 # Warmup Iteration 2: 83213631.656 ops/s
68 # Warmup Iteration 3: 97838773.067 ops/s
69 # Warmup Iteration 4: 97956246.539 ops/s
70 # Warmup Iteration 5: 98024030.802 ops/s
71 Iteration  1: 98029100.103 ops/s
72 Iteration  2: 97628740.314 ops/s
73 Iteration  3: 97603188.114 ops/s
74 Iteration  4: 97926136.932 ops/s
75 Iteration  5: 98051650.436 ops/s
76
77
78 Result "fr.polytechtours.javaperformance.tp2.MyBenchmark.fact":
79   97788715.770 +-(99.9%) 216107.541 ops/s [Average]
80   (min, avg, max) = (96757314.458, 97788715.770, 98099712.187), stdev = 288497.384
81   CI (99.9%): [97572608.229, 98004823.311] (assumes normal distribution)
```

On retrouve entre les lignes 1 à 10 les différentes informations concernant le test. Le reste du fichier (L12-75) contient tous les run. A la fin (L78-81) nous obtenons les résultats finaux.

1.3.1 Informations

On retrouve les informations concernant la JVM (version JDK, version VM, options, ...) ainsi que la version de JMH. De plus nous avons les paramètres de JMH concernant le nombre d'itérations et le temps pour les phases de chauffe et de test (voir les annotations Measurement et Warmup). Les valeurs des annotations Timeout, BenchmarkMode, Thread sont aussi présentes.

1.3.2 Runs

Pour chaque run on nous indique la progression totale du benchmark ainsi qu'une estimation du temps restant. Suit le numéro du run pour la méthodes testée. Puis on retrouve les différentes itérations du warmup avec les valeurs demandées du benchmark pour l'itération donnée. De la même façon suivent les itérations de test.

1.3.3 Résultats

Les résultats nous affichent (dans notre cas) le nombre d'opérations moyen par seconde avec un taux de fiabilité. De plus les valeurs min et max sont aussi présentes avec leur ecart-type.