TP 2 : JMH et les micro-benchmarks

1 Objectifs

L'objectif de ce TP est de mettre en place des micro-benchmarks à l'aide d'un framework spécialisé : JMH.

2 Matériels et logiciels nécessaires

Ces travaux pratiques se déroulent sur système GNU/Linux. Si vous travaillez sur votre machine personnelle, il est de votre responsabilité d'y installer et configurer tous les logiciels nécessaires. Les machines de l'université fonctionnent sous la distribution ubuntu. Vous pouvez travailler sur d'autres distributions si vous le souhaitez. Les versions du JDK que nous utilisons sont les versions 8, 9 et 11.

3 Évaluation

L'évaluation de chaque TP s'effectue via compte-rendu écrit, seul ou en binôme, des travaux effectués durant la séance de TP. Ce compte-rendu doit être fourni au format PDF pour chacun des TP. Vous pouvez utiliser tous les outils que vous souhaitez pour mettre en forme votre compte-rendu : Word, Latex, LibreOffice, Google doc... Le PDF doit être nommé comme ceci : **TPX_nomEtudiant1_nomEtudiant2.pdf**.

Dans le cas où le compte rendu PDF est accompagné d'autres documents (code, etc...), votre travail devra être rendu sous la forme d'un fichier zip sous le nom **TPX_nomEtudiant1_nomEtudiant2.zip**. Un fichier README dans le zip pour m'aider à comprendre le contenu de votre zip peut être le bienvenue.

L'objectif du compte rendu est de montrer que vous avez compris ce que vous faisiez, et que vous êtes capables d'avoir un regard critique. Prendre du recul sur les différents outils en allant plus loin que la simple description de ceux-ci est très important. Votre compte rendu doit ppouvoir être compris par n'importe qui. De plus, si vous connaissez ou si vous rencontrez d'autres outils qui ne sont pas cités dans le cours, n'hésitez pas à en parler aussi.

Le fond est important, mais la forme l'est tout autant. Votre compte rendu doit être "propre" et agréable à lire. Il peut contenir des screenshots, des morceaux de code ou tout ce qui peut aider à étayer vos propos. Une table des matières, un brève introduction, des parties clairement définies ainsi qu'une conclusion seraient les bienvenues.

Concernant les délais, le compte-rendu doit être rendu au plus tard une semaine après la dernière séance du TP sur la plateforme Celene dédiée à ce cours. La date limite sera aussi précisé sur le cours. Le mot de passe vous sera fourni lors des séances.

Bien entendu, je serai présent lors des TP, n'hésitez donc pas à me poser des questions si vous en avez besoin. Vous pouvez aussi me contacter par mail (florent.clarret@univ-tours.fr) entre deux séances en cas de problème.

Il n'existe pas une seule et unique correction possible pour ce TP. De ce fait, aucune correction ne vous sera fournie. Cependant, si vous souhaitez avoir des informations/précisions sur votre compte rendu, n'hésitez pas à me contacter soit par mail soit directement pendant les séances suivantes pour que nous puissions en discuter.

TP 2 : JMH et les micro-benchmarks

1.1 Génération et lancement du projet JMH

La méthode la plus simple afin de créer un projet JMH est de passer par maven :

```
[batman@gotham:~/tmp] : mvn archetype:generate -DinteractiveMode=false -DarchetypeGroupId=org.openjdk.jmh
    -DarchetypeArtifactId=jmh-java-benchmark-archetype -DgroupId=fr.polytechtours.javaperformance.tp2 -DartifactId=jmh
    -Dversion=1.0.0
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] >>> maven-archetype-plugin:3.0.1:generate (default-cli) > generate-sources @ standalone-pom >>>
[INFO]
[INFO] <<< maven-archetype-plugin:3.0.1:generate (default-cli) < generate-sources @ standalone-pom <<<
[INFO]
[INFO] --- maven-archetype-plugin:3.0.1:generate (default-cli) @ standalone-pom ---
[INFO] Generating project in Batch mode
[INFO] Archetype [org.openjdk.jmh:jmh-java-benchmark-archetype:1.19] found in catalog remote
[INFO] ----------------------------------------------------------------------------
[INFO] Using following parameters for creating project from Archetype: jmh-java-benchmark-archetype:1.19
[INFO] ----------------------------------------------------------------------------
[INFO] Parameter: groupId, Value: fr.polytechtours.javaperformance.tp2
[INFO] Parameter: artifactId, Value: jmh
[INFO] Parameter: version, Value: 1.0.0
[INFO] Parameter: package, Value: fr.polytechtours.javaperformance.tp2
[INFO] Parameter: packageInPathFormat, Value: fr/polytechtours/javaperformance/tp2
[INFO] Parameter: package, Value: fr.polytechtours.javaperformance.tp2
[INFO] Parameter: version, Value: 1.0.0
[INFO] Parameter: groupId, Value: fr.polytechtours.javaperformance.tp2
[INFO] Parameter: artifactId, Value: jmh
[INFO] Project created from Archetype in dir: /Users/batman/tmp/jmh
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 8.914 s
[INFO] Finished at: 2017-10-15T09:47:25+02:00
[INFO] Final Memory: 14M/152M
[INFO] ------------------------------------------------------------------------
```

Cela aura pour effet de vous créer un projet maven standard contenant tout ce qu'il faut afin de lancer des tests avec JMH. Il vous faut maintenant développer les benchmarks en eux-mêmes.

Comme pour tout projet maven, vous pouvez construire le jar à l'aide de la commande suivante :

```
[batman@gotham:~/tmp/jmh] : mvn clean install
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building JMH benchmark sample: Java 1.0.0
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ jmh ---
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ jmh ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /Users/batman/tmp/jmh/src/main/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ jmh ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to /Users/batman/tmp/jmh/target/classes
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ jmh ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /Users/batman/tmp/jmh/src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ jmh ---
```

```
[INFO] No sources to compile
[INFO]
[INFO] --- maven-surefire-plugin:2.17:test (default-test) @ jmh ---
[INFO] No tests to run.
[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ jmh ---
[INFO] Building jar: /Users/batman/tmp/jmh/target/jmh-1.0.0.jar
[INFO]
[INFO] --- maven-shade-plugin:2.2:shade (default) @ jmh ---
[INFO] Including org.openjdk.jmh:jmh-core:jar:1.19 in the shaded jar.
[INFO] Including net.sf.jopt-simple:jopt-simple:jar:4.6 in the shaded jar.
[INFO] Including org.apache.commons:commons-math3:jar:3.2 in the shaded jar.
[INFO] Replacing /Users/batman/tmp/jmh/target/benchmarks.jar with /Users/batman/tmp/jmh/target/jmh-1.0.0-shaded.jar
[INFO]
[INFO] --- maven-install-plugin:2.5.1:install (default-install) @ jmh ---
[INFO] Installing /Users/batman/tmp/jmh/target/jmh-1.0.0.jar to
    /Users/batman/.m2/repository/fr/polytechtours/javaperformance/tp2/jmh/1.0.0/jmh-1.0.0.jar
[INFO] Installing /Users/batman/tmp/jmh/pom.xml to
    /Users/batman/.m2/repository/fr/polytechtours/javaperformance/tp2/jmh/1.0.0/jmh-1.0.0.pom
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 3.793 s
[INFO] Finished at: 2017-10-15T09:53:43+02:00
[INFO] Final Memory: 22M/149M
[INFO] ------------------------------------------------------------------------
```

Enfin, vous pouvez lancer les benchmarks à l'aide de la commande suivante (peut varier selon les paramètres précisés lors de la génération du projet) :

```
[batman@gotham:~/tmp/jmh] : java -jar target/benchmarks.jar
# JMH version: 1.19
# VM version: JDK 1.8.0_121, VM 25.121-b13
# VM invoker: /Library/Java/JavaVirtualMachines/jdk1.8.0_121.jdk/Contents/Home/jre/bin/java
# VM options: <none>
# Warmup: 20 iterations, 1 s each
# Measurement: 20 iterations, 1 s each
# Timeout: 10 min per iteration
# Threads: 1 thread, will synchronize iterations
# Benchmark mode: Throughput, ops/time
# Benchmark: fr.polytechtours.javaperformance.tp2.MyBenchmark.testMethod1

# Run progress: 0,00% complete, ETA 00:13:20
# Fork: 1 of 10
# Warmup Iteration 1: 2972336088,658 ops/s
# Warmup Iteration 2: 2954626843,741 ops/s
# Warmup Iteration 3: 2909000215,059 ops/s
# Warmup Iteration 4: 2932221713,638 ops/s
# Warmup Iteration 5: 2947948679,564 ops/s
# Warmup Iteration 6: 2924302309,971 ops/s
# Warmup Iteration 7: 3090414818,514 ops/s
# Warmup Iteration 8: 3062594057,480 ops/s
# Warmup Iteration 9: 3077267415,953 ops/s
# Warmup Iteration 10: 3116499866,813 ops/s
# Warmup Iteration 11: 3090378414,579 ops/s
# Warmup Iteration 12: 3098975690,169 ops/s
# Warmup Iteration 13: 3118421547,852 ops/s
# Warmup Iteration 14: 3112105090,081 ops/s
# Warmup Iteration 15: 3072188081,370 ops/s
# Warmup Iteration 16: 2752941820,475 ops/s
# Warmup Iteration 17: 2800058412,538 ops/s
# Warmup Iteration 18: 2933602883,916 ops/s
# Warmup Iteration 19: 2999105937,817 ops/s
# Warmup Iteration 20: 2990685821,859 ops/s
Iteration  1: 2969003868,204 ops/s
Iteration  2: 3004260144,740 ops/s
Iteration  3: 2958892686,325 ops/s
Iteration  4: 3001776867,386 ops/s
Iteration  5: 2963945132,613 ops/s
Iteration  6: 2937427034,203 ops/s
Iteration  7: 2713769314,310 ops/s
Iteration  8: 2834707333,226 ops/s
```

```
Iteration  9: 2941239699,458 ops/s
Iteration 10: 2822432305,047 ops/s
Iteration 11: 2462780679,477 ops/s
Iteration 12: 2736274431,706 ops/s
Iteration 13: 2767682893,439 ops/s
Iteration 14: 2937272013,441 ops/s
Iteration 15: 2908576330,944 ops/s
Iteration 16: 2942766364,200 ops/s
Iteration 17: 2927723718,878 ops/s
Iteration 18: 2952350104,080 ops/s
Iteration 19: 2933824491,374 ops/s
Iteration 20: 2866154041,222 ops/s
```

A la fin du benchmark, vous obtiendrez un rapport ressemblant au suivant :

```
# Run complete. Total time: 00:13:30

Benchmark              Mode  Cnt         Score        Error  Units
MyBenchmark.testMethod1 thrpt  200 2918018342,185 28895972,261 ops/s
MyBenchmark.testMethod2 thrpt  200 2919706495,784 29542881,636 ops/s
```

Selon les paramètres, le benchmark peut-être très long. Ici, il dure de 10 à 15 minutes. En effet, il effectue 20 phases de warmup d'une seconde puis lance 20 itérations de tests, le tout 10 fois par test. Ici, on lui demande de nous afficher le nombre d'opérations qu'il a réussi à exécuter par seconde. Ce sont les paramètres par défaut de JMH.

Note : Il existe des plugins pour certains IDE permettant de lancer un benchmark directement via son interface graphique. Vous pouvez l'utiliser si vous le souhaitez, cependant il est recommandé de savoir se servir de JMH en ligne de commande.

1.2 Etude de JMH

Avant de vous lancer dans l'écriture d'un micro-benchmark, il vous est demandé de lire la documentation associé à JMH. Vous constaterez que JMH est très simple d'utilisation et qu'il repose en grande partie sur l'utilisation d'annotations. Parmi les plus importantes, il vous est demandé de décrire les suivantes : Benchmark, BenchmarkMode, Fork, Group, GroupThread, Measurement, OutputTimeUnit, State, State, TearDown, Timeout, Warmup, CompilerControl. Il existe d'autres annotations, vous avez la possibilité de les décrire si vous le souhaitez. En plus de cela, des classes utilitaires sont disponibles telles que BlackHole part exemple. Il peut être utile de regarder les options disponibles au niveau de la JVM, par exemple CompileCommand.

En complément, il est demandé de décrire les paramètres possibles à fournir à JMH ainsi que ce que signifie le bilan final affiché par celui-ci : comment doit-on le comprendre ? Comment déterminer quelle implémentation est la plus rapide ? Etc...

1.3 Mise en place d'un micro-benchmark

Maintenant que vous savez comment générer un projet JMH, il vous est demandé de mettre en place un micro-benchmark afin de comparer deux (ou plus) méthodes qui réalisent la même chose d'un point de vue fonctionnel, mais développé de manière différente afin de confronter les performances. Il n'est pas forcément nécessaire de comparer des méthodes complexes afin de se rendre compte de l'utilité de JMH, vous pouvez rester sur des méthodes simples. Voici une liste d'idées que vous pouvez mettre en place :

- StringBuffer vs StringBuilder
- StringBuilder vs String.concat

- String.replace vs StringUtils.replace
- StringUtils.trim vs String.trim
- De manière générale, n'importe quelle implémentation A d'un algorithme vs une implémentation B de celui-ci. On peut par exemple penser au calcul de la suite de Fibonacci, de la factorielle... Soyez créatifs !

Notez cependant qu'il est préférable de choisir la dernière option, à savoir confronter deux implémentations d'un algorithme que vous connaissez. Si vous n'avez pas d'idée, ou pas de code sous la main, vous pouvez utiliser les différentes implémentations de tris que j'ai placé sur l'espace sur célène.

N'hésitez pas à multiplier les tests avec des paramètres différents. Par exemple, si vous travaillez avec des chaînes de caractères, faites des tests avec des chaînes très courtes, d'autres avec des tailles plus grandes. L'objectif et de pouvoir comparer deux méthodes le plus objectivement possible, en prenant en compte tous les paramètres possibles. De plus, vous pouvez créer plusieurs classes de tests afin de jouer avec les paramètres de JMH ou de la JVM en elle-même. Vous êtes libres de tester autant de possibilités que vous le souhaitez. Vous devrez présenter dans votre rapport les différents résultats que vous avez obtenus en essayant de les expliquer.

1.4 Optionnel : Analyse des méthodes testées

Si vous le souhaitez, vous pouvez aussi utiliser les outils vus lors du premier TP (jvisualvm, java mission control etc) sur les méthodes que vous avez testées afin d'essayer de comprendre les raisons qui font qu'une méthode est plus efficace qu'une autre.