Java Performance - TP1

October 8, 2018

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

Chapter 1

Test des coutils

1.1 vmstat

vmstat est un outil en ligne de commande permettant d'obtenir différentes informations sur la VM Java:

- Nombre de processus:
  - Processus en cours d'exécution
  - Processus dormants qu'on ne peut interrompre
- Mémoire:
  - Mémoire virtuelle utilisée
  - Mémoire libre
  - Mémoire buffer
  - Mémoire cache
  - Mémoire inactive
  - Mémoire active
- Swap
  - Entrée
  - Sortie
- IO
  - Lecture
  - Ecriture
- Système
  - Nombre d'interruptions par seconde
  - Nombre de changement de contexte
- CPU
  - Temps en non-kernel
  - Temps en kernel
  - Temps en attente
  - Temps en attente d'IO
  - Temps pour la VM

```
administrateur@vm-ubuntu-javaperf:~$ vmstat 2
procs -----------mémoire---------- -échange- -----io---- -système- ------cpu-----
 r  b   swpd   libre tampon  cache   si   so    bi    bo   in   cs us sy id wa st
 0  0      0 466424  64000 1341028    0    0   184    27  344  563  5  4 89  2  0
 0  0      0 466416  64000 1341028    0    0     0     0  927 1728  0  0 100  0  0
 0  0      0 466416  64000 1341028    0    0     0     0  862 1618  0  0 99  0  0
 0  0      0 466416  64000 1341028    0    0     0     0  909 1680  0  1 99  0  0
 0  0      0 398880  64124 1343864    0    0  1486     0 2537 4680 14  6 75  4  0
 3  0      0 378160  64628 1346776    0    0  1362   274 2860 4909  7  9 77  7  0
 1  1      0 334016  64900 1365404    0    0  9430     0 2572 4149 15  8 61 16  0
 0  1      0 268792  65000 1370800    0    0  2746     0 3424 4380 47 12 34  8  0
 0  1      0 235560  65188 1378292    0    0  3834   194 2664 4028 20 10 63  7  0
 1  0      0 186456  65344 1385292    0    0  3580     0 2825 4593 35 12 47  6  0
 1  1      0 130244  66128 1389732    0    0  2418    78 3623 7562 46  9 38  7  0
 0  0      0 114388  66164 1392504    0    0  1360   256 3056 4889 21  6 70  4  0
 0  0      0 115860  66164 1392504    0    0     0     0 1054 1900  0  1 99  0  0
 0  0      0 121564  66172 1392508    0    0     0    64 1011 1904  0  1 99  0  0
 0  0      0 121564  66172 1392508    0    0     0     0  911 1696  0  0 99  0  0
 0  0      0 134460  66172 1392508    0    0     0     0  929 1783  0  1 99  0  0
```

Figure 1.1 – Sortie vmstat au lancement d'Eclipse

On peut remarquer que:

- La mémoire libre diminue tandis que la mémoire cache augmente
- La lecture des fichiers explose
- Une légère augmentation de l'écriture de fichiers de temps en temps
- Le temps CPU non kernel augmente tendis que le temps CPU en attente diminue.

Vmstat nous permet donc d'obtenir des informations globales sur l'état de la VM. Cependant cela ne permet pas une analyse poussée des besoins en resources d'un programme car aucune différenciation n'est faire.

1.2 iostat

Iostat permet d'obtenir des information générales sur les IOs (par périphérique) et CPU de tout le système. Cela peut être répété *n* fois avec un interval donné.

```
administrateur@vm-ubuntu-javaperf:~/java-performance$ iostat 2 10
Linux 4.15.0-34-generic (vm-ubuntu-javaperf)    25/09/2018      _x86_64_        (4 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           2,69    0,02    2,35    1,26    0,00   93,68

Device:            tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sda              18,12       406,22        62,41    1362891     209392
```

Figure 1.2 – iostat avec Eclipse ouvert

Cet outil est assez peu précis concernant Java étant donné que les statistiques obtenues concernent tout le système. Des interférences d'autres programmes peuvent altérer notre analyse.

1.3 nicstat

Cet outil est similaire à iostat à la différence qu'il analyse les données transitant par les cartes réseau.

```
    Time      Int   rKB/s   wKB/s   rPk/s   wPk/s    rAvs    wAvs %Util     Sat
11:32:19       lo    0.00    0.00    0.00    0.00    0.00    0.00  0.00    0.00
11:32:19    ens33    0.00    0.00    0.00    0.00    0.00    0.00  0.00    0.00
    Time      Int   rKB/s   wKB/s   rPk/s   wPk/s    rAvs    wAvs %Util     Sat
11:32:21       lo    0.17    0.17    2.00    2.00   87.75   87.75  0.00    0.00
11:32:21    ens33    0.45    0.31    3.00    3.50   153.2   89.86  0.06    0.00
    Time      Int   rKB/s   wKB/s   rPk/s   wPk/s    rAvs    wAvs %Util     Sat
11:32:23       lo    0.00    0.00    0.00    0.00    0.00    0.00  0.00    0.00
11:32:23    ens33    1.80    0.99    6.50    6.00   283.4   168.9  0.23    0.00
    Time      Int   rKB/s   wKB/s   rPk/s   wPk/s    rAvs    wAvs %Util     Sat
11:32:25       lo    0.00    0.00    0.00    0.00    0.00    0.00  0.00    0.00
11:32:25    ens33   165.1    2.53   115.4   45.97  1464.9   56.34  13.7    0.00
```

Figure 1.3 – Evolution du réseau lors du rafraichissement des packages disponibles dans Eclipse

1.4 JCMD

`jcmd` permet d'obtenir les différents processus Java lancés.

On peut par la suite lancer `jcmd <PID> <command>` pour effectuer des commandes sur la JVM. On retrouve notamment la commande help pour obtenir la liste des commandes disponibles.

La commande `Thread.print` affiche tous les threads lancés ainsi que leur status et pile d'appel.

La commande `GC.run` permet de lancer le GC manuellement.

De manière générale JCMD se connecte à la VM pour exécuter des commandes pour obtenir des informations sur la VM mais aussi pour exécuter des actions sur celle-ci.

1.5 JConsole

On retrouve des informations que nous avons déjà pu voir auparavant (Mémoire, Threads, etc.) mais sous forme graphique.

Cela permet de voir simplement et rapidement les évolutions au cours du temps.

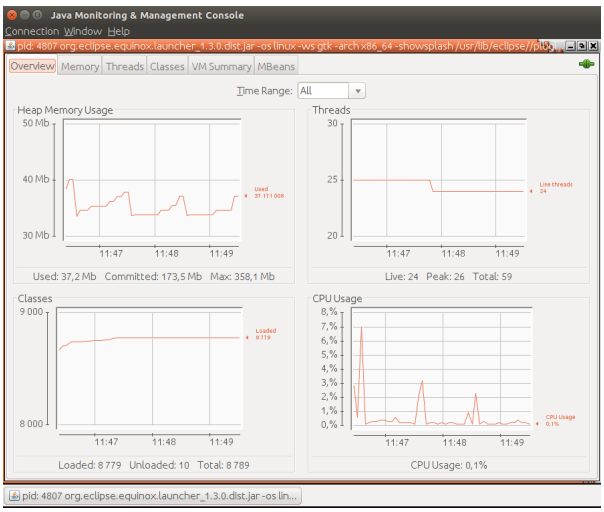


Figure 1.4 – Eclipse en fond (vue générale)

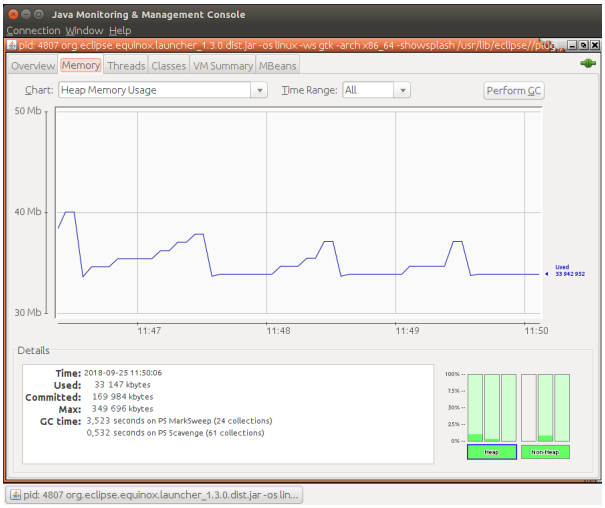


Figure 1.5 – Eclipse en fond (Mémoire heap)

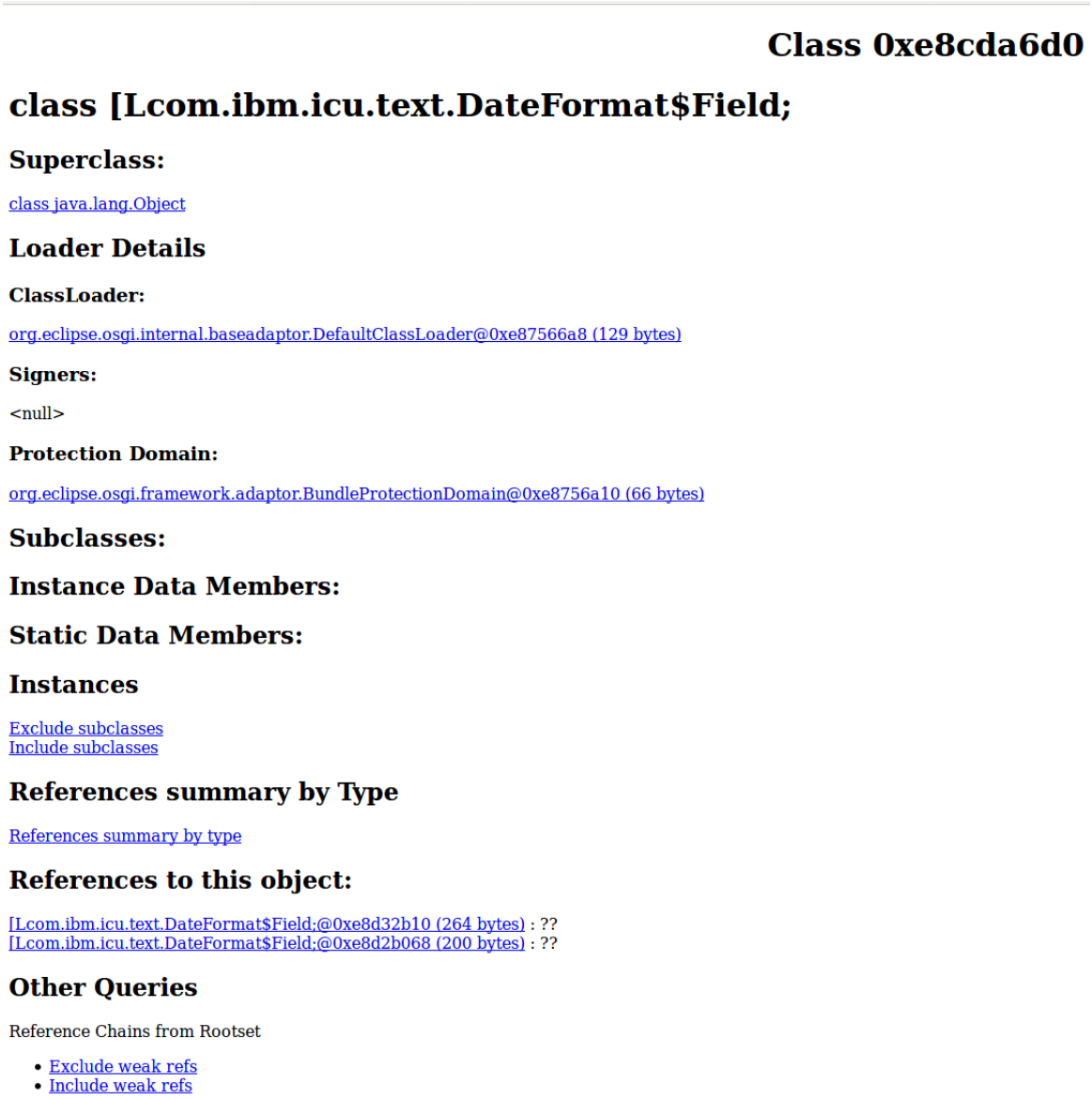
Cependant nous devons nous connecter à une application déjà lancée. Il est donc difficile de récupérer les statistiques au lancement d'une application.

1.6 JHat

JHat permet d'analyser les fichier hprof de la JVM.

Nous pouvons en générer un à la volée grâce à `jmap -dump:file=<file> <pid>`. Ensuite nous l'ouvrons avec `jhat <file>`. Dès que le fichier est analysé, un serveur HTTP est crée. Nous retrouvons sur ce dernier les informations du fichier sous forme visuelle.

Figure 1.6 – Toutes les classes



Class 0xe8cda6d0

class [Lcom.ibm.icu.text.DateFormat$Field;

Superclass:

class java.lang.Object

Loader Details

ClassLoader:

org.eclipse.osgi.internal.baseadaptor.DefaultClassLoader@0xe87566a8 (129 bytes)

Signers:

<null>

Protection Domain:

org.eclipse.osgi.framework.adaptor.BundleProtectionDomain@0xe8756a10 (66 bytes)

Subclasses:

Instance Data Members:

Static Data Members:

Instances

Exclude subclasses
Include subclasses

References summary by Type

References summary by type

References to this object:

[Lcom.ibm.icu.text.DateFormat$Field;@0xe8d32b10 (264 bytes) : ??
[Lcom.ibm.icu.text.DateFormat$Field;@0xe8d2b068 (200 bytes) : ??

Other Queries

Reference Chains from Rootset

- Exclude weak refs
- Include weak refs

Figure 1.7 – Détails d'une classe

1.7 JStack

Affiche la liste des threads de la VM, similaire à `Thread.print` de jcmd.

1.8 JVisualVM

Outil similaire à JConsole affichant des graphes des différentes métriques de la VM (mais en plus beau).

1.9 Java Mission Control

Encore un interface similaire aux autres. Cependant nous pouvons ajouter des plugins afin d'obtenir des interfaces supplémentaires.

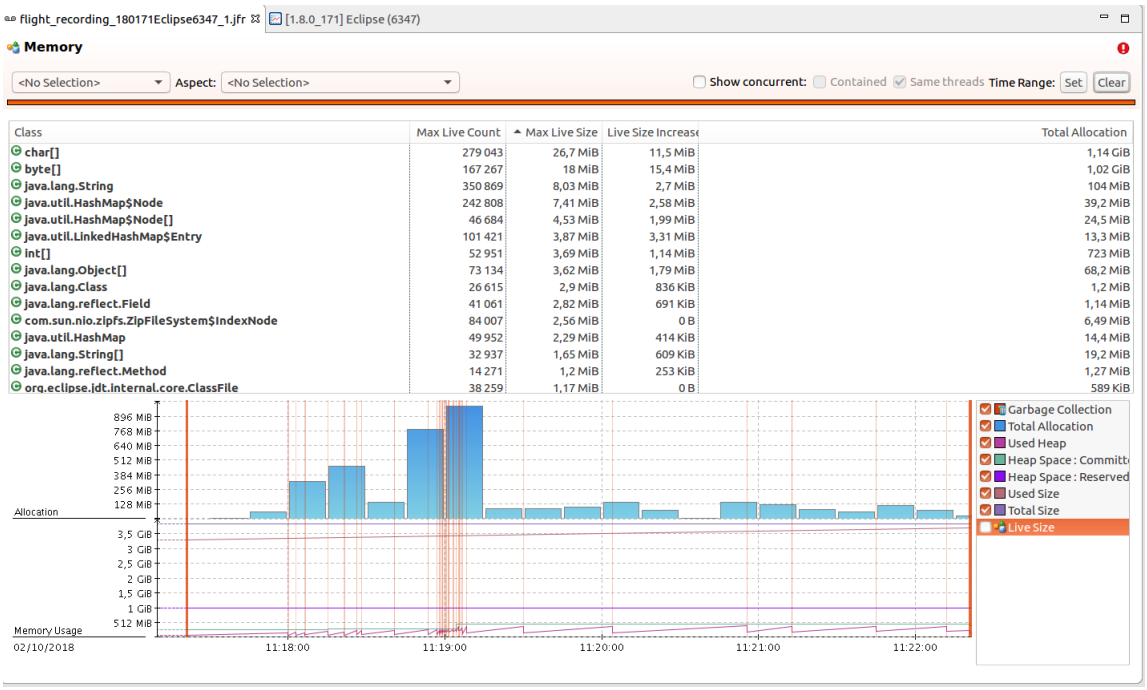


Figure 1.8 – Plugin GC de Java Mission Control