TP5: Etude et amélioration d'une application

—— GUO Xiaoqing

git:
https://github.com/GuoJulie/S9_Java-Performance/tree/main/TP5%20Etude%20et%20amelioration%20d'une%20application

1. Pre-analyse du programme et faiblesses

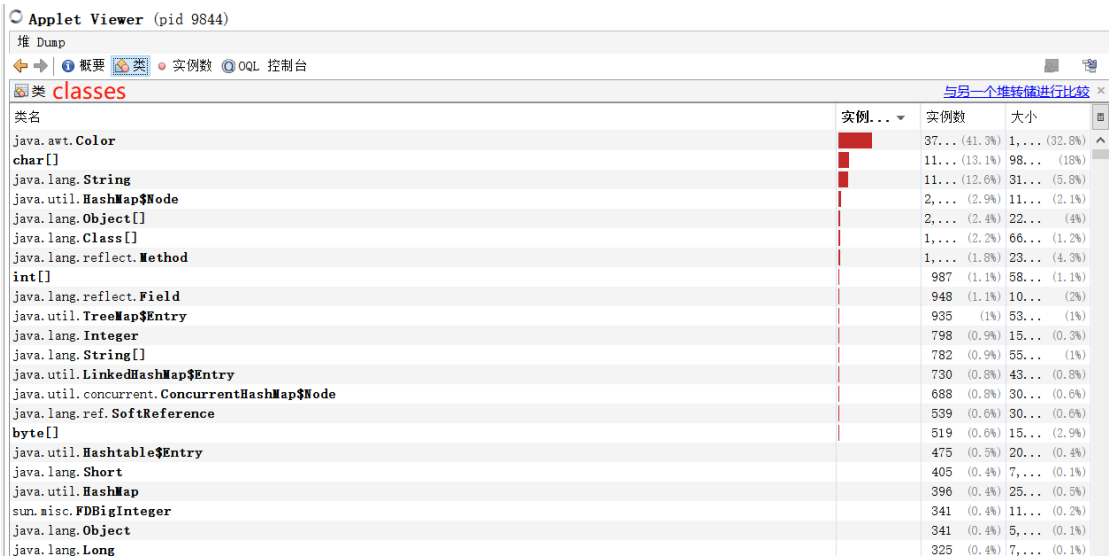
Build et Run :

```
[INFO] Scanning for projects...
[WARNING] The project org.polytechtours.javaperformance.tp:paintingants:jar:1.0.0-SNAPSHOT uses
[INFO]
[INFO] ---------< org.polytechtours.javaperformance.tp:paintingants >----------
[INFO] Building paintingants 1.0.0-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ paintingants ---
[INFO] Deleting D:\郭晓庆文档\计算机科学与技术\课件-MUNDUS\第三学年\2020-2021\09 Java Performance
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  0.487 s
[INFO] Finished at: 2021-01-31T21:21:30+01:00
[INFO] ------------------------------------------------------------------------
```

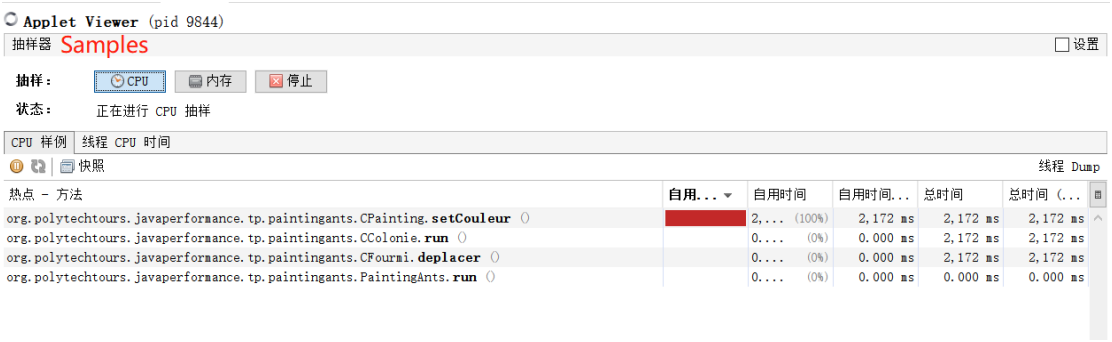
```
Seuil de luminance:40.0
Random:(42,213,194)(56,109,162)(0.16048697,0.1698532,6,3)(o,0.20812088,0.7781506,0.013728499,0.6390579);
Random:(243,163,25)(183,183,133)(0.10477132,0.90565026,1,1)(o,0.16191705,0.8259861,0.012096882,0.6600679);
Random:(183,183,133)(215,89,51)(0.16252291,0.05671762,4,3)(d,0.37761146,0.5197198,0.10266876,0.6684595);
Random:(215,89,51)(217,78,89)(0.61726314,0.56356984,7,0)(d,0.27650252,0.5048157,0.21868181,0.67689645);
Random:(56,109,162)(215,89,51)(0.03442757,0.9658876,7,0)(d,0.6399551,0.19708358,0.1629613,0.7637966);
Random:(217,78,89)(42,213,194)(0.98971605,0.3354897,7,1)(d,0.031438664,0.93041337,0.038147986,0.85736036);
```

Test :
J'ai utilisé l'outil d'augmentation des performances Java VisualVM :
VisualVM nous aide à analyser l'utilisation de la mémoire en détectant les informations des classes et objets chargés dans la JVM.Nous pouvons analyser la mémoire de l'application à travers les balises de surveillance de VisualVM.

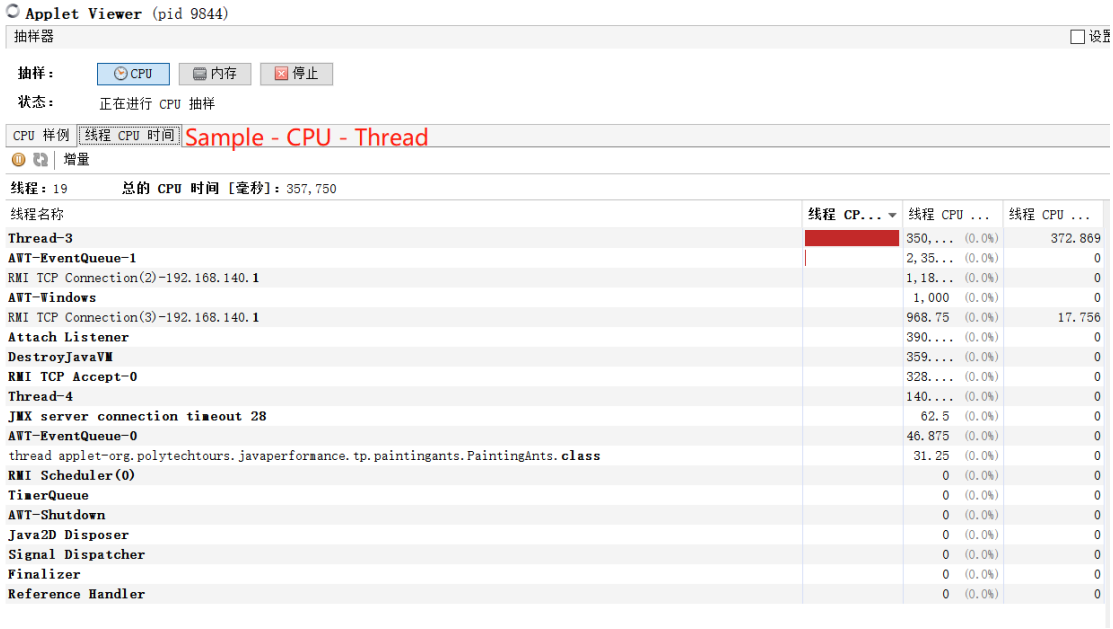
Lorsque le programme est en cours d'exécution, vérifiez l'onglet VisualVM Monitor :
La première image : Le CPU oscille autour de 23%.
La deuxième image : la mémoire du tas devient plus grande et oscille.
La troisième image : "classe" pas de changement de courbe
La quatrième image : Le nombre de threads est de 19.



Avant la fin du programme, cliquez sur le bouton Heap Dump, attendez un moment, obtenez le résultat de Dump, on clique sur Classes et constatez que la mémoire occupée par Color classe est la plus grande, et suivi du plus grand char[].



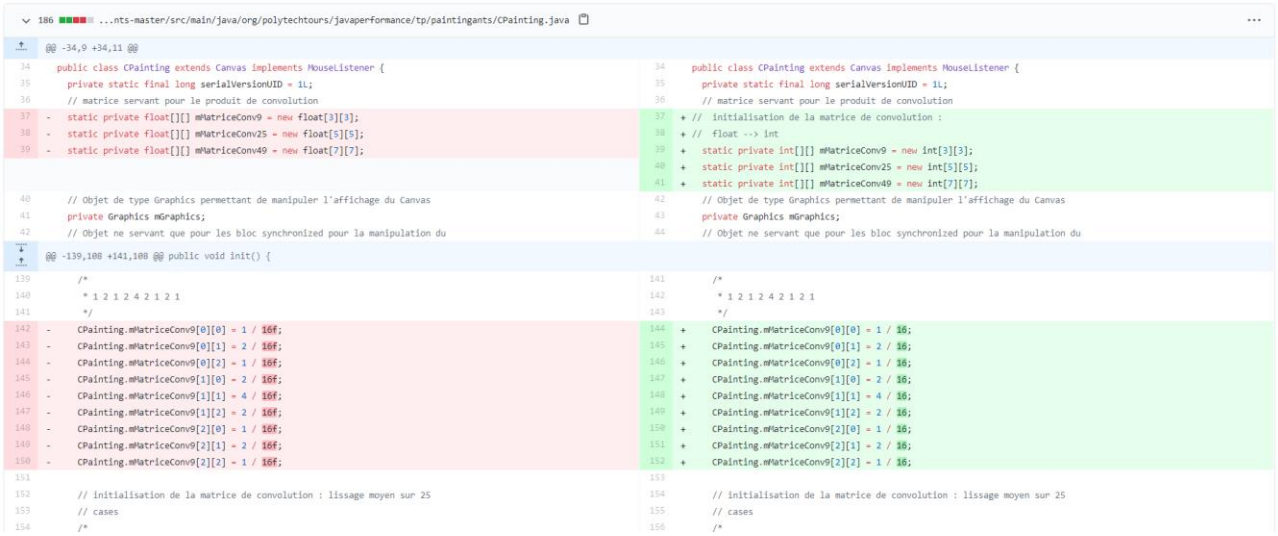
Cliquez sur Sampler, cliquez sur le bouton «CPU» pour démarrer la session d'analyse des performances du CPU, VisualVM détectera toutes les méthodes appelées de l'application, et on peut voir que la méthode setCouleur() a le plus long temps d'auto-utilisation sous CPU samples.



Passez à la page Thread CPU Time, nous pouvons voir que le processus de Thread-3 prend le plus de temps CPU.

2. Amélioration des performances et argumentation

1) Utilisez le type int chaque fois que possible → éviter le Boxing et Unboxing



2) Supprimer le mot-clé “synchronized ” → Si le mot clé "synchronized" est utilisé pour la méthode deplacer (), un thread en attente attendra indéfiniment s'il ne peut pas acquérir le verrou, ce qui conduit à une faible efficacité du programme.

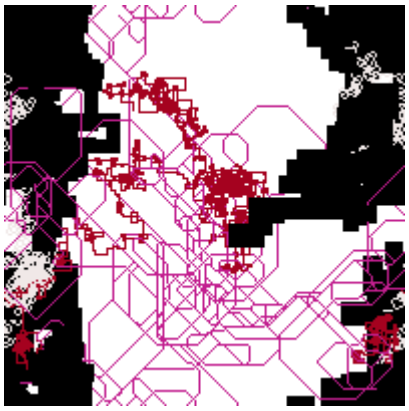
3. Post-analyse du programme

```
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ paintingants ---
[INFO] Deleting D:\郭晓庆文档\计算机科学与技术\课件-MUNDUS\第三学年\2020-2021\09 Java Performance\02 TPs\S9_
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  0.408 s
[INFO] Finished at: 2021-01-31T23:39:56+01:00
[INFO] ------------------------------------------------------------------------
```

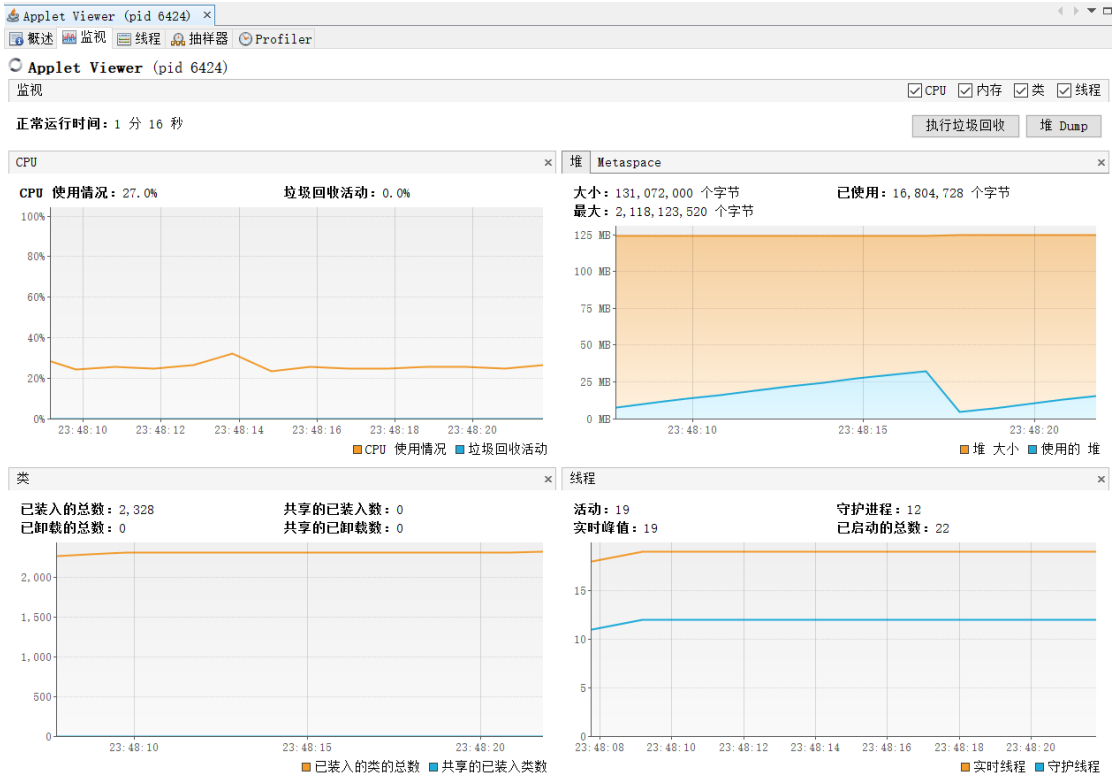
Il n'y a pas de aucun changement significatif (juste un peu moins long) dans le temps de compilation de programme.

```
Seuil de luminance:40.0
Random:(34,86,42)(8,251,251)(0.22364923,0.5701427,7,2)(o,0.39328784,0.55254835,0.054163814,0.80879474
Random:(62,176,33)(8,251,251)(0.1715936,0.48462674,3,2)(o,0.11513375,0.09609422,0.78877205,0.53178245
Random:(8,251,251)(62,176,33)(0.19175771,0.2594323,1,1)(d,0.017789986,0.025233574,0.9569764,0.8300032
Random:(102,207,131)(62,176,33)(0.58798486,0.890051,0,1)(d,0.42161727,0.5564836,0.021899104,0.9947719
```

« Random » aléatoire requise un peu.



Le programme fonctionne plus vite.



La plage de fluctuation est réduite et les performances de la mémoire sont légèrement optimisées.

概述 监视 线程 抽样器 Profiler [heapdump] 23:48:41 ×

Applet Viewer (pid 6424)

堆 Dump

概要 类 实例数 OQL 控制台

类

与另一个堆转储进行比较

类名	实例...	实例数	大小
java.awt.Color		30... (37.1%)	1,... (29%)
char[]		11... (13.9%)	97... (18.9%)
java.lang.String		11... (13.4%)	31... (6.1%)
java.util.HashMap$Node		2,... (3.2%)	11... (2.3%)
java.lang.Object[]		2,... (2.6%)	22... (4.3%)
java.lang.Class[]		1,... (2.3%)	66... (1.3%)
java.lang.reflect.Method		1,... (1.9%)	23... (4.5%)
int[]		1,... (1.2%)	59... (1.2%)
java.lang.reflect.Field		948 (1.1%)	10... (2.1%)
java.util.TreeMap$Entry		939 (1.1%)	53... (1%)
java.lang.Integer		798 (1%)	15... (0.3%)
java.lang.String[]		782 (0.9%)	55... (1.1%)
java.util.LinkedHashMap$Entry		730 (0.9%)	43... (0.9%)
java.util.concurrent.ConcurrentHashMap$Node		642 (0.8%)	28... (0.6%)
java.lang.ref.SoftReference		542 (0.6%)	30... (0.6%)
byte[]		521 (0.6%)	15... (3.1%)
java.util.Hashtable$Entry		475 (0.6%)	20... (0.4%)
java.lang.Short		405 (0.5%)	7,... (0.1%)
java.util.HashMap		395 (0.5%)	25... (0.5%)
java.lang.Object		342 (0.4%)	5,... (0.1%)
sun.misc.FDBigInteger		341 (0.4%)	11... (0.2%)
java.lang.Long		325 (0.4%)	7,... (0.2%)
java.lang.reflect.Constructor		310 (0.4%)	37... (0.7%)
java.util.HashMap$Node[]		290 (0.3%)	72... (1.4%)
java.lang.Class$ReflectionData		287 (0.3%)	26... (0.5%)
java.lang.Byte		256 (0.3%)	4,... (0.1%)
java.lang.invoke.MemberName		235 (0.3%)	12... (0.2%)
java.lang.invoke.LambdaForm$Name		231 (0.3%)	11... (0.2%)
java.lang.ref.WeakReference		228 (0.3%)	10... (0.2%)

La classe de Color et le char[] prennent un peu moins de mémoire.

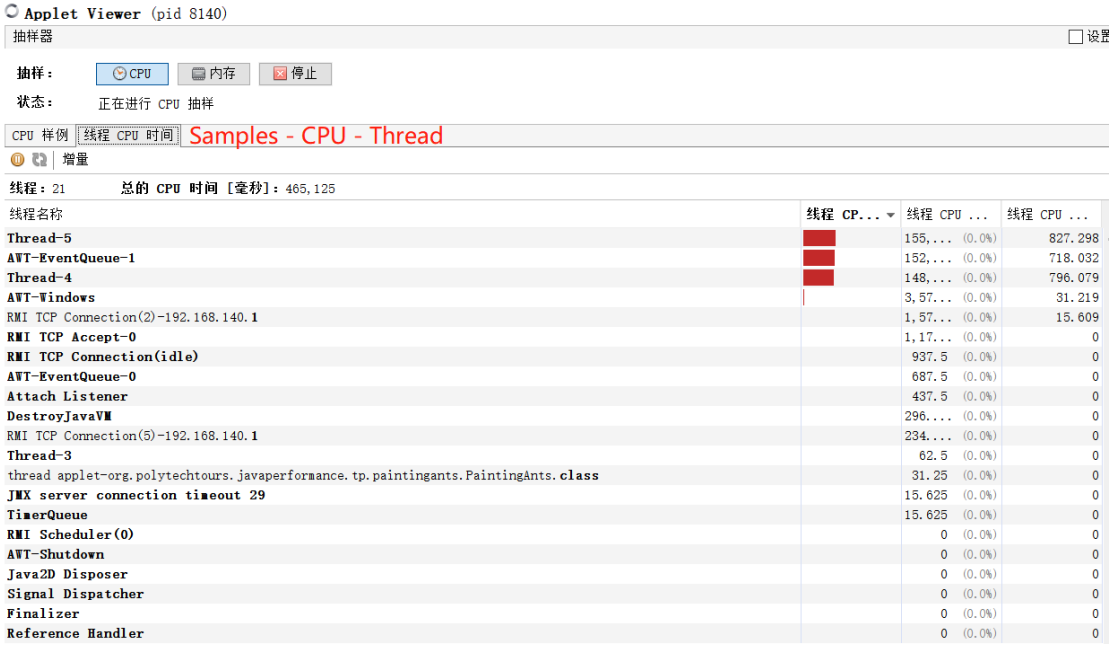
CPU 样例 线程 CPU 时间

快照

线程 Dump

热点 - 方法	自用...	自用时间	自用时间...	总时间	总时间 (...
org.polytechtours.javaperformance.tp.paintingants.CPainting.setCouleur ()		57... (52%)	57,498 ms	57,498 ms	57,498 ms
org.polytechtours.javaperformance.tp.paintingants.PaintingAnts.run ()		53... (48%)	6.23 ms	53,100 ms	6.23 ms
org.polytechtours.javaperformance.tp.paintingants.CColonie.run ()		0.... (0%)	0.000 ms	57,498 ms	57,498 ms
org.polytechtours.javaperformance.tp.paintingants.CFourmi.deplacer ()		0.... (0%)	0.000 ms	57,498 ms	57,498 ms

la méthode setCouleur() a pas le plus long temps d'auto-utilisation sous CPU samples. C'est char[] maintenant qui a e plus long temps d'auto-utilisation.



Le nombre de threads n'a pas changé de manière significative, il fluctue autour de 19.