**TP5: Etude et amélioration d'une application**

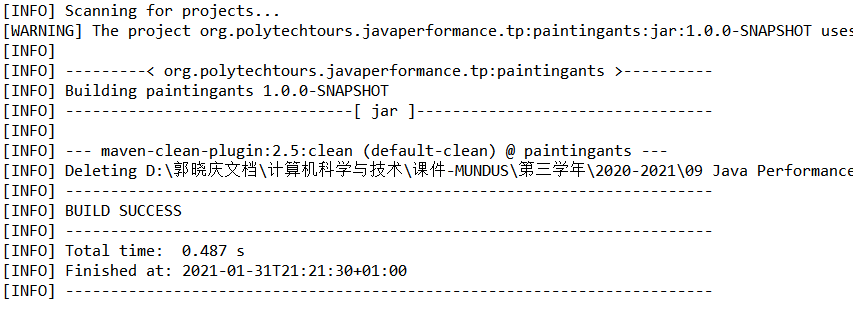
**—— GUO Xiaoqing**

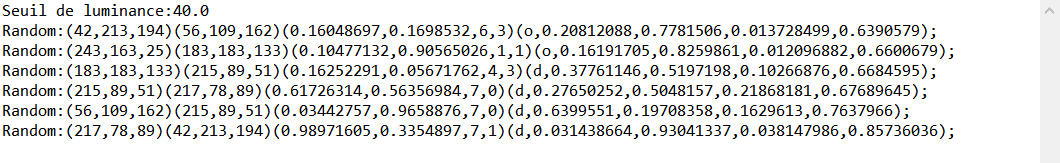
git:

<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 :







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.



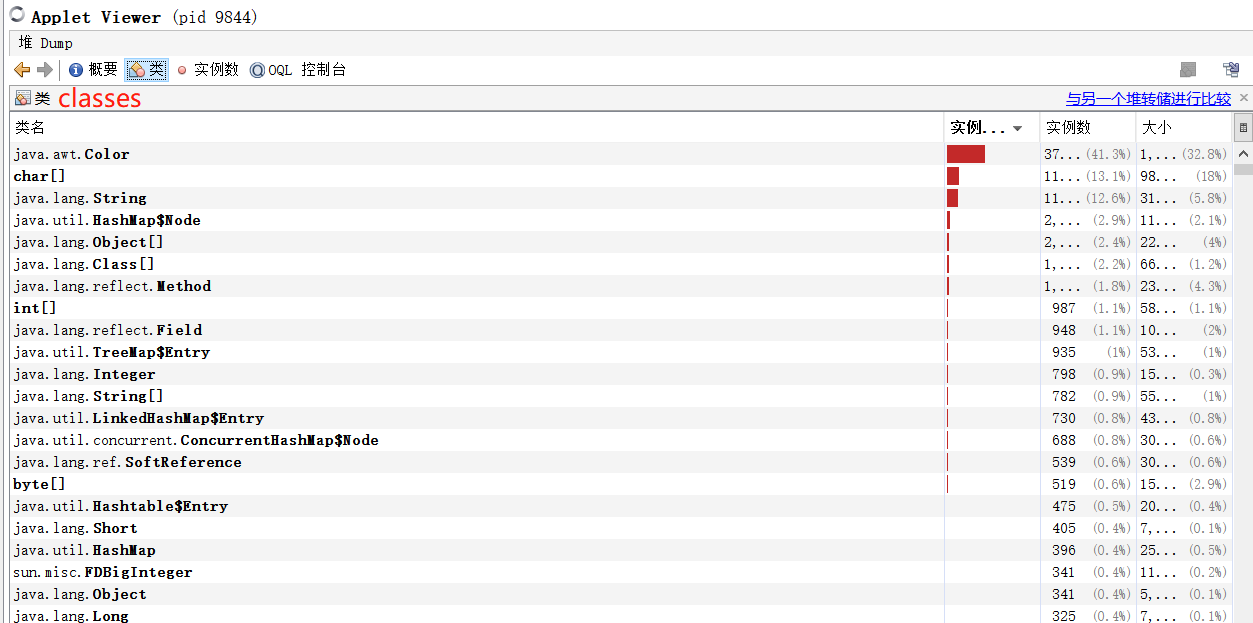
Lorsque le programme est en cours d'exécution, vérifiez l'onglet VisualVM Monitor :

La première image : Le CPU oscille autour de 23%.

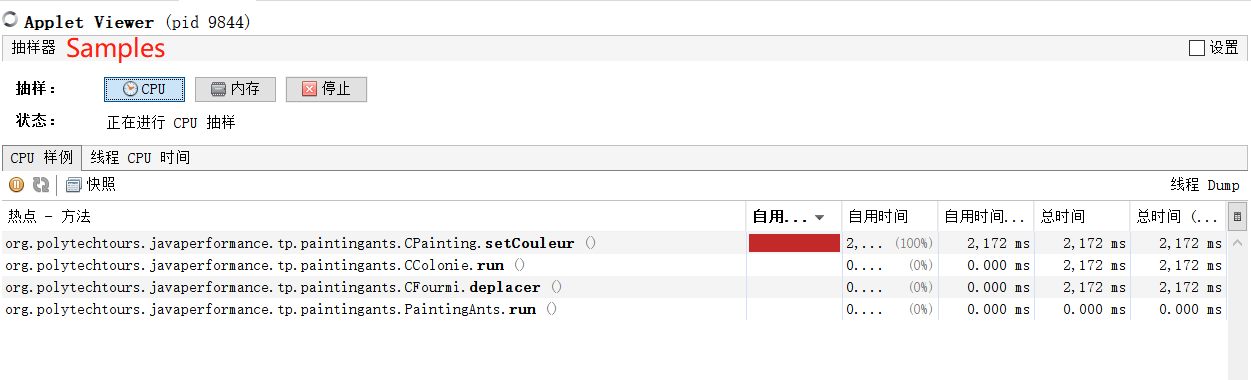
La deuxième image : la mémoire du tas devient plus grande et oscille.

La troisième image : "classe" pas de changement de courbe

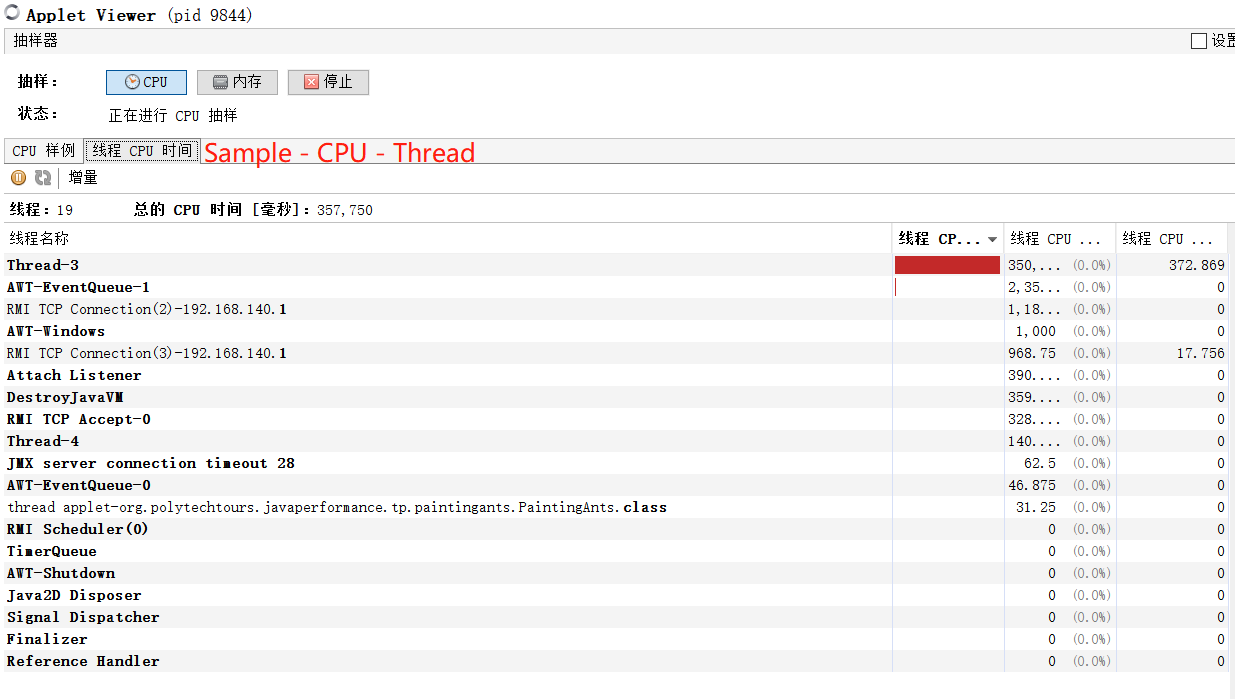
La quatrième image : Le nombre de threads est de 19.



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[].



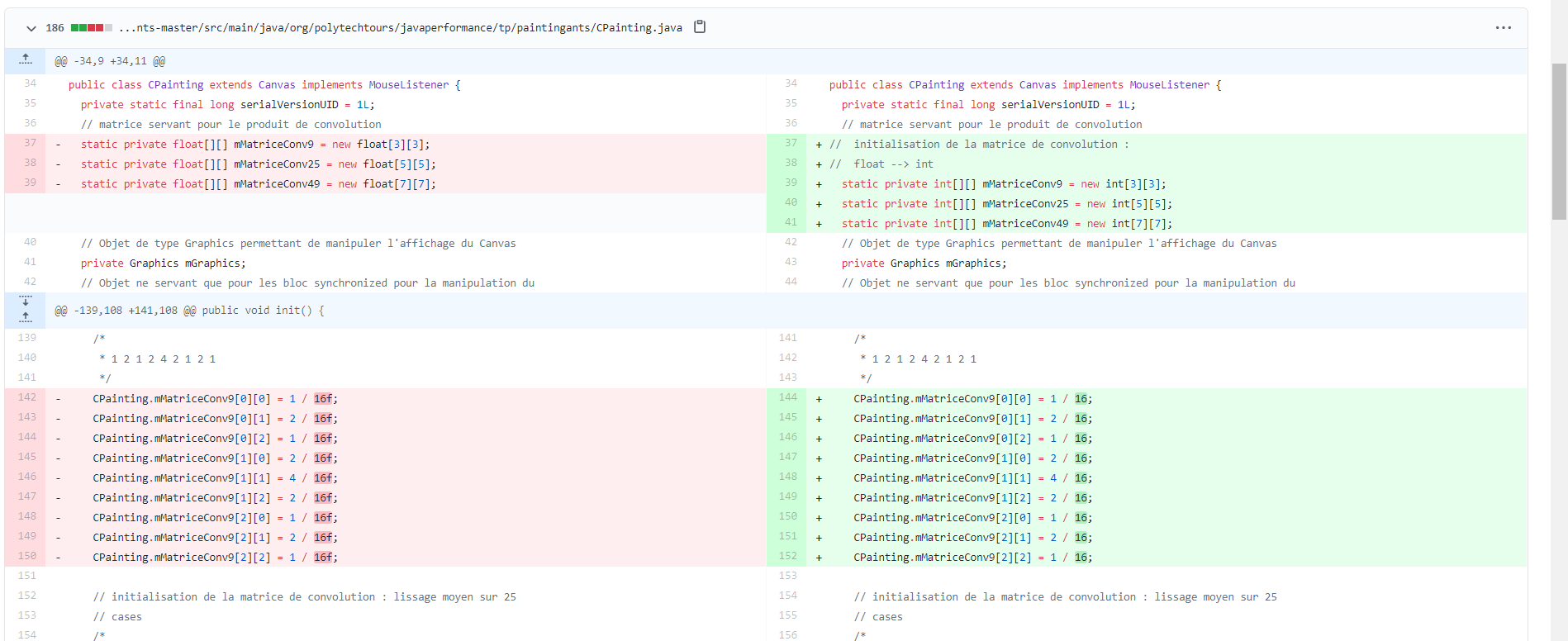
Cliquez sur Sampler, cliquez sur le bouton «CPU» pour démarrer la session d'analyse des performances du CPU, VisualVM détectera toutes les méthodes appelées de l'application, et on peut voir que la méthode setCouleur() a le plus long temps d'auto-utilisation sous CPU samples.



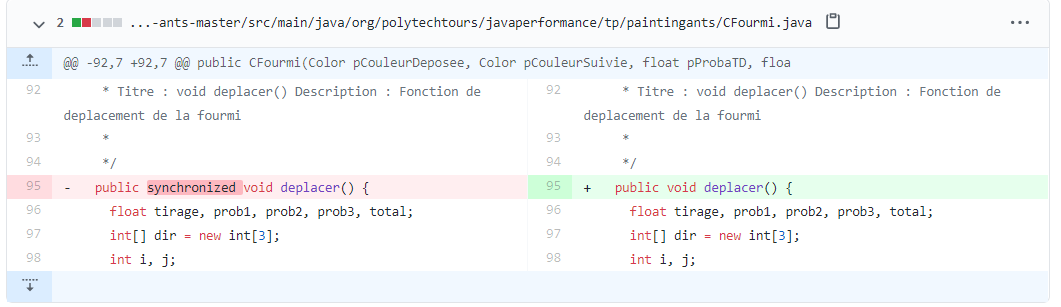
Passez à la page Thread CPU Time, nous pouvons voir que le processus de Thread-3 prend le plus de temps CPU.

# 2. Amélioration des performances et argumentation

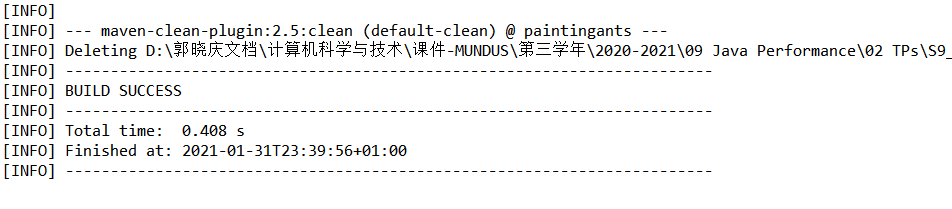
1. Utilisez le type int chaque fois que possible 🡪 éviter le Boxing et Unboxing

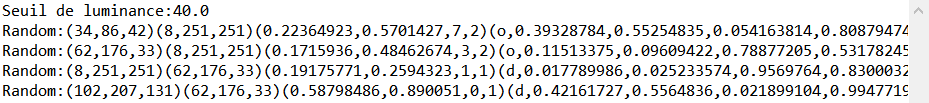


1. Supprimer le mot-clé “synchronized ” 🡪 Si le mot clé "synchronized" est utilisé pour la méthode deplacer (), un thread en attente attendra indéfiniment s'il ne peut pas acquérir le verrou, ce qui conduit à une faible efficacité du programme.

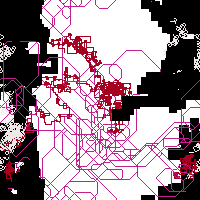


# 3. Post-analyse du programme

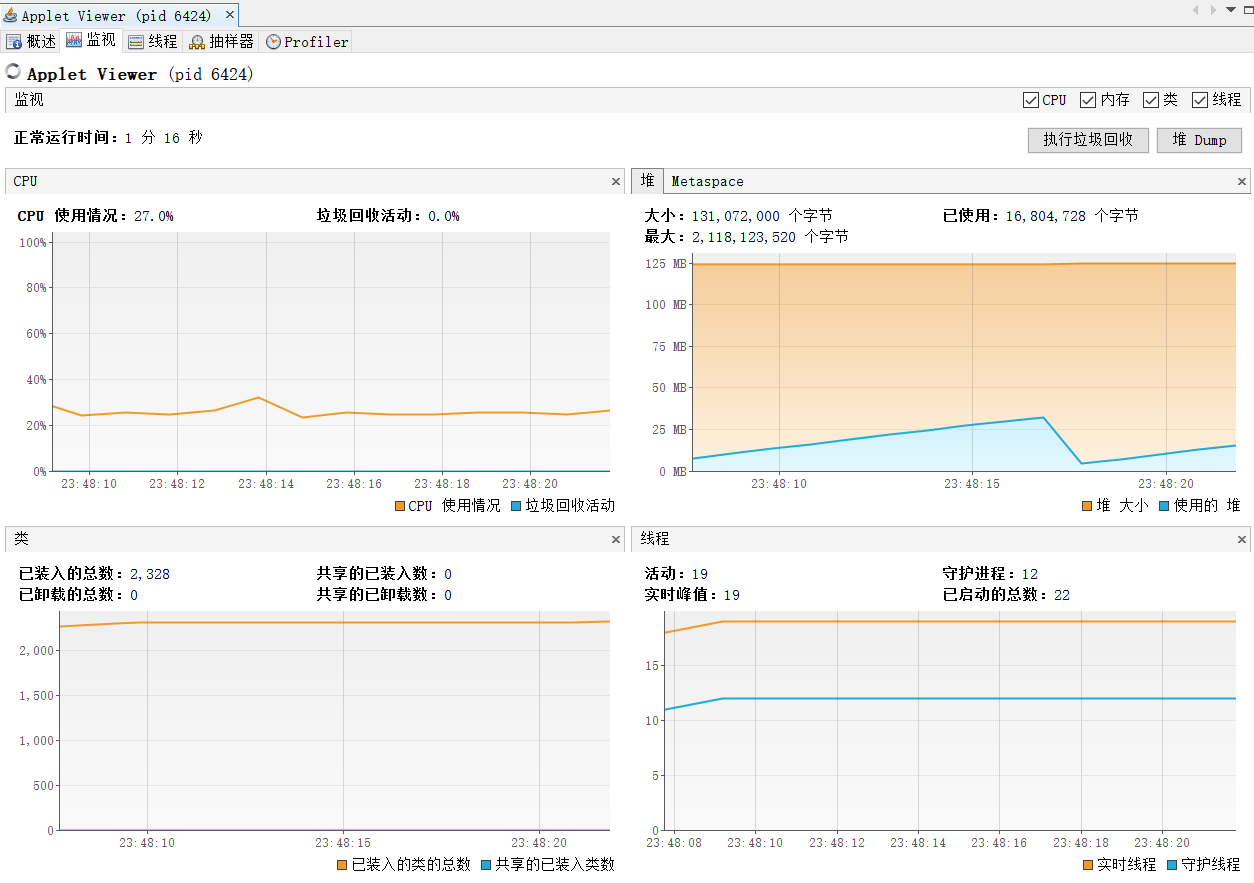
 Il n'y a pas de aucun changement significatif (juste un peu moins long) dans le temps de compilation de programme.



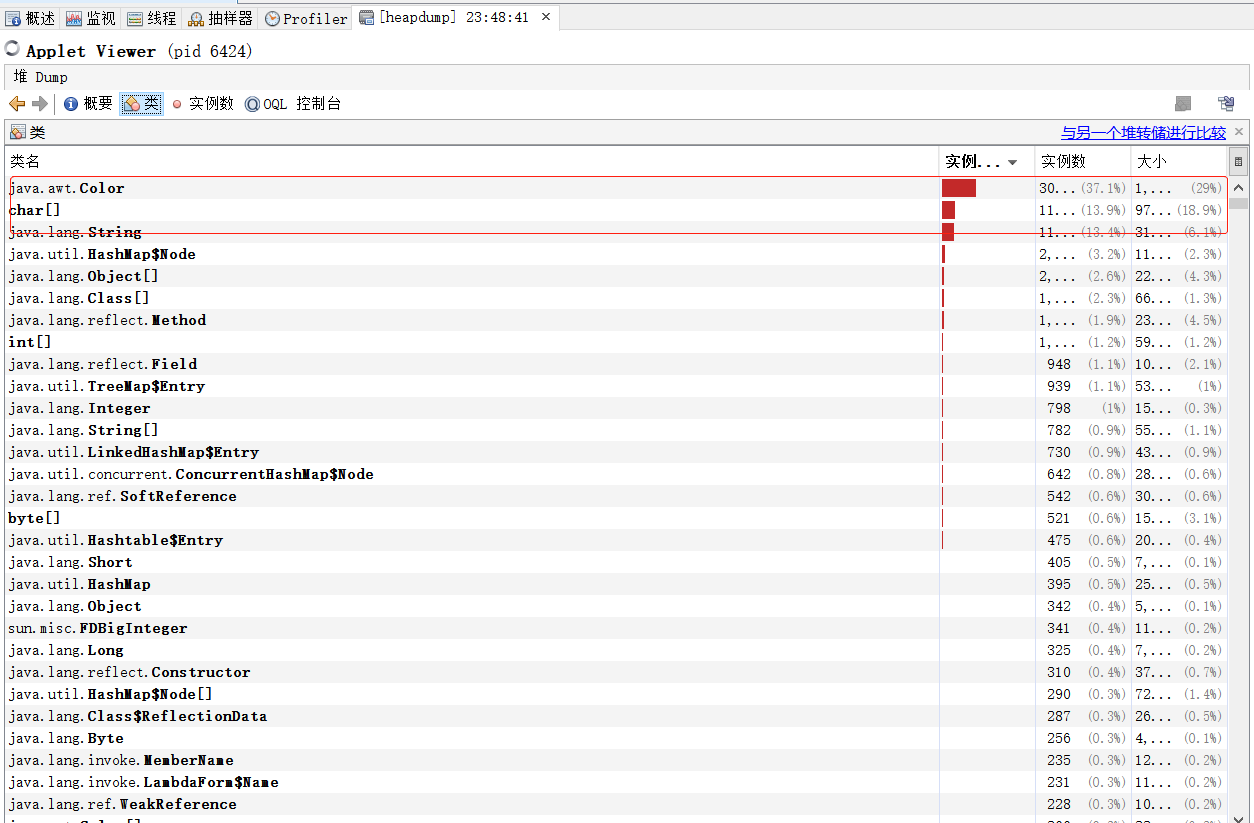
« Random » aléatoire requise un peu.



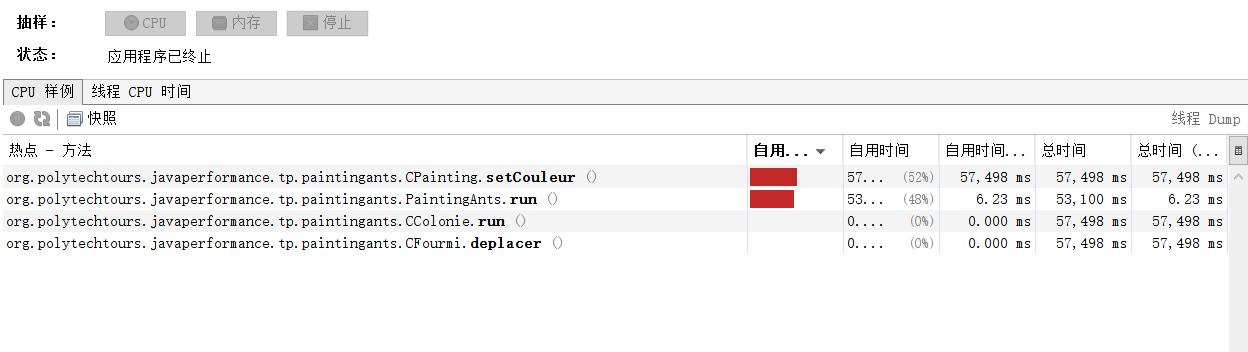
Le programme fonctionne plus vite.



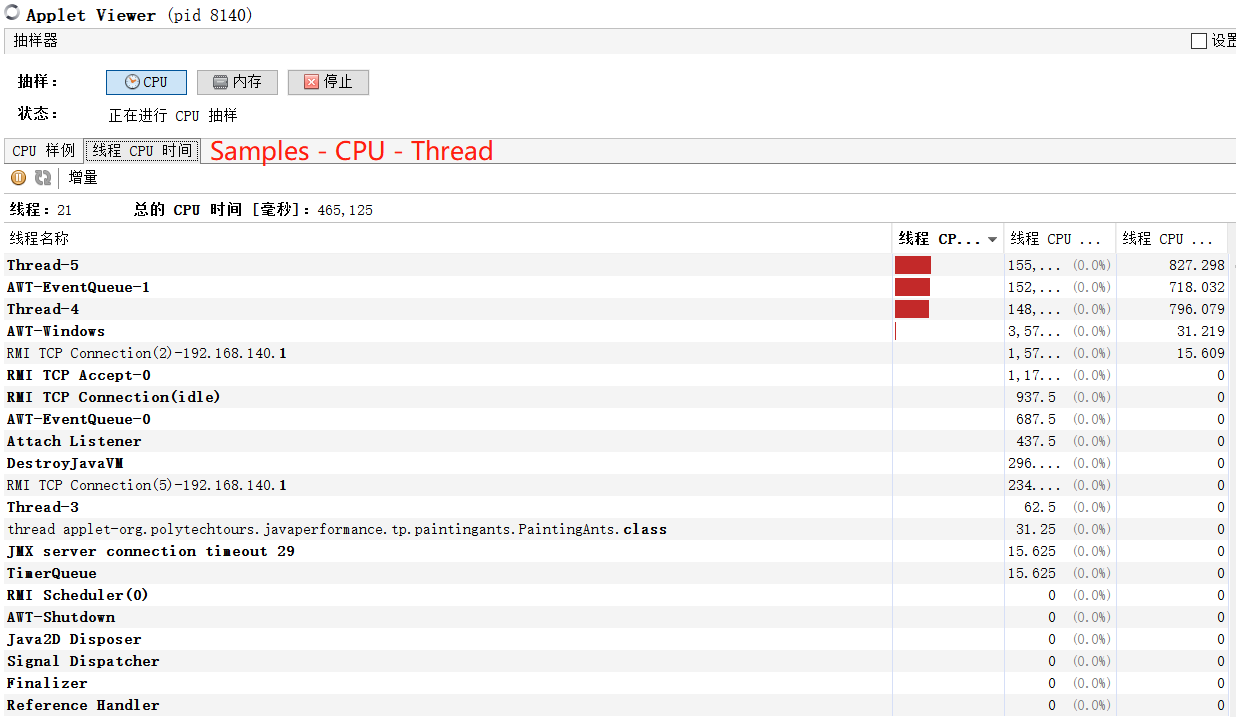
La plage de fluctuation est réduite et les performances de la mémoire sont légèrement optimisées.



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



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.