

TP5 Optimisation paintingants

PENG Hanyuan & YAN Wenli

Nous avons crée différentes branches pour des changements.

D'abord, on utilise des outils comme Jvisualvm pour visualier des performances sur ce sujet. Dans le moniteur, nous avons choisi des performance sur la mémoire en une minute pour assurer que nous comparons sous les mêmes facteurs.

On peut trouver qu'il prend beaucoup de temps (CPU) pour le calcul de convolution, et la fonction `setColor`. En même temps, la class `Color` est beaucoup appelée cependant.

0. Statut original

- FPS original

running (9782)

- CPU original

CPU 样例		线程 CPU 时间						线程 Dump	
			快照						
热点 - 方法		自用时间 [%]		自用时间	自用时间 ...	总时间	总时间 (...)		
org.polytechtours.performance.tp.fourmispeintre.PaintingAnts.run ()		<div></div>		... (50%)	107 ms	17,76...	107 ms		
org.polytechtours.performance.tp.fourmispeintre.CPainting.convolution ()		<div></div>		...(48.2%)	5,940 ...	17,12...	5,940 ...		
org.polytechtours.performance.tp.fourmispeintre.CPainting.setColor ()		<div></div>		... (1.4%)	507 ms	17,63...	6,447 ...		
org.polytechtours.performance.tp.fourmispeintre.CColonie.run ()		<div></div>		... (0.4%)	131 ms	17,76...	6,579 ...		
org.polytechtours.performance.tp.fourmispeintre.CFourmi.deplacer ()		<div></div>		... (0%)	0.000 ...	17,63...	6,447 ...		

- Memery original

增量

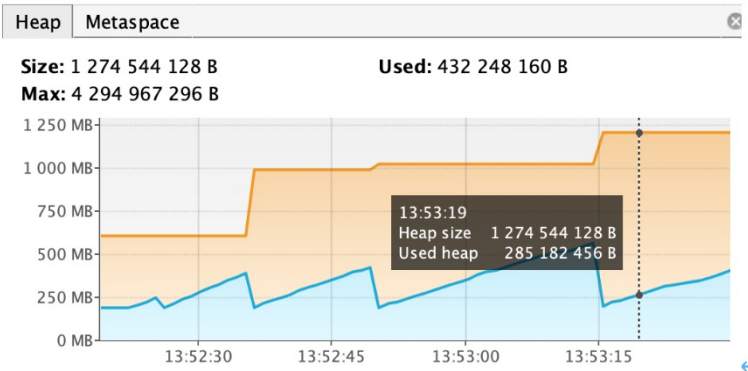
快照

执行 GC

堆 Dump

类: 1,292 实例: 5,499,947 字节: 196,302,912

类名	字节 [%] ▼	字节	实例数
java.awt.Color	<div></div>	157,522,528 (80.2%)	4,922,579 (89.5%)
java.awt.Color[]	<div></div>	16,032,000 (8.1%)	2,000 (0.0%)
java.lang.Object[]	<div></div>	4,723,280 (2.4%)	142,970 (2.5%)
char[]	<div></div>	2,719,592 (1.3%)	47,399 (0.8%)
java.util.TreeMap\$Entry	<div></div>	2,158,560 (1.0%)	53,964 (0.9%)
java.io.ObjectStreamClass\$WeakClassKey	<div></div>	1,866,432 (0.9%)	58,326 (1.0%)
int[]	<div></div>	1,769,088 (0.9%)	5,570 (0.1%)
java.util.HashMap\$Node	<div></div>	860,768 (0.4%)	26,899 (0.4%)
java.util.HashMap	<div></div>	634,080 (0.3%)	13,210 (0.2%)



1. Moins de Boxing et Unboxing

Pour éviter le Boxing et Unboxing, nous avons utilisé que des types originaux comme `int` dans ce sujet.

2. Matrice de convolution en static

Les matrices de convolution sont initialisées dans le constructeur de la classe `CPainting` dans la version originale. Nous pensons que nous pouvons mettre ces matrices en static pour qu'elles initialisent pendant la charge de classe.

```
private static int[][] mMatriceConv9 = ...;
private static int[][] mMatriceConv25 = ...;
private static int[][] mMatriceConv49 = ...;
```

3. Changement de type de données : float - int

Après des études sur le projet, on pense qu'il n'a pas besoin d'utiliser le type `float` pour la plus part de temps, `int` est suffisant, donc on modifie le type où il utilisait `float`.

```
// matrice servant pour le produit de convolution
- private static float[][] mMatriceConv9 = {{1 / 16f, 2 / 16f, 1 / 16f}, {2 / 16f, 4 / 16f, 2 / 16f}, {1 / 16f, 2 / 16f, 1 / 16f}};
- private static float[][] mMatriceConv25 = {{1 / 44f, 1 / 44f, 2 / 44f, 1 / 44f, 1 / 44f}, {1 / 44f, 2 / 44f, 1 / 44f, 2 / 44f, 1 / 44f}, {2 / 44f, 1 / 44f, 2 / 44f, 1 / 44f, 1 / 44f}, {1 / 44f, 2 / 44f, 1 / 44f, 2 / 44f, 1 / 44f}, {1 / 44f, 1 / 44f, 2 / 44f, 1 / 44f, 1 / 44f}};
+ private static int[][] mMatriceConv9 = {{1, 2, 1}, {2, 4, 2}, {1, 2, 1}};
+ private static float[][] mMatriceConv25 = {{1, 1, 2, 1, 1}, {1, 2, 3, 2, 1}, {2, 3, 4, 3, 2}, {1, 2, 3, 2, 1}, {1, 1, 2, 1, 1}};
private static float[][] mMatriceConv49 =
- {{1 / 128f, 1 / 128f, 2 / 128f, 2 / 128f, 2 / 128f, 1 / 128f, 1 / 128f},
- {1 / 128f, 2 / 128f, 3 / 128f, 4 / 128f, 3 / 128f, 2 / 128f, 1 / 128f},
- {2 / 128f, 3 / 128f, 4 / 128f, 5 / 128f, 4 / 128f, 3 / 128f, 2 / 128f},
- {2 / 128f, 4 / 128f, 5 / 128f, 8 / 128f, 5 / 128f, 4 / 128f, 2 / 128f},
- {2 / 128f, 3 / 128f, 4 / 128f, 5 / 128f, 4 / 128f, 3 / 128f, 2 / 128f},
- {1 / 128f, 2 / 128f, 3 / 128f, 4 / 128f, 3 / 128f, 2 / 128f, 1 / 128f},
- {1 / 128f, 1 / 128f, 2 / 128f, 2 / 128f, 2 / 128f, 1 / 128f, 1 / 128f}};
+ {{1, 1, 2, 2, 2, 1, 1},
+ {1, 2, 3, 4, 3, 2, 1},
+ {2, 3, 4, 5, 4, 3, 2},
+ {2, 4, 5, 8, 5, 4, 2},
+ {2, 3, 4, 5, 4, 3, 2},
+ {1, 2, 3, 4, 3, 2, 1},
+ {1, 1, 2, 2, 2, 1, 1}};

+ case 3:
+     R = R / 16;
+     G = G / 16;
+     B = B / 16;
+     break;
```

```

private void convolution(int x, int y, int dimension) {
    // produit de convolution discrete sur 9 cases
    int i, j, k, l, m, n;
-   float R, G, B;
+   int R, G, B;
    Color lColor;

    for (i = 0; i < dimension; i++) {
        for (j = 0; j < dimension; j++) {
-           R = G = B = 0f;
+           R = G = B = 0;

            for (k = 0; k < dimension; k++) {
                for (l = 0; l < dimension; l++) {
                    m = (x + i + k - (dimension - 1) + mDimension.width) % mDimension.width;
                    n = (y + j + l - (dimension - 1) + mDimension.height) % mDimension.height;
                    if(dimension == 3){
-                       R += CPainting.mMatriceConv9[k][l] * mCouleurs[m][n].getRed() / 16f;
-                       G += CPainting.mMatriceConv9[k][l] * mCouleurs[m][n].getGreen() / 16f;
-                       B += CPainting.mMatriceConv9[k][l] * mCouleurs[m][n].getBlue() / 16f;
+                       R += CPainting.mMatriceConv9[k][l] * mCouleurs[m][n].getRed();
+                       G += CPainting.mMatriceConv9[k][l] * mCouleurs[m][n].getGreen();
+                       B += CPainting.mMatriceConv9[k][l] * mCouleurs[m][n].getBlue();
                    }
                }
            }
        }
    }
}

```

4.mCouleur 2D en 1D

Nous avons changé la structure de données de `mCouleur` d'un tableau à deux dimensions à un tableau à une dimension.

Comparaison:

- Lors de parcourir toutes les données d'un tableau, un tableau 2D a besoin plus de boucles `for`.
- Lors de l'allocation de mémoire, le code de tableau unidimensionnel est plus concis et le tableau multidimensionnel doit être attribué pas à pas à l'aide de boucles. Le tableau multidimensionnel a besoin de plus de mémoire pour stocker les pointeurs qui pointent sur la deuxième dimension.
- Les tableaux unidimensionnels sont plus pratiques pour lire et écrire des données dans un fichier.

```

// il sert aussi pour la fonction paint du Canvas
-   private Color[][] mCouleurs;
+   private Color[] mCouleurs;
    // couleur du fond

    // initialisation de la matrice des couleurs
-   mCouleurs = new Color[mDimension.width][mDimension.height];
+   mCouleurs = new Color[mDimension.width * mDimension.height];
    synchronized (mMutexCouleurs) {
        for (i = 0; i != mDimension.width; i++) {
            for (j = 0; j != mDimension.height; j++) {
-               mCouleurs[i][j] = new Color(mCouleurFond.getRed(), mCouleurFond.getGreen(), mCouleurFond.getBlue());
+               mCouleurs[i * mDimension.width + j] = new Color(mCouleurFond.getRed(), mCouleurFond.getGreen(), mCouleurFond.getBlue());
            }
        }
    }

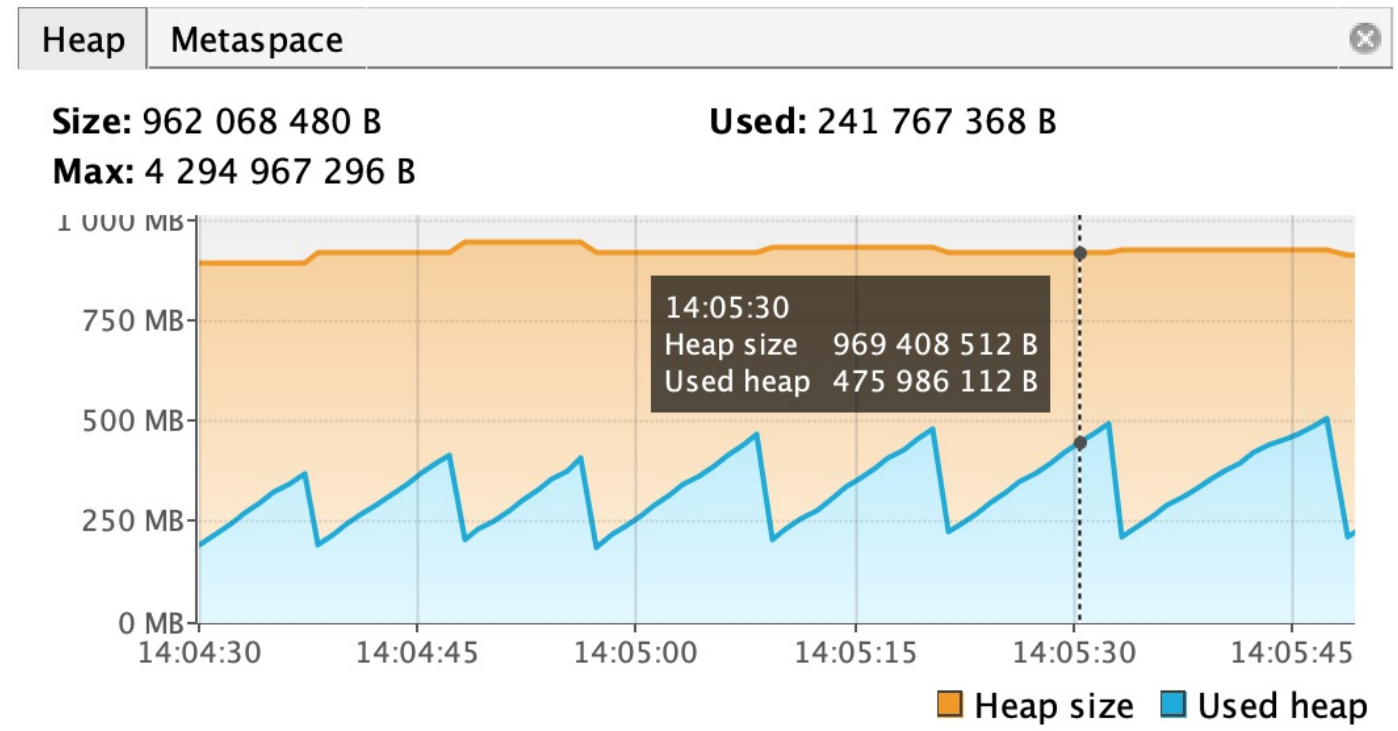
    public Color getCouleur(int x, int y) {
        synchronized (mMutexCouleurs) {
-           return mCouleurs[x][y];
+           return mCouleurs[x * mDimension.width + y];
        }
    }
}

```

...

```
-         if(dimension == 3){
-             R += CPainting.mMatriceConv9[k][l] * mCouleurs[m][n].getRed();
-             G += CPainting.mMatriceConv9[k][l] * mCouleurs[m][n].getGreen();
-             B += CPainting.mMatriceConv9[k][l] * mCouleurs[m][n].getBlue();
-         }else if(dimension == 5){
-             R += CPainting.mMatriceConv25[k][l] * mCouleurs[m][n].getRed();
-             G += CPainting.mMatriceConv25[k][l] * mCouleurs[m][n].getGreen();
-             B += CPainting.mMatriceConv25[k][l] * mCouleurs[m][n].getBlue();
-         }else{
-             R += CPainting.mMatriceConv49[k][l] * mCouleurs[m][n].getRed();
-             G += CPainting.mMatriceConv49[k][l] * mCouleurs[m][n].getGreen();
-             B += CPainting.mMatriceConv49[k][l] * mCouleurs[m][n].getBlue();
-         }
+         if (dimension == 3) {
+             R += CPainting.mMatriceConv9[k][l] * mCouleurs[m * mDimension.width + n].getRed();
+             G += CPainting.mMatriceConv9[k][l] * mCouleurs[m * mDimension.width + n].getGreen();
+             B += CPainting.mMatriceConv9[k][l] * mCouleurs[m * mDimension.width + n].getBlue();
+         } else if (dimension == 5) {
+             R += CPainting.mMatriceConv25[k][l] * mCouleurs[m * mDimension.width + n].getRed();
+             G += CPainting.mMatriceConv25[k][l] * mCouleurs[m * mDimension.width + n].getGreen();
+             B += CPainting.mMatriceConv25[k][l] * mCouleurs[m * mDimension.width + n].getBlue();
+         } else {
+             R += CPainting.mMatriceConv49[k][l] * mCouleurs[m * mDimension.width + n].getRed();
+             G += CPainting.mMatriceConv49[k][l] * mCouleurs[m * mDimension.width + n].getGreen();
+             B += CPainting.mMatriceConv49[k][l] * mCouleurs[m * mDimension.width + n].getBlue();
+         }
```

Impact sur la performance



La mémoire requise pour cette méthode est plus petite et donc les performances de la mémoire légèrement optimisées.

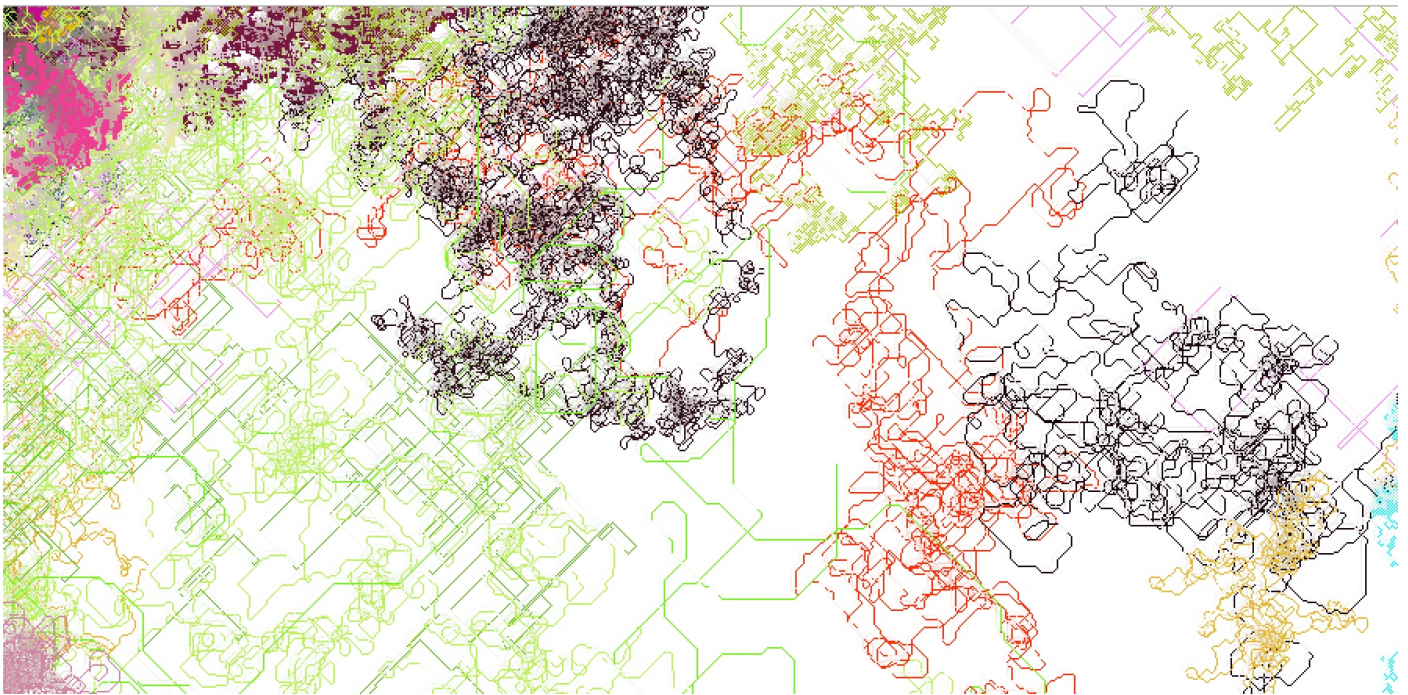
5.BufferStrategy

Nous avons essayé de utiliser le BufferStrategy dans la classe `CPainting` pour améliorer la performace (La branche `bufferStrategy`).

Après modifier le code, le programme exécute plus rapide.

running (45098)

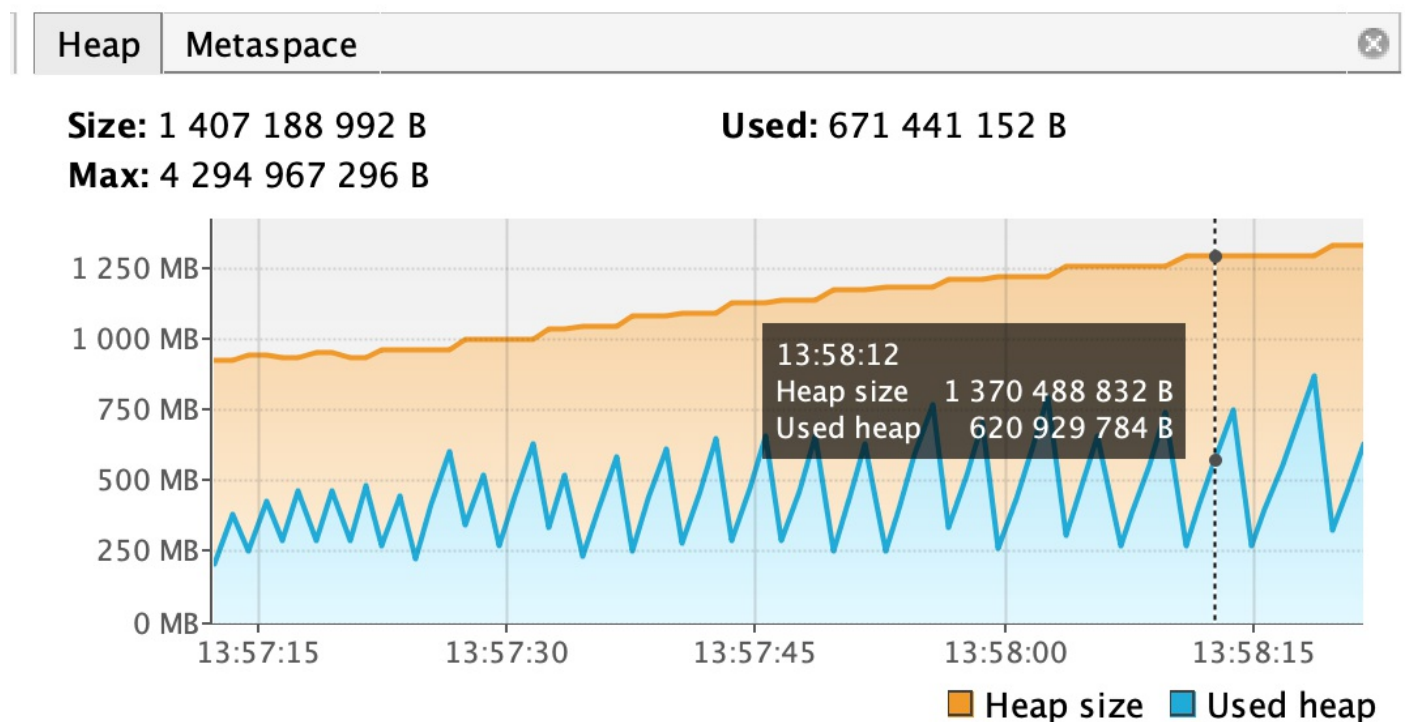
Mais le résultat n'est pas correct (notamment la diffusion) et nous n'avons pas fixer le bug.



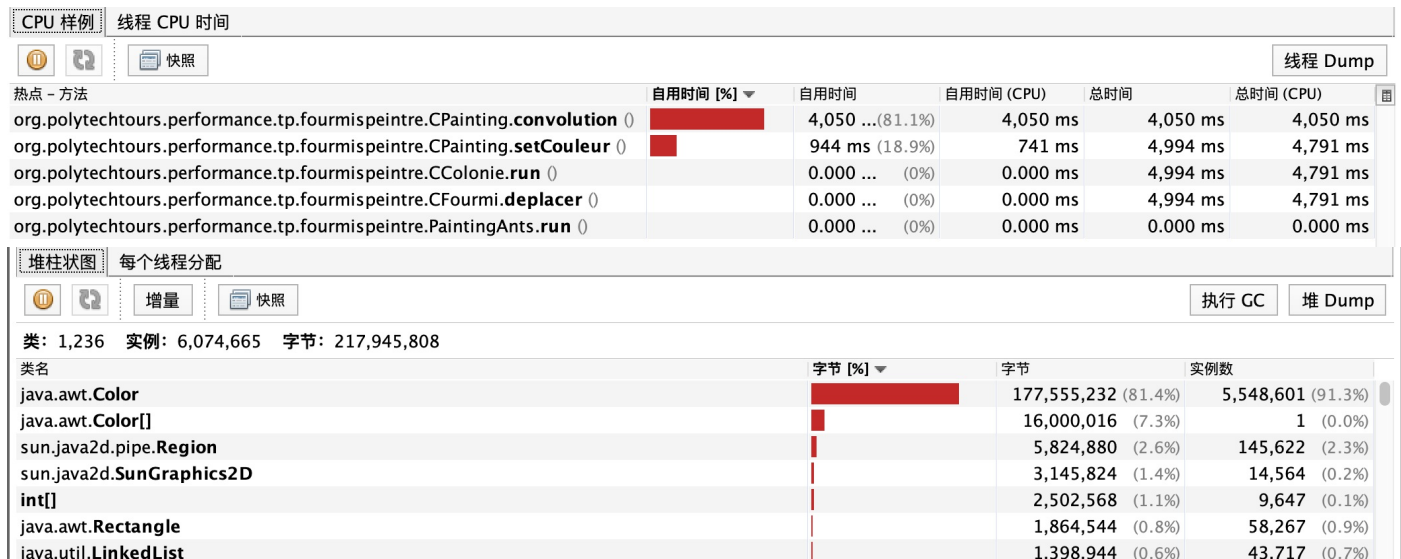
Pour le nombre de buffer, nous avons essayé de utiliser plus de buffer (2 par exemple). Mais il exécute moins rapide si le nombre de buffer est plus que 1.

```
bs = getBufferStrategy();
if(bs == null) { //True if our canvas has no buffer strategy (should only happen once
when we first start the game)
    createBufferStrategy(1); //Create a buffer strategy using two buffers (double
buffer the canvas)
    return ;
}
```

Impact en performance



Selon cette image on peut trouver que la quantité totale de mémoire utilisée est similaire à celle d'origine. Bufferstrategy n'a pas d'optimisation sur la mémoire, mais le garbage collector est appelé plus fréquemment.



6.BufferedImage & Raster

Après l'essai de `BufferStrategy`, nous avons trouvé une autre classe `BufferedImage`.

La classe `BufferedImage` décrit une image avec un tampon accessible de données d'image. `BufferedImage` est composé d'un `Raster`.

Nous pouvons utiliser l'objet `WritableRaster` d'une `BufferedImage` pour manipuler l'image. Donc nous n'avons plus besoin de l'objet `Color` pour stocker l'information de couleur d'un pixel. À la place de `Color`, nous utilisons directement un tableau de `int` pour représenter couleur `RGB`.

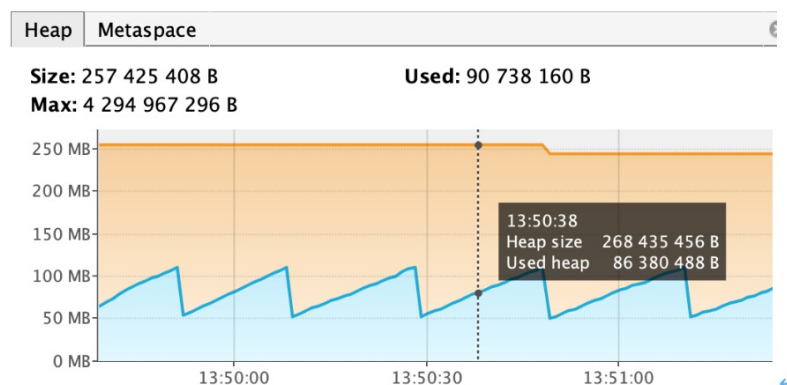
Nous pouvons aussi diminuer la charge de l'affichage. Comme nous pouvons stocker l'information de pixel dans le `Raster`, nous pouvons afficher l'image après la fin de convolution pour chaque déplacement. Nous pouvons aussi faire afficher après un délais donné (affiche l'image chaque 10 déplacements par exemple).

Impact sur la performance










Après l'utilisation de `BufferedImage` et `Raster`, le programme exécute plus rapide et le résultat est bon.




running (34357)

Jvisualvm



Avec `Raster`, la performance sur mémoire est grandement améliorée. Avant, Heap size est 1250M, maintenant est juste 250M. Et c'est pareil en Used Heap.

CPU 样例		线程 CPU 时间					
				 快照		线程 Dump	
热点 - 方法		自用时间 [%] ▼		自用时间	自用时间 (CPU)	总时间	总时间 (CPU)
org.polytechtours.performance.tp.fourmispeintre.PaintingAnts.run ()				3,1... (33.7%)	0.000 ms	3,159 ms	0.000 ms
org.polytechtours.performance.tp.fourmispeintre.CPainting.paint ()				3,0... (32.7%)	0.000 ms	3,067 ms	0.000 ms
org.polytechtours.performance.tp.fourmispeintre.CPainting.setCouleur ()				3,0... (32.6%)	3,060 ms	3,060 ms	3,060 ms
org.polytechtours.performance.tp.fourmispeintre.PaintingAnts.stop ()				87.... (0.9%)	0.000 ms	87.6 ms	0.000 ms
org.polytechtours.performance.tp.fourmispeintre.CColonie.run ()				0.0... (0%)	0.000 ms	3,060 ms	3,060 ms
org.polytechtours.performance.tp.fourmispeintre.CFourmi.deplacer ()				0.0... (0%)	0.000 ms	3,060 ms	3,060 ms

堆柱状图		每个线程分配					
				 快照		执行 GC	
 增量						堆 Dump	

类: 1,296 实例: 239,317 字节: 25,923,016

类名	字节 [%] ▼	字节	实例数
int[]	<div></div>	16,413,000 (63.3%)	1,666 (0.6%)
java.lang.Object[]	<div></div>	1,570,760 (6.0%)	44,039 (18.4%)
char[]	<div></div>	1,562,136 (6.0%)	26,492 (11.0%)
java.util.TreeMap\$Entry	<div></div>	610,960 (2.3%)	15,274 (6.3%)
java.io.ObjectStreamClass\$WeakClassKey	<div></div>	533,312 (2.0%)	16,666 (6.9%)
java.lang.String	<div></div>	441,936 (1.7%)	18,414 (7.6%)
java.util.HashMap	<div></div>	336,288 (1.2%)	7,006 (2.9%)
java.lang.Class	<div></div>	330,552 (1.2%)	2,889 (1.2%)
java.util.HashMap\$Node	<div></div>	321,728 (1.2%)	10,054 (4.2%)

Nous pouvons trouver que la méthode `convolution` n'utilise pas beaucoup le CPU après la modification, parce que avec le `BufferedImage`, nous stockons les résultats de convolution dans le `Raster` et nous dessinons l'image une seule fois pour chaque déplacement. Par contre, dans la version originale, il dessine sur l'image après le calcul de chaque pixel.

Avec `BufferedImage`, nous n'utilisons plus la classe `Color` pour représenter le couleur de chaque pixel. Nous utilisons un tableau `int` de dimension 3 pour les couleurs. Donc le GC se lance moins fréquent et le programme utilise moins de mémoire.