Tipos de estrategias de GC

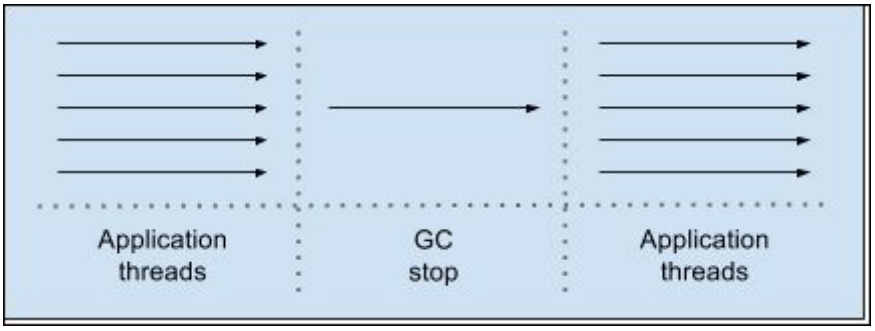
Hay varios tipos de colectores diferentes de la GC. Estos tipos también se conocen como las estrategias y están, a su vez, se dieron cuenta por diferentes implementaciones dependiendo del fabricante JVM. Hoy en día, existen muchas implementaciones diferentes y estos son mezclas de las estrategias básicas. En primer lugar, vamos a echar un vistazo a la mayoría estrategias comunes del Hotspot VM:

- El serial collector
- El parallel collector
- El colector concurrente
- La **Garbage First (G1)**

Cabe señalar que, dentro de una estrategia, diferentes algoritmos de GC se pueden habilitar para los jóvenes generación y la vieja generación, respectivamente, todo para optimizar la gestión de memoria y rendimiento.

El serial collector

El serial collector utiliza un solo hilo como se muestra en el siguiente diagrama. Este hilo se detiene todos los otros hilos, una JVM llamado comportamiento de parada-el-mundo. Si bien es un colector relativamente eficiente, ya que no hay sobrecarga de comunicación entre los hilos, no puede tomar ventaja de las máquinas de multiprocesador. Por lo tanto, es el más adecuado para un solo procesador máquinas y como sabemos, estos ya no son comunes.

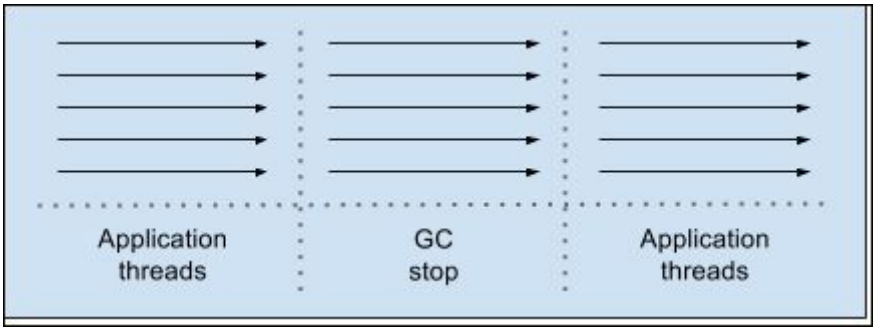


El colector de serie en el trabajo
del colector de serie se selecciona de forma predeterminada en las configuraciones de hardware y sistema operativo que no son elegidos como máquinas de tipo servidor o se pueden activar de forma explícita con el siguiente parámetro VM:
-XX: + UseSerialGC

Concurrent collector

El colector paralelo, también conocido comúnmente como el colector de rendimiento, es el colector por defecto en las máquinas de grado servidor para Java sE 7. también se puede activar de forma explícita con el siguiente parámetro VM:
-XX: + UseParallelGC

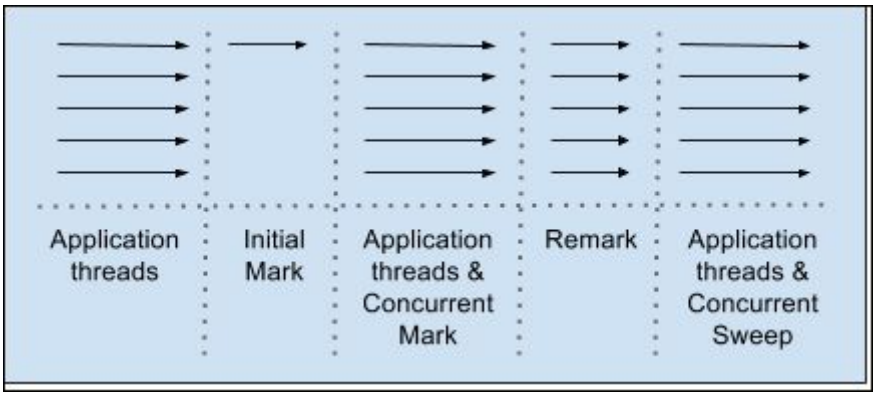
el colector paralelo realiza colecciones de menor importancia en paralelo, lo que puede reducir significativamente mejorar el rendimiento de las aplicaciones que tienen un montón de colecciones menores.
Como se puede ver en el siguiente diagrama, el colector paralelo todavía requiere una denominada *stop-the-world.* actividad Sin embargo, puesto que las colecciones se realizan en paralelo, haciendo buen uso de muchas CPU, disminuye la sobrecarga del garbage collector y por lo tanto aumenta el rendimiento de la aplicación.



El colector paralelo en el trabajo
Desde el lanzamiento de J2SE 5.0 actualización 6, puede beneficiarse de una característica llamada *compactación paralelo* que permite que el colector paralelo a realizar también importantes colecciones en paralelo. Sin compactación paralelo, grandes colecciones se realizan usando un único hilo, que puede limitar significativamente la escalabilidad.
Este colector incluye una fase de compactación donde GC identifica las regiones que están libres y utiliza sus hilos para copiar datos en esas regiones. Esto produce un montón que está lleno densamente en un extremo con un gran bloque vacío en el otro extremo. En la práctica, esto ayuda a reducir la fragmentación del montón, que es crucial cuando se está tratando de asignar objetos de gran tamaño.
Compactación paralelo está habilitado de forma predeterminada a partir de la actualización de Java SE 7 4. De otro modo, se puede activar de forma explícita por el parámetro **UseParallelOldGC**.
Muchos piensan que la antigua palabra en nombre de GC se refiere a él como viejos o en desuso. Nada podría estar más equivocado. Antiguo simplemente se refiere al área de memoria generación de edad y está GC frecuencia, se prefiere el colector paralelo regular.

El colector concurrente

El colector concurrente es un colector de baja pausa más comúnmente conocido como e collector **Concurrent Mark Sweep** (**CMS)** Se realiza la mayor parte de su trabajo al mismo tiempo que la aplicación sigue ejecutando. Esto optimiza el rendimiento manteniendo pausas GC corto.
Básicamente, este colector consume recursos del procesador con el fin de tener mayores tiempos de pausa colección más cortos. Esto puede suceder debido a que el colector concurrente utiliza un solo hilo garbage collector que se ejecuta simultáneamente con las roscas de la aplicación. Por lo tanto, el propósito del colector es concurrente para completar la colección de la generación titular antes de que se llene.
El siguiente diagrama nos da una idea de cómo funciona el colector concurrentes:



El colector concurrentes en el trabajo

En un primer momento, el colector identifica los objetos vivos, que son directamente accesibles en una fase **Initial Mark**.A continuación, en la fase **Concurrent Mark**,el colector marca todos los objetos vivos que son accesibles, mientras que la aplicación aún está en marcha.una posterior **Remark** Es necesaria etapa de volver a examinar los objetos que están modificados en la fase de marcado concurrente. Por último, la fase **Concurrent Sweep** recupera todos los objetos que se han marcado.

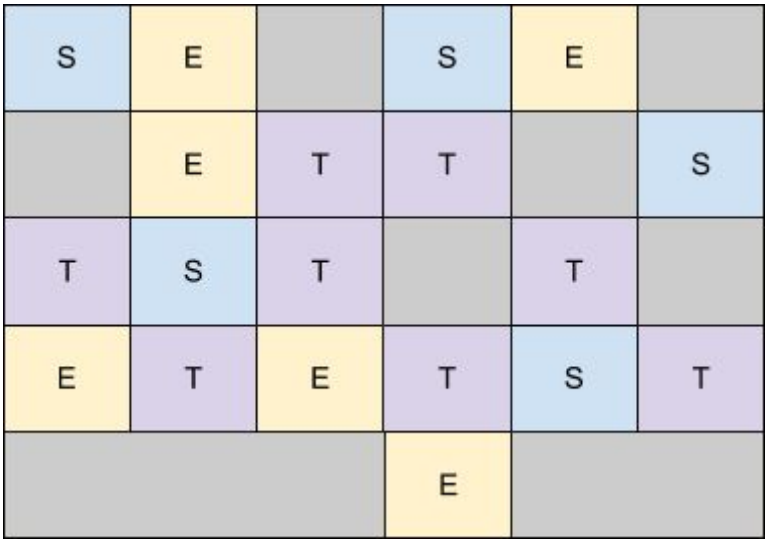
El reverso de la moneda es que esta técnica, que se utiliza para minimizar las pausas, en realidad puede reducir el rendimiento general de la aplicación. Por lo tanto, está diseñado para aplicaciones cuyo tiempo de respuesta es más importante que el rendimiento global.

El colector concurrente está activada con el -XX: + parámetro UseConcMarkSweepGC VM.

El colector G1

El colector G1 está incluido y totalmente compatible-en el Oracle JDK 7 actualización 4 distribución de Java SE 7. G1 y está dirigido a entornos de servidores con CPUs multinúcleo equipadas con grandes cantidades de memoria. Se llama un colector parallel-concurrent regionalizado y se habilita mediante el uso de la -XX: parámetro + UseG1GC VM.

Cuando se utiliza G1, el montón se divide en regiones de igual tamaño entre 1 y 32 MB. JVM establece este tamaño en el arranque. El objetivo es tener no más de 2048 regiones en una máquina virtual. El **Eden** (E), **Survivor (S),**y **Tenured (T)**de la generación se dividen en conjuntos lógicos no continuas de estas regiones, como se visualiza en el diagrama siguiente:



El montón con sus áreas de sub memoria; usando el collector G1
G1 no es un colector en tiempo real, sino que trata de cumplir los objetivos de tiempo de pausa definidos por el usuario utilizando un modelo de predicción de pausa mediante el ajuste de la cantidad de regiones utilizadas por las diferentes áreas de memoria. Fragmentaciones del heap se reducen por la compactación realizada por la copia paralelo de objetos entre conjuntos de regiones llamadas **Collection Sets** (cSet).
Las referencias a objetos en las regiones son seguidos por independientes **Remembered Sets** (**RSet**)**.**

Estos permiten colección paralelo e independiente entre las regiones desde la exploración puede ser minimizado a una región en lugar de todo el montón.
Vale la pena señalar que, para el colector G1, la JVM puede ser un poco más grande en comparación con otros coleccionistas. Esto se debe a la **cSet** y estructuras de datos **RSet** y no es nada de qué preocuparse.

El objetivo de G1 es permanecer siempre dentro de un momento de pausa y recuperar tanta memoria como sea posible, a partir de las áreas que contienen el espacio más recuperable.
Todo esto hace que el colector G1 muy estable en términos de interrupciones mínimas para la recogida y la compactación y eficaz en términos de uso de memoria.

La página web de Oracle establece que:

Applications that require a large heap, have a big active data set, have bursty or non-uniform workloads or suffer from long garbage collection induced latencies should benefit from switching to G1.

Esto sin duda puede ser cierto para muchas aplicaciones, pero, en realidad, el colector G1 está todavía poco utilizado en la producción. Esto podría ser debido al hecho de que es relativamente nuevo y, como tal, por lo que pocos saben al respecto. Muchos ni siquiera se molestan en absoluto GC sintonía. Para la mayoría de los demás, sin embargo, el CMS es "suficientemente bueno" para la mayoría de aplicaciones.