Types of JVM Garbage Collectors in Java with implementation details

Difficulty Level : Medium Last Updated : 22 Jan, 2020

prerequisites: Garbage Collection, Mark and Sweep algorithm

Garbage Collection**:** Garbage collection aka GC is one of the most important features of Java. Garbage collection is the mechanism used in Java to de-allocate unused memory, which is nothing but clear the space consumed by unused objects. To deallocate unused memory, Garbage collector track all the objects that are still in use and it marks the rest of the object as garbage. Basically garbage collector use Mark and Sweep algorithm to clear unused memory.

Types of Garbage Collection:

The JVM actually provides four different garbage collectors. Each garbage collector will vary in Application throughput and Application pause. Application throughput denotes the speed at which a Java application runs and Application pause means the time taken by the garbage collector to clean the unused memory spaces.

1. **Serial Garbage Collector**: This is the simplest GC implementation, as it basically works with a single thread. If we select Serial garbage collector as our default garbage collector then whenever we will call garbage collector to clean unused memory then serial garbage collector holds all the running threads of application i.e. It works by freezing all the application threads and It will create a single thread to perform garbage collection. It freezes all other running threads of application until garbage collection operations have concluded. If we use Serial Garbage Collector as our default garbage collector then the application throughput will decrease and application pause time will increase. As a result, this GC implementation freezes all application threads when it runs. Hence, it is not a good idea to use it in multi-threaded applications like server environments.
   Implementation:
   If you want to use serial garbage collector, then we have to explicitly mention while running jar like:

   java -XX:+UseSerialGC -jar GFGApplicationJar.java

2. **Parallel Garbage Collector**: Parallel Garbage Collector is the default garbage collector in Java 8. It is also known as **Throughput collector**. Parallel Garbage Collector is same as Serial Garbage Collector because Parallel Garbage Collector also freezes the running threads of the application while performing the garbage collection. But the difference is, Parallel Garbage Collector uses multiple threads to perform cleaning of unused heap area. The advantage of using Garbage Collector as default GC is we can mention few attributes for the garbage collector, like
   - How many threads, can garbage collector use to perform garbage collection.
     Implementation:

     *java -XX:+UseParallelGC -XX:ParallelGCThreads=**NumberOfThreads** -jar GFGApplicationJar.java*

   - Maximum pause can garbage collector take while performing garbage collection
     Implementation:

     *java -XX:+UseParallelGC -XX:MaxGCPauseMillis=**SecInMillisecond** -jar GFGApplicationJar.java*

   The parallel garbage collector is far better than serial garbage collector but one problem with the parallel garbage collector is it pauses the application during minor operations also. It is best suited if applications that can handle such pauses. If we are using JDK 8 then parallel GC is the default garbage collector.

   Implementation: If we are running on java 9 and want to use parallel garbage collector then we should use below command:

   java -XX:+UseParallelGC -jar GFGApplicationJar.java

3. **CMS Garbage collector**: CMS Garbage collector is known as concurrent mark-sweep garbage collector. This garbage collector uses multiple threads to scan the heap memory consistently to the mark objects that are unused and then sweep the marked objects. As we know, Serial garbage collector and Parallel garbage collector freeze the

running threads of the application while performing the garbage collection. But CMS Garbage collector will perform freezing of running threads i.e. application pause in two cases only:

- While performing the garbage collection, If there is a change in heap memory in parallel.
- While marking the referenced objects in the old generation space.

If we compare CMS collector with Parallel garbage collector, CMS collector uses more CPU to ensure better application throughput. If we are developing an application where we can provide more CPU resources for better performance then CMS garbage collector is the blockquoteferred choice over the parallel collector. To enable CMS Garbage Collector, we can use the following argument:

> *java -XX:+UseParNewGC -jar GFGApplicationJar.java*

4. **G1 Garbage Collector**: Firstly G1 Garbage Collector is introduced in JDK 7. Initially, It was designed to provide better support for larger heap memory application. G1 Garbage Collector is the default garbage collection of Java 9. G1 collector replaced the CMS collector since it’s more performance efficient. How G1 Garbage Collector works is different from other collectors. Unlike other collectors, the G1 collector partitions the heap space into multiple equal-sized regions. Basically it is mainly designed for an application having heap size greater than 4GB. It divides the heap area into multiple regions vary from 1MB to 32MB. While performing the garbage collection, G1 Garbage Collector mark the heap region which has objects that are in use throughout the heap. By the help of this garbage collector has the information about the regions that contains most use less objects and garbage collector first perform the garbage collection on that region only. Thats why it is known as G first garbage collector. G1 also does compact the free heap space just after garbage collection that makes G1 Garbage Collector better than other garbage collectors. G1 Garbage Collector is the default garbage collector of Java 9.
   Implementation: If we are using Java version less than 9 and we want to use G1 Garbage Collector then we have to mention explicitly while running jar file like:

   > *java -XX:+UseG1GC -jar GFGApplicationJar.java*