JVM INTERNALS

Epsilon: The JDK's Do-Nothing Garbage Collector

The benefits of Java's memory allocator that does no garbage collection

by Andrew Binstock

November 21, 2019

Performance tuning on the JDK is a fine art that often centers on choosing the optimal garbage collector and tweaking its settings to achieve the smallest impact for a given workload. A persistent challenge in doing this work is knowing how fast the workload would run if there were no garbage collection at all. To address this need (and some others that I'll touch on shortly), the Epsilon garbage collector (GC) was included in JDK 11. This GC, as defined in JEP 318, allocates memory but does not recycle it. That is, it does no garbage collection.

To tell the JVM to use the Epsilon GC, you need to specify these two runtime switches on the command line.

```
-XX:+UnlockExperimentalVMOptions
-XX:+UseEpsilonGC
```

If you run the following code with and without these switches, you can see the difference Epsilon makes.

```
public class EpsilonDemo {

    public static void main(String[] args) {

        final int GIGABYTE = 1024 * 1024 * 1024;
        final int ITERATIONS = 100;

        System.out.println("Starting allocations...

        // allocate memory 1GB at a time
        for (int i = 0; i < ITERATIONS; i++) {
            var array = new byte[GIGABYTE];
        }

        System.out.println("Completed successfully"
    }
}
```

This code attempts to allocate 100 GB of memory. That's all it does—simply allocates and exits. If you run it without specifying Epsilon, you see the two literals from the beginning and ending `println` statements, which means the default GC allocated 100 blocks of 1 GB and garbage-collected them as needed to fit within your system's available memory.

However, if you run it using the Epsilon GC via the command-line switches, the program output is:

```
Starting allocations...
Terminating due to java.lang.OutOfMemoryError: Java
```

Because the early allocations were not garbage-collected by Epsilon, the heap exceeded available memory and the JDK generated the comparatively uncommon “Out of Memory” error.

To run this code, you’ll need JDK 11 or later and an available heap of less than 100 GB. If you run it on larger systems, simply increase the ITERATIONS variable as needed and you’ll be able to generate the error.

There is a strong temptation to use Epsilon on deployed programs, rather than to confine it to performance tuning work. As a rule, the Epsilon team discourages this use, with two exceptions. Short-running programs, like all programs, invoke the garbage collector at the end of their run. However, as JEP 318 explains, “accepting the garbage collection cycle to futilely clean up the heap is a waste of time, because the heap would be freed on exit anyway.”

The Epsilon team foresaw that under certain situations that are especially sensitive to latency, Epsilon could be used. Quoting from JEP, “For ultra-latency-sensitive applications, where developers are conscious about memory allocations and know the application memory footprint exactly, or even have (almost) completely garbage-free applications, accepting the GC cycle might be a design issue.”

But other uses are discouraged. Even programs that are known to use little heap space are at risk for blowing up if they are run with Epsilon on a system with unanticipated memory constraints. For this reason, Epsilon requires the command-line switch that tells you it’s an experimental feature. Technically speaking, it is an integral part of JDK 11 (and subsequent JDKs), but the “experimental” term should serve as a reminder.

Getting back to performance tuning, if you’ve ever wondered how much of your program’s performance is affected by garbage collection, Epsilon is your solution.

Also in This Issue

Andrew Binstock

Andrew Binstock (javamag_us@oracle.com, @platypusguy) is the editor in chief of *Java Magazine*. Previously, he was the editor of *Dr. Dobb's Journal*. He co-founded the company behind the open-source iText PDF library, which was acquired in 2015. His book on algorithm implementation in C went through 16 printings before joining the long tail. Previously, he was the editor in chief of *UNIX Review* and, earlier, the founding editor of the *C Gazette*. He lives in Silicon Valley with his wife. When not coding or editing, he studies piano.

Share this Page

Contact

US Sales: +1.800.633.0738
Global Contacts
Support Directory
Subscribe to Emails

About Us

Careers
Communities
Company Information
Social Responsibility Emails

Downloads and Trials

Java for Developers
Java Runtime Download
Software Downloads
Try Oracle Cloud

News and Events

Acquisitions
Blogs
Events
Newsroom

ORACLE | Integrated Cloud
Applications & Platform Services

© Oracle | Site Map | Terms of Use & Privacy | Cookie Preferences | Ad Choices