
ORACLE®


Java™
ORACLE®

Java Safepoints

Bangalore JUG

Fairoz Matte
Principle Member Of Technical Staff
Java Developer Community Support Engineer
April 08, 2017

Java Your Next (Cloud)

Program Agenda

1. Problem Statement
2. What is Safe point?
3. Why do we need to halt at safepoint
4. When Thread is at safepoint
5. How Safepoint works
6. Demo

Problem Statement?

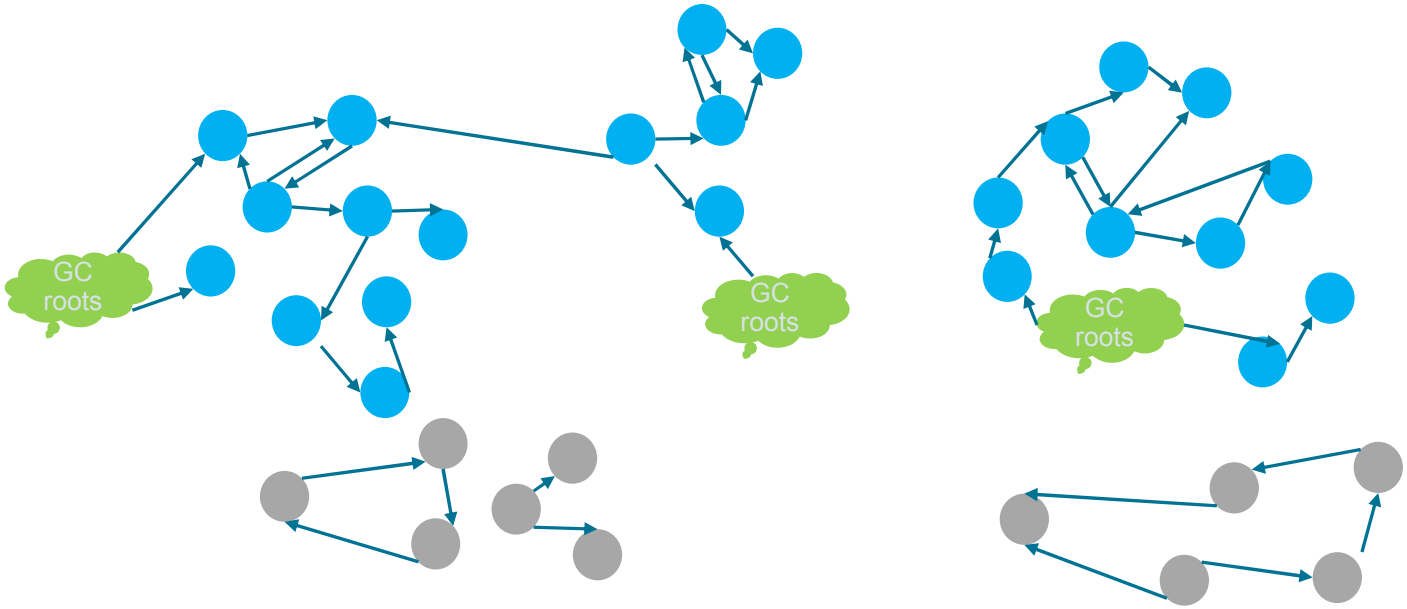
Problem Statement – GC Approach

- Find all the objects that are live (referenced)

  Scanning through GC roots (Special objects) and finding the references from root to other objects.

- Moving objects in heap or getting exclusive access to JVM data structures

Problem Statement – GC Approach



Problem Statement - GC Approach

- The application threads need to be stopped for the marking to happen as you cannot really traverse the graph if it keeps changing.
- All the threads need to be suspended/paused for a period of time and they can be resumed from the last paused state
- Thread under execution need to be brought to consistent state

What is Safe point

- A safepoint is a state of your application execution where all references to objects are perfectly reachable by the VM.
- Safepoint is known point in execution where state is known, can be examined and updated
- A safepoint means that all threads need to reach a certain point of execution before they are stopped. Then the VM operation is performed. After that all threads are resumed.

Why do we need to halt at safepoint

- Perform operations that are atomic to all application thread
- It is easier

When Thread is at safepoint

- A Java thread is at a safepoint if it is blocked on a lock or synchronized block, waiting on a monitor parked, or blocked on blocking IO
- A Java thread is at a safepoint while executing JNI code. Before crossing the native call boundary the stack is left in a consistent state before handing off to the native code
- safepoint-safe states [_thread_blocked, _thread_in_native ,_thread_in_VM]
- _thread_in_Java is not a safe point state

How Safepoint works

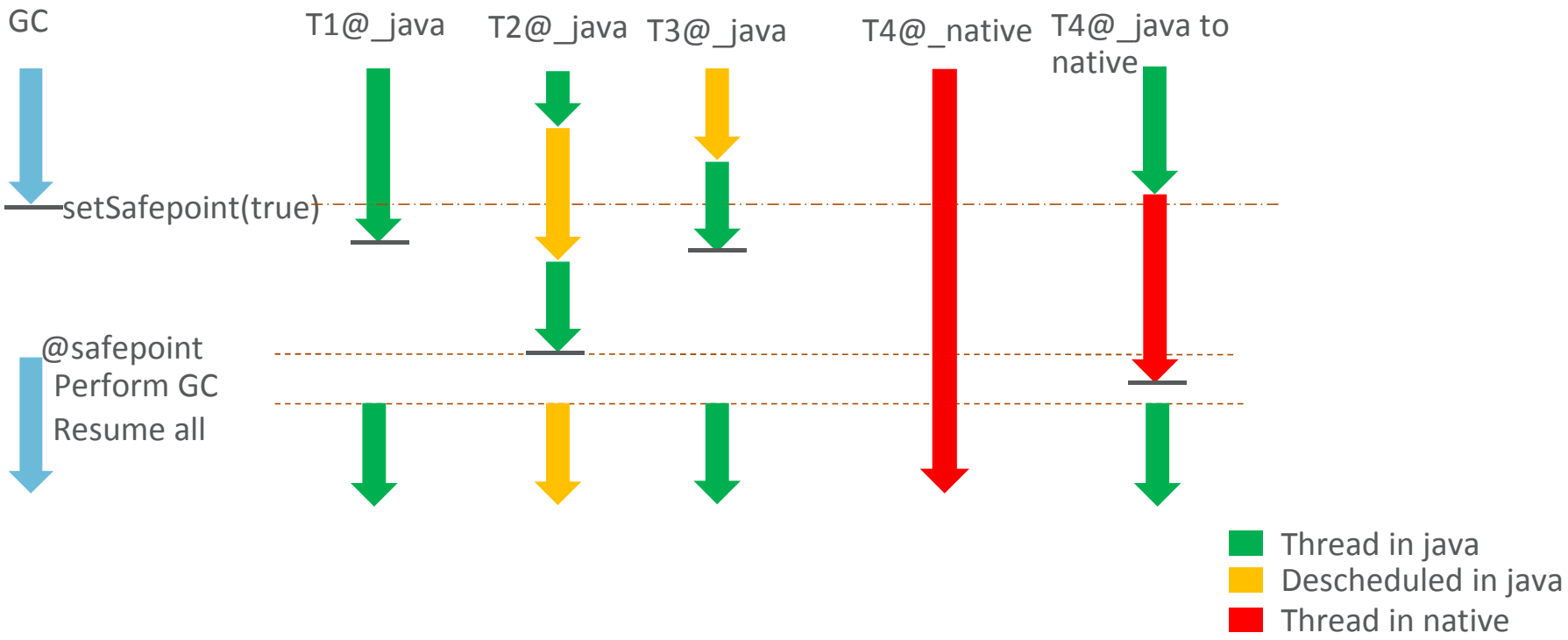
- Java threads poll a 'safepoint flag' at certain intervals and transition into a safepoint state (thread is blocked at a safepoint) when they observe a 'Go to safepoint' flag

- safepoint polls

  - ✓ Between any 2 bytecodes while running in the interpreter
  - ✓ On 'non-counted' loop back edge in C1/C2 compiled code
  - ✓ Method entry/exit

'{poll}' or **'{poll return}'**

How Safepoint works



How Safepoint works

- Application/Profiler thread request JVM to perform safepoint operation by setting setSafepoint(true) flag
- Once other application threads see the setSafepoint(true) flag, start themselves parking at next safepoint poll
- Blocked threads are prevented from restarting
- Native threads can still be running
- Once JVM identifies the all applications threads at safepoint, it performs the safepoint operation (In our case running GC) and set the setSafepoint(false)

How Safepoint works

```
public class SafePointCheck {

  public static void main(String[] args) {
    for(int i=0; i<1000; i++){
      try {
        Collection<Object> list = new ArrayList<>();
        list.add(new byte[1024 * 1024]);
        Object obj = new byte[1024 * 1024];
      } catch (OutOfMemoryError e) {

      }
    }
  }
}
```

Demo

Demo

Running with **-XX:+PrintGCApplicationStoppedTime -XX:+PrintGCApplicationConcurrentTime**

```
C:\Users\fmatte.ORADEV\Desktop>"d:\Program Files\Java\jdk1.8.0_121\bin\java.exe" -
XX:+PrintGCApplicationStoppedTime -XX:+PrintGCApplicationConcurrentTime  SafePointCheck
Application time: 0.0296857 seconds
Total time for which application threads were stopped: 0.0093728 seconds, Stopping threads took: 0.0000147
seconds
Application time: 0.0051237 seconds
Total time for which application threads were stopped: 0.0124898 seconds, Stopping threads took: 0.0000107
seconds
Application time: 0.0206502 seconds
Total time for which application threads were stopped: 0.0519142 seconds, Stopping threads took: 0.0000147
seconds
Application time: 0.0089119 seconds
Total time for which application threads were stopped: 0.0184516 seconds, Stopping threads took: 0.0000116
seconds
Application time: 0.0323765 seconds
Total time for which application threads were stopped: 0.0647164 seconds, Stopping threads took: 0.0000116
seconds
Application time: 0.0279346 seconds
Total time for which application threads were stopped: 0.1110481 seconds, Stopping threads took: 0.0000116
seconds
Application time: 0.0749393 seconds
Total time for which application threads were stopped: 0.2330930 seconds, Stopping threads took: 0.0000143
seconds
Application time: 0.0271434 seconds
Total time for which application threads were stopped: 0.1632386 seconds, Stopping threads took: 0.0000116
seconds
Application time: 0.0247801 seconds
```

Demo

Running with **-XX:+PrintSafepointStatistics -XX:PrintSafepointStatisticsCount=1**

```
C:\Users\fmatte.ORADEV\Desktop>"d:\Program Files\Java\jdk1.8.0_121\bin\java.exe" -
XX:+PrintGCApplicationStoppedTime -XX:+PrintGCApplicationConcurrentTime -XX:+PrintSafepointStatistics  -
XX:PrintSafepointStatisticsCount=1  SafePointCheck

Application time: 0.0304269 seconds
         vmop                    [threads: total initially_running wait_to_block]    [time: spin block
sync cleanup vmop] page_trap_count
0.140: ParallelGCFailedAllocation       [       9          0              0    ]      [     0     0     0
0    12    ]  0
Total time for which application threads were stopped: 0.0154595 seconds, Stopping threads took: 0.0000348
seconds
Application time: 0.0059832 seconds
         vmop                    [threads: total initially_running wait_to_block]    [time: spin block
sync cleanup vmop] page_trap_count
0.164: ParallelGCFailedAllocation       [       9          0              0    ]      [     0     0     0
0    13    ]  0
Total time for which application threads were stopped: 0.0165020 seconds, Stopping threads took: 0.0000281
seconds
Application time: 0.0245990 seconds
         vmop                    [threads: total initially_running wait_to_block]    [time: spin block
sync cleanup vmop] page_trap_count
0.208: ParallelGCFailedAllocation       [       9          0              0    ]      [     0     0     0
0    45    ]  0
Total time for which application threads were stopped: 0.0487803 seconds, Stopping threads took: 0.0000290
seconds
Application time: 0.0100797 seconds
```

Demo

Forcing safepoints poll for counted loops
-XX:+UseCountedLoopSafepoints

Q&A

Java
Your
Next
(Cloud)