ex1-JIT

Step 1

Java -XX:+PrintCompilation demo.MainV1 500

Step 2

Java -XX:+UnlockDiagnosticVMOptions -XX:+LogCompilation demo.MainV1 7000

Note : hotspot_pid19000.log file is created.

Search for random() and see the compiler C1 & C2

Step: 3

Arguments-> Program Arguments:7000

VM arguments: -XX:+PrintCodeCache

Step: 4

Tuning the code cache

VM Warning : Codecache is full. Compiler has been disabled

Set the below options :

Codecache Size Options

Option	Default	Description
InitialCodeCacheSize	160K (varies)	Initial code cache size (in bytes)
ReservedCodeCacheSize	32M/48M	Reserved code cache size (in bytes) - maximum code cache size
CodeCacheExpansionSize	32K/64K	Code cache expansion size (in bytes)

Now try with :

VM arguments: -XX:ReservedCodeCacheSize=28m -XX:+PrintCodeCache

jvm compiler flags: -server, -client

Step1:

Arguments-> Program Arguments: 10000

VM arguments -> -server -XX:+PrintCompilation

Step2:

Arguments -> Program Arguments: 10000

VM arguments -> -client -XX:+PrintCompilation

Note:

Disable TieredCompilation

VM arguments -> -XX:-TieredCompilation -XX:+PrintCompilation

use jconsole to monitor the code cache

note: jconsole process id is not showing, give write access to the below folder:

C:\Users\venkat\AppData\Local\Temp\hsperfdata_venkat

> Memory Pool (Code Cache)

JVM Versions

32-bit : Faster if heap < 3GB, Max heap size=4GB, Client Compiler only

64-but : it would be faster if using long/double, heap > 4GB, Client & Server Compiler

Check the default java final flags

java -XX:+PrintFlagsFinal

Check for the application process-id

Check the CICompilerCount value

[CICompilerCount to two tells the JVM to use number of threads for compiler]

jinfo -flag CICompilerCount <process-id>

Now change the count and check the output:

VM arguments: -XX:CICompilerCount=6 -XX:+PrintCompilatin

CompileThreashold

[Number of interpreted method invocations before (re-)compiling]

jinfo -flag CompileThreshold <process-id>

Now, try with the lower threashold value

-XX:CICompilerCount=6 -XX:CompileThreshold=1000 -XX:+PrintCompilation

Reading the compiler's mind

The -XX:+LogCompilation flag produces a low-level XML file about compiler and runtime decisions

-XX:+UnlockDiagnosticVMOptions -XX:+LogCompilation -XX:+PrintInlining -XX:+PrintCompilation

Print Assembly:

-XX:+UnlockDiagnosticVMOptions -XX:+PrintAssembly

Exception:

"Could not load hsdis-amd64.dll; library not loadable; PrintAssembly is disabled".

Solution:

Download hsdis-1.1.1-win32-amd64.zip file from http://fcml-lib.com/download.html and copy hsdis-amd64.dll file in jdk bin folder.

(or) copy from this example lib folder

Ref : x86_64 Assembly to understand generate code

ex2.2-GC-Test

GC Algorithms Benchmarking:

-Xms1g -Xmx1g -Xloggc:gc_parallel2g.log -XX:+UseParallelGC -XX:+PrintGCDetails -XX:+PrintGCDateStamps

-Xms4g -Xmx4g -Xloggc:gc_parallel4g.log -XX:+UseParallelGC -XX:+PrintGCDetails -XX:+PrintGCDateStamps

-Xms2g -Xmx2g -Xloggc:gc_cms2g.log -XX:+UseConcMarkSweepGC -XX:+PrintGCDetails -XX:+PrintGCDateStamps

-Xms4g -Xmx4g -Xloggc:gc_cms4g.log -XX:+UseConcMarkSweepGC -XX:+PrintGCDetails -XX:+PrintGCDateStamps

-Xms1g -Xmx1g -Xloggc:gc_g1gc2g.log -XX:+UseG1GC -XX:+PrintGCDetails -XX:+PrintGCDateStamps

-Xms4g -Xmx4g -Xloggc:gc_g1gc4g.log -XX:+UseG1GC -XX:+PrintGCDetails -XX:+PrintGCDateStamps

G1 GC With Pause time goal:

-XX:MaxGCPauseMillis=200

ex2.3-GC-heap-sizes

Monitor with VisualVM

Run with :

java -Xmx20m -verbosegc Main
Check the default value for UseAdaptiveSizePolicy
jinfo -flag UseAdaptiveSizePolicy <process-id>

default : -XX:+UseAdaptiveSizePolicy

Set the ratios for Eden and Old generations

jinfo -flag NewRatio <process-id>

-XX:NewRatio=n
[if n is 2 means; old space is two times bigger than eden space]

Note : By setting the NewRatio, UseAdaptiveSizePolicy will be disabled

Try with
java -Xmx20m -XX:NewRatio=1 Main

-XX:SurvivorRatio=n

jinfo -flag SurvivorRatio <process-id>

Run with :

java -Xmx20m -XX:SurvivorRatio=5

-XX:MaxTenuringThreashold=n

jinfo -flag MaxTenuringThreshold <process-id>

ex3.1-HeapDump-MAT

1. Run the demo with -Xmx10m and -Xmx20m

2. Now replace the below code in CustomerManager, onceagain run the demo.

```
public Optional<Customer> getNextCustomer() {
        synchronized(customers)
                { if(customers.size()>0)
                        return Optional.of(customers.remove(0));
                }
                        return Optional.empty(); }
```

Eclipse MAT (Memory analyzer Tool)

File -> Acquire Heap dump -> Select the java process

Getting Started -> Leak Suspects Report

-> Problem Suspect 1-> Select Details

Analyse the "Shortest Paths To the Accumulation Point"

ex3.2 Memory Leak JMX

1. Select ConsumerHeap.java -> Run As

   -> Run configuration

   -> set vm args

      -XX:+HeapDumpOnOutOfMemoryError -Xms32m -Xmx32m

Error will be thrown :

```
java.lang.OutOfMemoryError: Java heap space
Dumping heap to java_pid6664.hprof ...
Heap dump file created [1262470435 bytes in 5.784 secs]
```

Solution :

Analyse the error with "ECLIPSE MEMORY ANALYZER"

ex4.1-StringPool-Performance

The String pool is implemented as a fixed capacity HashMap with each bucket containing a list of strings with the same hashCode.

From command-prompt:

java -XX:+PrintStringTableStatistics

Note : These Strings are loaded by the core java

Try with application:

java -XX:+PrintStringTableStatistics MainTest2

Now, Try with:

java -XX:+PrintStringTableStatistics -XX:StringTableSize=120121

Note : compare the Elapsed Time

Tuning:

java -XX:+PrintFlagsFinal MainTest2

Check the InitialHeapSize & MaxHeapSize

java -XX:+PrintStringTableStatistics -XX:StringTableSize=120121 -XX:MaxHeapSize=600m MainTest2

Note : Out of memory exception comes

Now, try with

java -XX:+PrintStringTableStatistics -XX:StringTableSize=120121

-XX:InitialHeapSize=1g

compare the Elapsed Time

Shortcut Flags

-XX:InitialHeapSize: -Xms

-XX:MaxHeapSize: -Xmx

ex4.2-StringDeduplication

4th execution of StringDeduplication. It took 35ms and edited 2,20,034 strings.

All were classified as "new", which means they have never been analyzed.

In the above example, all strings were deduplicated, which left a total of 8.5MB of memory.

"Total Exec" information is for summary of all invocations

-XX:+UseStringDeduplication -XX:+PrintStringDeduplicationStatistics

Enables the deduplication of Java Strings. This command-line option and feature was introduced in JDK 8u20. String deduplication is disabled by default.

-XX:+PrintStringDeduplicationStatistics

Enables the printing of String deduplication statistics. The default value is disabled. This command-line option can be very helpful when you want to know if enabling deduplication will result in a significant savings in the amount of space

in use by String objects in the Java heap. Hence, enabling this command-line option provides data that justifies whether there may be value in enabling

-XX:+UseStringDeduplication.

-XX:StringDeduplicationAgeThreshold

Sets the String object age threshold when a String object is considered a candidate for deduplication. The default value is 3.

More specifically, a String becomes a candidate for deduplication once a String object has been promoted to a G1 old region, or its age is higher than the deduplication age threshold.

ex5.1 Collections

clear() vs removeAll()

the purpose of clear() and removeAll(Collection c) are different in API, clear() method is meant to reset a Collection by removing all elements, while removeAll(Collection c) only removes elements which are present in supplied collection.

This method is not designed to remove all elements from a Collection.

So, if your intention is to delete all elements from a Collection, then use clear(), while if you want to remove only some elements, which are present in another Collection,

e.g. list of closed orders, then use removeAll() method .

Clear() vs null

Use Clear if we likely want to repopulate that list at some point. If it's a single use list and we are never going to use it again , then consider setting to null.

Note : There is a popular and widely accepted advice in the world of software development that says:

We should always return an empty list instead of null!

There are two considerable advantages:

We eliminate the risk of a null pointer error (i.e. NullReferenceException in C#, NullPointerException in Java, etc.)

We don't have to check for null in client code - your code becomes shorter, more readable and easier to maintain

Contains in a linked list is proportional to the number of entries.

A HashSet will, on average, do the trick in constant time.

ex5.2 soft weak reference

Soft References' Use Cases

Soft references can be used for implementing memory-sensitive caches where memory management is a very important factor.

A cache can, for example, prevent its most recently used entries from being discarded by keeping strong referents to those entries, leaving the remaining entries to be discarded at the discretion of the Garbage Collector.

Weak References' Use Cases

A weakly referenced object is cleared by the Garbage Collector when it's weakly reachable.

Weak references are most often used to implement canonicalizing mappings. A mapping is called canonicalized if it holds only one instance of a particular value. Rather than creating a new object, it looks up the existing one in the mapping and uses it.

ex5.3 phantom reference

Phantom references have two major differences from soft and weak references.

We can't get a referent of a phantom reference. The referent is never accessible directly through the API and this is why we need a reference queue to work with this type of references.

The Garbage Collector adds a phantom reference to a reference queue after the finalize method of its referent is executed. It implies that the instance is still in the memory

Use Cases

There're two common use-cases they are used for.

The first technique is to determine when an object was removed from the memory which helps to schedule memory-sensitive tasks. For example, we can wait for a large object to be removed before loading another one.

The second practice is to avoid using the finalize method and improve the finalization process.

lab-jmc

Customize the overview tab

Select Dashboard - > + -> *eden -> select used

New Graph

Select Historical Data Settings -> +

new chart is created

select the chart

Dashboard -> + -> Select Attributes -> *sur -> PeakUsage -> Used

MBean Browser

Search *eden -> select PeakUsage

Right click used -> visualize-> Chart

System

Memory

Threads

Diagnostic Commands

Select GC.heap_info -> Execute