Hints and Tips for Java on z/OS – Considerations

Topics:

1. Java and LE runtime options
2. JCL REGION size parameter
3. Java Start Up Time
4. 64 Bit Java
5. Shared Classes Group Access Examples

Topic 1 - Java and LE runtime options

These are described in the LE Programming Reference (PDF, 5.24MB). The java executables (java, javac, javah, JZOS Batch Launcher, etc.) are built with a recommended set of runtime options.

You can use the runtime option RPTOPTS(ON) to produce a report that display the options in effect for the java SDK executable.

Topic 2 - JCL REGION size parameter

This needs to be large enough to allow for all of the heap and stack storage requirements, plus LE and Java code, and LE internal control blocks. OutOfMemoryError is not very granular. You may be able to get further useful information by inspecting the Java stack traceback.

Topic 3 - Java Start Up Time

When running hundreds or thousands of small Java batch jobs, the Java start up elapsed time and CPU time become an important performance measurement for many customers. Use of the following Java options makes it possible to reduce the Java startup times for applications that start a new JVM frequently:

- -Xquickstart Java option
- Shared classes and AOT Java options

Java Shared Classes and AOT Examples

You can see below that in this example, the default shared class library size of 16 megabytes was too large for the current BPXPRMxxx setting, which has resulted in an error message. This

problem was caused by BPXPRMxx IPCSHMMPAGES set to 8 MegaBytes, which was less than the defalut shared class library size.

```
$ java -Xshareclasses:listAllCaches
JVMSHRC005I No shared class caches available
Could not create the Java virtual machine.
```

In this next java command, the -Xscmx1m option was specified to use a shared class size of 1 Megabytes. This command was successful. Note that this example requires access to a HelloWorld java program.

```
> java -Xscmx1m -Xshareclasses:name=cache1 HelloWorld
Hello World
```

The following commands demonstrate how to use different shared classes with different sizes.

```
> java -Xshareclasses:listAllCaches
Shared Cache       OS shmid      in use      Last detach time
cache1             1646599       0           Fri Jun 10 10:43:47 2009

Could not create the Java virtual machine.

> java -Xscmx8m -Xshareclasses:name=cache2 HelloWorld
Hello World

> java -Xscmx8m -Xshareclasses:name=cache3 HelloWorld
Hello World
```

Using the ipcs -bom command, the 1 megabyte shared memory segment is displayed for user TESTER

```
> ipcs -bom

IPC status as of Fri Nov 14 10:52:15 2008
Shared Memory:
T             ID     KEY          MODE       OWNER    GROUP   NATTCH
SEGSZPG PGSZ     SEGSZ
m         8196 0x012ed3f0 --rw-------       TCP   DEPTD60          1
1     4K         4096
m         8197 0x022ed3f0 --rw-------       TCP   DEPTD60          1
1     4K         4096
m         8198 0x032ed3f0 --rw-------       TCP   DEPTD60          1
1     4K         4096
m      1646599 0x0100f6f5 --rw-------    TESTER   DEPTD60          0
256     4K      1048576
m         8200 0x0100b3f5 --rw-------    TESTER   DEPTD60          0
2048     4K      8388608
m         8201 0x0100bcf5 --rw-------    TESTER   DEPTD60          0
2048     4K      8388608
```

The following java commands demonstrates options to display shared class information.

```
$ java -Xshareclasses:listAllCaches
```

```
Shared Cache       OS shmid      in use      Last detach time
cache1             1646599       0           Fri Nov 14 10:50:33 2008

cache2             8200          0           Fri Nov 14 10:51:57 2008

cache3             8201          0           Fri Nov 14 10:52:12 2008
Could not create the Java virtual machine.

> java -Xshareclasses:name=cache1,printStats

Current statistics for cache "cache1":

base address       = 0x26F00058
end address        = 0x26FFFFF8
allocation pointer = 0x26FFD418

cache size         = 1048488
free bytes         = 604
ROMClass bytes     = 1037248
Metadata bytes     = 10636
Metadata % used    = 1%

# ROMClasses       = 234
# Classpaths       = 1
# URLs             = 0
# Tokens           = 0
# Stale classes    = 0
% Stale classes    = 0%

Cache is 99% full

Could not create the Java virtual machine.

> java -Xshareclasses:name=cache2,printStats

Current statistics for cache "cache2":


base address       = 0x26F00058
end address        = 0x276FFFF8
allocation pointer = 0x2703F508

cache size         = 8388520
free bytes         = 7067020
ROMClass bytes     = 1307824
Metadata bytes     = 13676
Metadata % used    = 1%

# ROMClasses       = 306
# Classpaths       = 2
# URLs             = 0
# Tokens           = 0
# Stale classes    = 0
% Stale classes    = 0%

Cache is 15% full
```

```
Could not create the Java virtual machine.
```

The following java command demonstrates how to destroy or remove a shared class library. In this example the shared memory segment for the shared class is also removed. It should be noted that the share memory for java shared classes is not removed when the JVM terminates. The following java command with the destroy option must be issued to remove the shared class and its shared memory as shown below.

```
> ipcs -bom

IPC status as of Fri Nov 14 10:52:15 2008
Shared Memory:
T            ID     KEY          MODE         OWNER     GROUP   NATTCH
SEGSZPG PGSZ      SEGSZ
m          8196 0x012ed3f0 --rw-------       TCP   DEPTD60            1
1     4K           4096
m          8197 0x022ed3f0 --rw-------       TCP   DEPTD60            1
1     4K           4096
m          8198 0x032ed3f0 --rw-------       TCP   DEPTD60            1
1     4K           4096
m       1646599 0x0100f6f5 --rw-------    TESTER   DEPTD60            0
256     4K        1048576
m          8200 0x0100b3f5 --rw-------    TESTER   DEPTD60            0
2048     4K         8388608
m          8201 0x0100bcf5 --rw-------    TESTER   DEPTD60            0
2048     4K         8388608

> java -Xshareclasses:name=cache1,destroy
JVMSHRC010I Shared Cache "cache1" is destroyed
Could not create the Java virtual machine.

> ipcs -bom

IPC status as of Fri Nov 14 10:57:31 2008
Shared Memory:
T           ID     KEY          MODE        OWNER     GROUP   NATTCH
SEGSZPG PGSZ      SEGSZ
m        8196 0x012ed3f0 --rw-------       TCP   DEPTD60            1
1      4K           4096
m        8197 0x022ed3f0 --rw-------       TCP   DEPTD60            1
1      4K           4096
m        8198 0x032ed3f0 --rw-------       TCP   DEPTD60            1
1      4K           4096
m        8200 0x0100b3f5 --rw-------    TESTER   DEPTD60            0
2048       4K         8388608
m        8201 0x0100bcf5 --rw-------    TESTER   DEPTD60            0
2048       4K         8388608
```

Topic 4 - 64 Bit Java

Use of 64 bit Java may make it possible to define large Java heap size to avoid out of virtual memory conditions and improve application reliability. Large Java heaps also means that Garbage Collection will occur less frequently, which may improve your applications performance. Use of 64 bit Java Compress References option -Xcompressedrefs and large page option -Xlp in IBM Developer Kit for Java 6 will improve performance of 64 bit Java significantly.

Examples First Time Using 64 bit Java

To be able to run 64 bit Java, the memory limit or memlimit for memory above the bar must be sufficient to run the 64 bit application. Before attempting to use 64 bit Java for the first time on z/OS, check that the memlimit "memory above the 2 Gigabyte bar" is not zero by using the ulimit command as shown below. In this example, the memory above the bar is 1000 megabytes. Your configuration may be different.

```
> ulimit -a
core file         8192b
cpu time          unlimited
data size         unlimited
file size         unlimited
stack size        unlimited
file descriptors  1500
address space     unlimited
memory above bar  1000m
```

The following 64 bit Java command is invoked to display the java version. The following command displays "s390x-64" , which means the 64 Bit Java is being run.

```
> java -version
java version "1.6.0"
Java(TM) SE Runtime Environment (build pmz6460sr5-20090604_01(SR5))
IBM J9 VM (build 2.4, J2RE 1.6.0 IBM J9 2.4 z/OS s390x-64
jvmmz6460sr5-20090519_35743 (JIT enabled, AOT enabled
J9VM  - 20090519_035743_BHdSMr
JIT   - r9_20090518_2017
GC    - 20090417_AA)
JCL   - 20090529_01
```

Next example will use -verbose:gc java command option to display the default Java heap size values. In this example maxHeapSize value is 0x20000000 or 512 megabytes.

```
> java -verbose:gc -version

<initialized>
  <attribute name="gcPolicy" value="-Xgcpolicy:optthruput" />
  <attribute name="maxHeapSize" value="0x20000000" />
```

```
  <attribute name="initialHeapSize" value="0x400000" />
  <attribute name="compressedRefs" value="false" />
  <attribute name="pageSize" value="0x1000" />
  <attribute name="requestedPageSize" value="0x1000" />
</initialized>
```

The next examples demonstrates changing the memlimit for above the bar storage from 1000 megabytes to 200 megabytes using the ulimit -M option. After setting 200 megabyte above the bar storage, the 64 bit java command is invoked with default heap size of 200 megabytes. As expected the java command fails. Note this example requires access to a Java HelloWorld program.

```
> ulimit -M 200

> ulimit -a
core file         8192b
cpu time          unlimited
data size         unlimited
file size         unlimited
stack size        unlimited
file descriptors  1500
address space     unlimited
memory above bar  200m

> java HelloWorld
CEE0814S Insufficient storage was available to extend the stack.
[1] + Done(139) java HelloWorld
  16908772 Segmentation violation /PRIPKC/J6.0_64/bin/java
```

If the first time use of 64 bit Java fails, the first thing to check is ulimit -a memory above the bar storage is sufficient to run a java application.

The next example shows invoking java command for HelloWorld with a smaller maximum heap size value of 100 megabytes to compensate for the 200 megabyte memory above the bar storage. As you can see this HelloWorld example was successful with the smaller Java heap size.

```
> java -Xmx100M HelloWorld
Hello World
```

Topic 5 - Shared Classes Group Access Examples

On Linux®, AIX®, z/OS®, and i5/OS® platforms, if multiple users in the same operating system group are running the same application, use the **groupAccess** sub option, which creates the cache allowing all users in the same primary group to share the same cache. If multiple operating system groups are running the same application, the %g modifier can be added to the cache name, causing each group running the application to get a separate cache.

Note in the example below two shared classes are created. One that is shared with other JVMs in the same group and another that is shared with with JVM using the same user ID.

```
> java -Xscmx8m -Xshareclasses:groupAccess,name=%gcache5 HelloWorld
Hello World

> java -Xscmx8m -Xshareclasses:listAllCaches
Shared Cache       OS shmid      in use      Last detach time
DEPTD60cache5      73739         0           Fri Mar  2 13:59:01 2007

> java -Xscmx8m -Xshareclasses:groupAccess,name=cache5 HelloWorld
Hello World

> java -Xscmx8m -Xshareclasses:listAllCaches
Shared Cache       OS shmid      in use      Last detach time
DEPTD60cache5      73739         0           Fri Mar  2 13:59:01 2007

cache5             204810        0           Fri Mar  2 14:01:40 2007

> id
uid=258(TESTER) gid=0(DEPTD60
```

For Java 8 SR1 -- a mechanism to persist the Shareclasses across IPLs had been added. Find details in Knowledge Center:
https://www.ibm.com/support/knowledgecenter/en/SSYKE2_8.0.0/com.ibm.java.zos.80.doc/diag/appendixes/cmdline/Xshareclasses.html