WebSphere Liberty z/OS

Shared Class Cache

Version Date: September 13, 2018

See "Document Change History" on page 16 for a description of the changes in this version of the document

Beena Hotchandani
Bhotchan@us.ibm.com
IBM WebSphere Performance

WP102766 at
ibm.com/support/techdocs

This page intentionally left blank

Table of Contents

Introduction....4
Basic concepts....4
OpenJ9 Documentation....5
Benefits and Performance Information....6
Benefits of shared class cache....6
Performance of shared class cache....6
Server startup time....6
Shared class cache and AOT....7
Memory footprint savings with shared class cache....8
Effect of shared class cache on throughput....8
Shared class cache and startup time for 100 Liberty servers....9
Operational Details....10
WebSphere Application Server traditional: default cache sizes, names, and location....10
Liberty: default cache sizes, names, and location....10
Specifying a non-default cache name....10
Specifying a non-default cache size....10
Specifying a non-default cache location....11
Displaying shared class cache information....11
Cache size limit....12
Destroying an existing shared class cache....13
Tuning the shared class cache....13
Summary and References....15
Summary....15
References....15
Document Change History....16

Introduction

When a Java Virtual Machine (JVM) first starts up, Java class files have to be read in and processed into JVM internal class representations. These internal representations can then be referenced by Java code executing within the JVM. Once created, the read-only elements of the object can be stored in shared class cache and referenced from there. The overhead of this activity typically occurs during JVM and application startup in what's often referred to as a "warm up"[1].

Imagine the ability to store the created class representations in memory so the *next time* the JVM starts the startup time can be reduced. Imagine further the ability to have *multiple* JVMs access the same in-memory cache, allowing faster startup for *all JVMs* sharing that cache. That's what's behind the "shared class cache" function of the IBM SDK for Java on z/OS[2]. That function came in back in IBM JDK 5, which was almost 10 years ago.

For WebSphere Application Server, the ability to share JVM class cache between regions can take several forms:

- *For WAS traditional z/OS* -- you can share class cache between multiple servant regions, or between servant regions and adjunct regions. You can share between multiple control regions as well, but you can't share between control regions and servant/adjunct regions[3].
- For Liberty z/OS -- you can share class cache between multiple instances of Liberty.

Basic concepts

- Shared class cache (SCC) is created using the JVM option `-Xshareclasses`.
- Every shared class cache has a name associated with it. You can take the default name, which is `sharedcc_%u` (where `%u` is the current user name), or you can specify a name, for example:

  `-Xshareclasses:name=myCache`

  Other JVMs can be directed to use that shared class by coding the same JVM option.
- The default location for shared class cache meta-data is under the `/tmp/javasharedresources` directory, but you can change that with either a `controlDir=` suboption[4], for example:

  `-Xshareclasses:name=myCache,controlDir=/myCachePath/`
- For WebSphere Application Server z/OS traditional, you can code the JVM option by manually editing the `jvm.options` for the server, or by updating the option field for the server process in the Admin Console. For Liberty z/OS, you code the JVM option by creating and editing a `jvm.options` file in the same directory where `server.xml` resides. In either case, the server must be restarted for the JVM option to take effect and the shared class cache to be created.
- The shared class cache lives beyond the life of the JVM. If you stop your JVM and restart it, the previous shared class cache is still available and ready to use.

  You can explicitly delete the cache[5]. An IPL of your system will delete any defined shared class caches[6]. Do not IPL just to clear the cache; using the explicit delete method.

See "Operational Details" on page 10 for more on how to use shared class cache.

1 This is why controlled benchmark tests often involve a period of activity before the formal measuring takes place. That's to insure the measured results are after the classes are created and loaded.
2 Shared class cache is a function of the IBM JDK on multiple platforms, not just z/OS. The focus of this document is z/OS.
3 This is because the control region runs under a separate storage key from servant and adjunct regions.
4 Or `cacheDir=` statement. That results in the same output as `controlDir=` statement.
5 See "Destroying an existing shared class cache" on page 13.
6 The `-Xshareclasses` option has a suboption `snapshotCache`, which will create a snapshot file of an existing non-persistent shared cache. The `restoreFromSnapshot` suboption will restore a new non-persistent shared cache from a snapshot file. See https://www.ibm.com/support/knowledgecenter/en/SSYKE2_8.0.0/com.ibm.java.vm.80.doc/docs/xshareclasses.html for more on these suboptions.

OpenJ9 Documentation

The following link provides a full set of details on shared class cache:

https://www.eclipse.org/openj9/docs/xshareclasses/

Benefits and Performance Information

Benefits of shared class cache

There are several benefits of using shared class cache:

- *Faster start time* -- A fresh shared classes empty cache gets populated by the first JVM JVM as it loads Java classes, so the overhead costs are minimal. Thereafter, the classes are already in memory and partially verified. Therefore, the loading of the classes from a populated cache is faster that loading classes from disk. The benefit is this: from the second time onwards the server startup time is much faster, since many classes are pre-loaded.
- *Less memory used* -- There is also a memory footprint reduction when using shared class cache, as the cache is backed by shared memory segments. In case of WAS z/OS, when more than one servant regions is sharing the same cache the memory savings per servant region is the size of shared class cache[7].

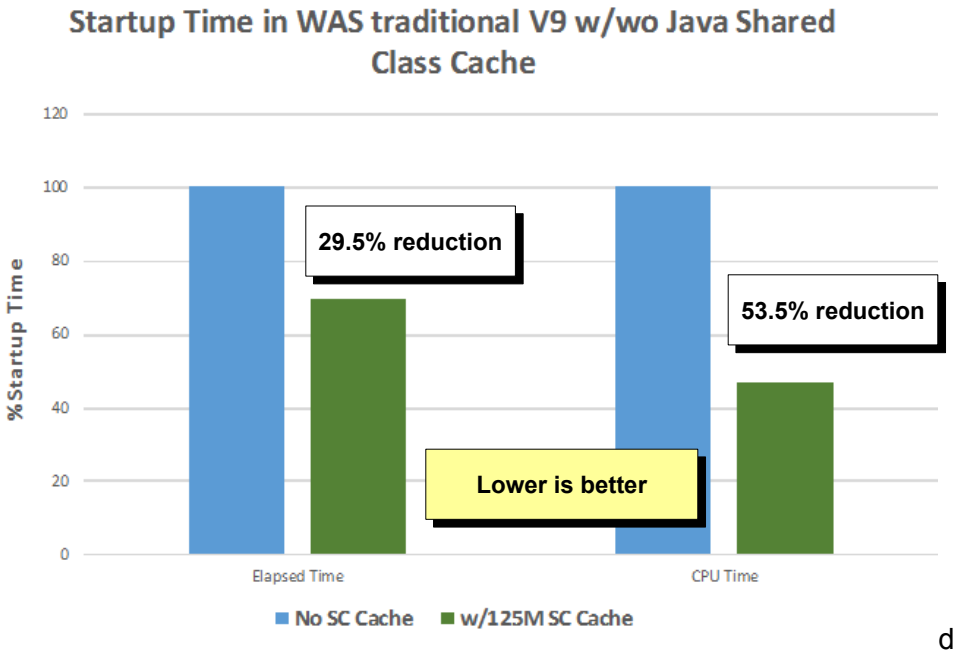
Performance of shared class cache

This feature improves performance since class byte codes need to be loaded only once. Specifically, server startup time can be improved quite a bit. There are some savings in memory footprint as well.

We performed various runs with different options and here are some results.

Server startup time

We ran a comparison of WebSphere Application Server Version 9 traditional[8] using a 125MB shared class cache. The objective was to measure the time -- both elapsed and CPU -- for the server to initialize. The results are summarized in the following graphic:



d

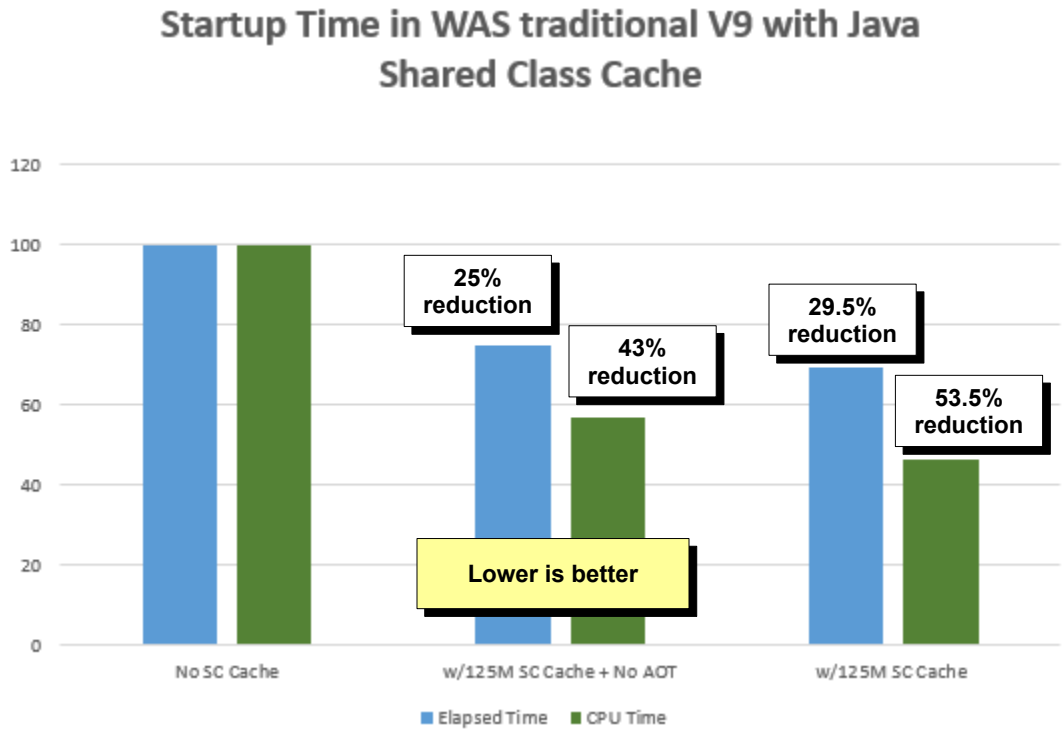
As you can see, with shared class cache the server startup time was 29.5% less elapsed time, and 53.5% less CPU time.

7 To be precise, the memory savings is only the size of the "ROMClass bytes", "Class debug area used bytes" and "Zip cache bytes" in the shared cache. Other data in the cache (the AOT and other metadata) doesn't represent a savings, but rather a tax for using the cache. Also the cache may contain free space, which doesn't contribute to the memory savings.

8 WAS "traditional" is the controller/servant region model, as opposed to Liberty.

Shared class cache and AOT

AOT stands for "Ahead of Time," and it refers to an option of the JVM to compile methods ahead of usage[9]. The combination of shared class cache and AOT further reduces server startup time:

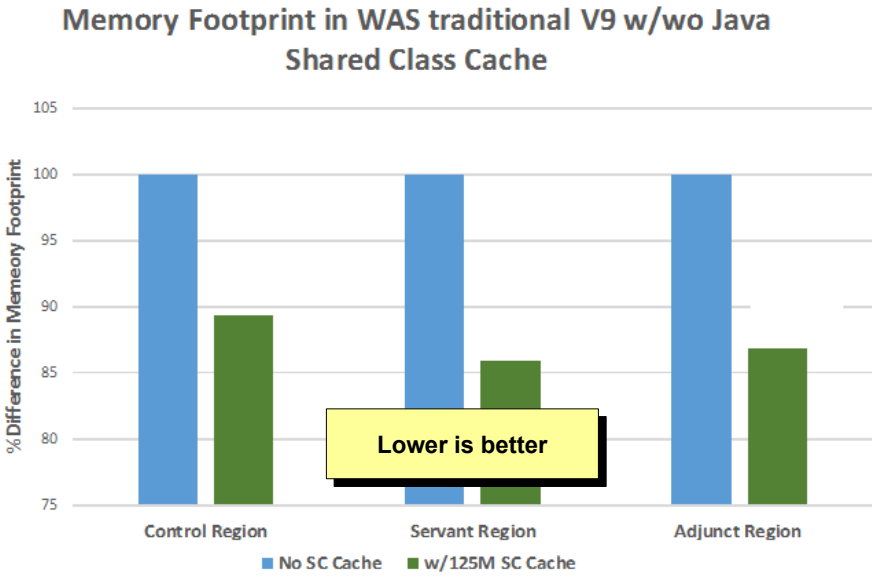


Note: AOT is enabled by default when a JVM is started with `-Xshareclasses`. To disable AOT requires the use of `-Xnoaot`, which is not generally recommended for production usage. The message of this chart is that AOT -- enabled by default -- provides an incremental benefit for server startup time. It *may* involve a very slight degradation of throughput.

9 As opposed to allowing the Just-in-Time (JIT) compiler to decide when to compile the classes, which may be after several iterations of execution. There are pros and cons to using AOT vs. JIT: https://en.wikipedia.org/wiki/Ahead-of-time_compilation

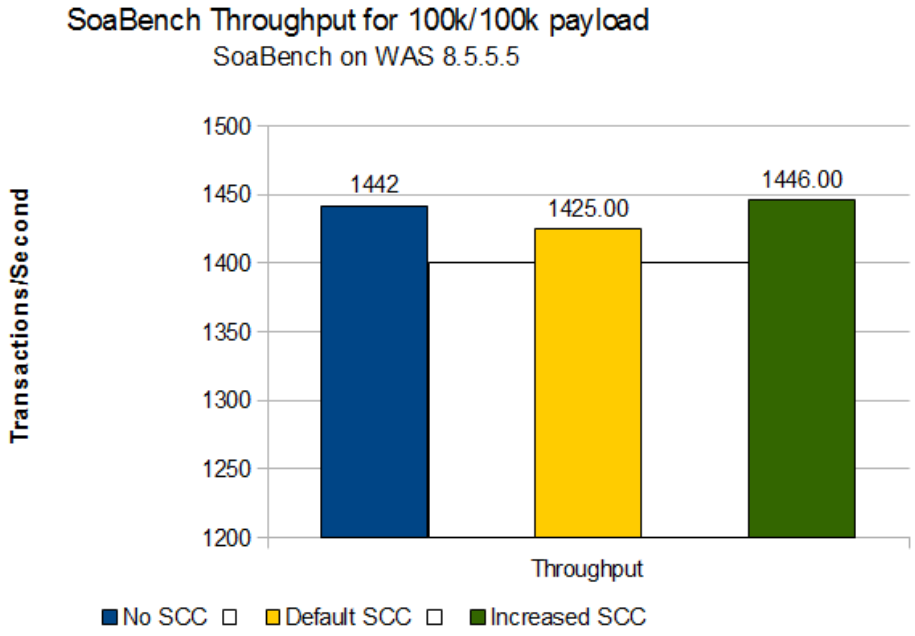
Memory footprint savings with shared class cache

The following graphic illustrates the memory savings with shared class cache for each of the region types of a WebSphere Application Server traditional server[10]:



Effect of shared class cache on throughput

We ran a throughput test using a Service Oriented Architecture (SOA) benchmark workload and compared the throughput with no shared class cache, with the default size of shared class cache, and with a larger shared class cache. The results are shown next:

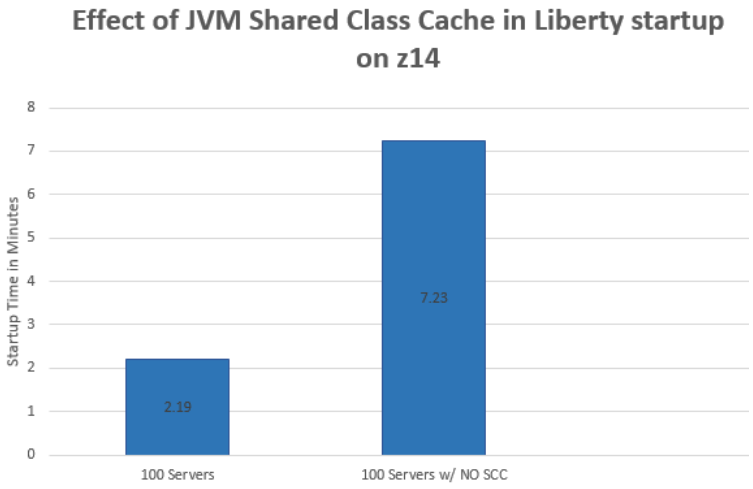


Generally speaking, the shared class cache size should not affect throughput much, if at all. The differences seen here might best be explained by minor fluctuations in the results between runs. That does not mean the size of your shared class cache is meaningless. There is a process for sizing it to your workload. See "Tuning the shared class cache " on page 13 for more on determining the optimal size.

10 In this example, a 125MB shared cache for a controller region, and a separate 125MB shared cache for one servant and one adjunct.

Shared class cache and startup time for 100 Liberty servers

Here's a picture that shows how shared class cache can improve the time it takes for Liberty servers to start:



The column on the left is the time it took to start 100 Liberty servers with shared class cache enabled, and the column on the right is the time it took for the same 100 servers on the same system to start when shared class cache was not enabled: 2.19 minutes vs. 7.23 minutes.

Operational Details

In this section we will offer some of the details around shared class cache.

WebSphere Application Server traditional: default cache sizes, names, and location

The default shared class *size* for WebShere Application Server is shown in the following table:

	WAS z/OS Version 7.x	*WAS z/OS Version 8.x*	*WAS z/OS Version 8.5.5.13 and higher*
Control Region	50MB	75MB	75MB
Servant Region	50MB	75MB	125MB
Adjunct Region	50MB	75MB	125MB

The default class cache *names* are as follows:

	WAS z/OS Version 7.x	*WAS z/OS Version 8.x*
Control Region	webspherev70_%g_control	webspherev80_%g_control
Servant Region	webspherev70_%g_servant	webspherev80_%g_servant
Adjunct Region	webspherev70_%g_adjunct	webspherev80_%g_adjunct
Notes: • Where %g is the UNIX group ID used by the control region, servant region, and adjunct. • The default *size* changed in 8.5.5.13, but the default *names* remained the same.		

If you start a WAS z/OS server and the jvm.options specifies -Xshareclasses with no other suboptions, then the sizes and names shown in the tables above will be used.

The default location for the shared class cache is /tmp/javasharedresources/.

Liberty: default cache sizes, names, and location

For Liberty z/OS, the shared class cache is automatically enabled so no -Xshareclasses in the jvm.options file is needed. The default size is 60MB, which by our testing appears to be sufficient. The location in the file system is:

/<WLP_USER_DIR>/servers/.classCache/javasharedresources

Where WLP_USER_DIR is the directory under which your Liberty server is running.

Specifying a non-default cache name

If you wish to share cache between a servant region and the adjunct region, you would manually change the default name using the name= suboption on -Xshareclasses, for example:

```
-Xshareclasses:name=myCache
```

Then when two JVMs -- for example, a servant region and an adjunct region -- both have jvm.options settings with that value, they would share a class cache[11].

Specifying a non-default cache size

You can set the cache size using -Xscmx. For example, the following two jvm.options statements will create a cache with name myCache, and specify a maximum size of 200MB:

```
-Xshareclasses:name=myCache
-Xscmx200MB
```

11 As noted, the control region is unique in that it's uses a different storage key, and therefore its cache can not be shared except between other controllers.

Specifying a non-default cache location

You can also specify the directory path where you want to create the shared class cache. This is done with the `controlDir=` suboption[12]. For example:

```
-Xshareclasses:name=myCache,controlDir=/mySharedLoc/,groupAccess,nonFatal
```

Note that multiple suboptions are separated with a comma.

FYI: The normal behavior is for the JVM to fail to start if the class data sharing fails for some reason. The nonFatal suboption allows the JVM to start even if class data sharing fails. Normal behavior for the JVM is to refuse to start if class data sharing fails. The JVM will attempt to connect to the cache in read-only mode. If this attempt fails, the JVM will start without class data sharing.

Displaying shared class cache information

If you wish to display basic information about the shared class caches under the *default location*, then issue the following Java command:

```
java -Xshareclasses:listAllCaches
```

That would yield output something like this:

```
>java -Xshareclasses:listAllCaches
Listing all caches in cacheDir /tmp/javasharedresources/
Cache name   level         cache-type   feature  OS shmid OS semid    last detach time
Compatible shared caches
myCache1     Java8 64-bit  persistent   cr                           Wed Jul 11 16:07:11 2018
myCache2     Java8 64-bit  persistent   cr                           In use
 :
myCache10    Java8 64-bit  persistent   cr                           Fri Aug 10 12:39:17 2018
```

If you wished to display basic information about shared class caches under a specific *non-default* location, then issue the `-Xshareclasses` command and name the directory:

```
java -Xshareclasses:controlDir=/mySharedLoc,listAllCaches
```

The `printStats` suboption will display details about a specific shared cache:

```
>java -Xshareclasses:controlDir=/mySharedLoc,name=myCache,printStats
Current statistics for cache "myCache":

Cache created with:
        -Xnolinenumbers                     = false
        BCI Enabled                         = true
        Restrict Classpaths                 = false
        Feature                             = cr

Cache contains only classes with line numbers

base address                        = 0x000003FFE4C57000
end address                         = 0x000003FFE87FE000
allocation pointer                  = 0x000003FFE5BCFBD0

cache size                          = 62913952
softmx bytes                        = 62913952
free bytes                          = 33092912
ROMClass bytes                      = 16223184
AOT bytes                           = 6424756
Reserved space for AOT bytes        = -1
Maximum space for AOT bytes         = 8388608
JIT data bytes                      = 147584
Reserved space for JIT data bytes   = -1
Maximum space for JIT data bytes    = -1
Zip cache bytes                     = 1130480
```

12 Or `cacheDir=` suboption. They both have the same effect.

```
Data bytes                              = 363936
Metadata bytes                          = 529884
Metadata % used                         = 1%
Class debug area size                   = 5001216
Class debug area used bytes             = 2021474
Class debug area % used                 = 40%

# ROMClasses                            = 6243
# AOT Methods                           = 2648
# Classpaths                            = 3
# URLs                                  = 93
# Tokens                                = 0
# Zip caches                            = 21
# Stale classes                         = 0
% Stale classes                         = 0%

Cache is 47% full

Cache is accessible to current user = true
```

Cache size limit

The maximum theoretical limit of cache is 2GB. The size of the cache you can specify depends upon the physical memory and swap space that is available to the system. The cache for sharing is allocated by using IPCS. The following shows some example display commands for cache and associated semaphores:

```
-> ipcs -m
IPC status as of Fri Aug 10 12:15:02 2018
Shared Memory:
T         ID     KEY        MODE       OWNER    GROUP
m    8003604 0x6177dd4d --rw-rw----   ASCRA1   WSCFG1
m    8003605 0x6177db4d --rw-rw----    ASSR1   WSCFG1
m    8003606 0x6177cb4d --rw-rw----     OMVS   WSCFG1
```

Shows three caches created after WAS traditional startup

```
-> ipcs -mx
IPC status as of Fri Aug 10 12:15:07 2018
Shared Memory:
T     KEY       OWNER    GROUP   SEGSZPG PGSZ       SEGSZ      ATPID ATADDR  INFO
m 0x6172454d     OMVS WASADMIN        16  1M     16777216

m 0x6177dd4d   ASCRA1   WSCFG1       125  1M    131072000       1365 0x000002000c800000
m 0x6177db4d    ASSR1   WSCFG1       125  1M    131072000       1320 0x0000020004b00000
m 0x6177cb4d     OMVS   WSCFG1        75  1M     78643200   33555635 0x0000020000000000
```

Shows the pid (example 1365 for the Adjunct Region) associated with the cache.

```
-> ipcs -sx
IPC status as of Fri Aug 10 12:20:37 2018
Semaphores:
T     KEY       OWNER    GROUP TERMA CNADJ SNCNT SZCNT  INFO
s 0x8177dc4d   ASCRA1   WSCFG1     0     0     0     0     L
s 0x8177da4d    ASSR1   WSCFG1     0     0     0     0     L
s 0x8177ca4d     OMVS   WSCFG1     0     0     0     0     L
```

Shows the associated semaphores of the shared class cache

The shared page size of z/OS UNIX Systems Services (USS) is fixed at 4K pages in 31-bit mode and 1MB in 64-bit mode.

The shared class cache will persist beyond the life of any JVM connected to it. To remove a shared class cache, either use the `destroy` suboption of `-Xshareclasses`, or re-IPL the z/OS system.

Destroying an existing shared class cache

You can explicitly delete a cache with the `destroy` suboption. For example:

```
java -Xshareclasses:name=myCache,controlDir=/myCachePath/,destroy
```

Note: In this case that's a Java command entered in a UNIX shell environment, not coded as a JVM option for an application server.

Or you can use the IPCS command, which tends to be a more reliable method. The **`ipcs -mx`** command will show the PID associated with the server's shared memory:

```
/<WLP_USER_DIR>/servers/.classCache/javasharedresources > ipcs -mx
  IPC status as of Tue Aug 21 10:39:00 2018
  Shared Memory:
  T     KEY       OWNER    GROUP   SEGSZPG PGSZ    SEGSZ    ATPID      ATADDR  INFO
  m 0x611b780f   SYSTEM WASADMIN        60  1M             62914560
```

The **`ipcs -m`** command will provide the ID of shared class cache:

```
/<WLP_USER_DIR>/servers/.classCache/javasharedresources > ipcs -m
  IPC status as of Tue Aug 21 10:40:20 2018
  Shared Memory:
  T         ID     KEY        MODE       OWNER    GROUP
  m       8196 0x611b780f --rw-------   SYSTEM WASADMIN
```

The **`ipcrm -m <id>`** command will delete the shared cache from memory:

```
/<WLP_USER_DIR>/servers/.classCache/javasharedresources > ipcrm -m 8196
```

What remains are the semaphores, which you can query with the **`ipcs -s`** command:

```
/<WLP_USER_DIR>/servers/.classCache/javasharedresources > ipcs -s
  IPC status as of Tue Aug 21 10:42:16 2018
  Semaphores:
  T         ID     KEY        MODE       OWNER    GROUP
  s       4100 0x811b770f --ra-------   SYSTEM WASADMIN
```

And delete with the **`ipcrm -m <id>`** command:

```
/<WLP_USER_DIR>/servers/.classCache/javasharedresources > ipcrm -s 4100
```

An IPL of the z/OS system will result in any defined shared class caches being removed. If all you wish to do is clear shared class cache, we do not recommend this approach.

Tuning the shared class cache

In general, you should seek to maximize the number of shared classes, which implies tuning the size of the shared class cache to share the maximum number of classes.

Note: As a reminder, for WAS z/OS traditional, you can't share classes between a Control Region (CR) and a Servant Region (SR) or an Adjunct Region (AR). That's because of the CR uses a different storage key from the SR or AR. You can share between CRs, and you can share between SRs and ARs. But you can't share between CR and SR/AR.

The shared cache size can only be set when creating a new cache. That means if you need to resize the cache, it needs to destroyed and re-created. When tuning a cache size, the most important thing is that you run a typical workload when performing the sizing. Using a non-typical workload when sizing may result in a poorly tuned size when more typical work is run later.

The following illustrates the "iterative approach" to tune the shared class cache size.

Note: The commands that follow assume the shared class cache is in the default location. You can use a non-default location. That involves using the `controlDir=` suboption to specify the location.

- First, check to see if shared class caches exist, and what names they go by:

  ```
  java -Xshareclasses:listAllCaches
  ```

  If there are no shared class caches, go back to the `jvm.options` files of the servers and set the `-Xshareclasses` option, then restart the server.

- Then, use the `printStats` suboption to display information about the shared class cache, for example:

  ```
  java -Xshareclasses:name=webspherev70_%g,printStats
  ```

  That will yield a listing of the statistics for the shared cache like we showed on page 11. Look for the "Cache is <value>% full" statement at the end of the listing.

- If the cache is 100% full, then you will need to destroy the cache before setting a larger value in the server `jvm.options` file. For example:

  ```
  java -Xshareclasses:name= webspherev70_%g,destroy
  ```

- Next, enable shared classes with a larger cache. We recommend increasing the size by 50%. Do this with a `-Xscmx` option in the `jvm.options` file. See "Specifying a non-default cache size" on page 10 for an example. Stop and restart the server to pick up the new size size.

- Exercise a *typical workload* so the Java class files get loaded into the shared class cache. Be sure to exercise the workload for a long enough period of time so all the typically used classes get loaded (this is the "warm up" period we mentioned earlier).

- Check the cache size again using the `printStats` suboption. If the cache size is still 100% full, then *destroy* the cache, *increase* the size by 50% more, *restart* the servers and check again.

  If the cache size is *less* than 100%, then the optimal size is one that achieves a cache utilization in the 95% to 99% range. You can calculate that by subtracting "free space" from the "cache size" values provided by the `printStats` subption.

- Iterate on these steps until you achieve a shared class cache size that achieves an optimal utilization of the cache for the workload you are exercising.

Another approach is to size the cache to some value you expect to be much *more* than is needed for your workload. Then run your workload, and use the `printStats` suboption to display information about the shared class cache. Compare the "cache size" value to the "free bytes" area to come up with the bytes used. For example:

```
cache size                              = 209715200
free bytes                              = 93092912
```

Would suggest that your used cache was 209715200 - 93092912 = 116622288. The cache should therefore be sized to something where 95% to 99% utilization is achieved.

Finally, another approach is to use the soft limit feature by setting a `-XX:SharedCacheHardLimit`, and run with the shared cache verbose option, it tells you how much data it *tried* to store into the cache but could not. This gives a hint at how much bigger the shared cache needs to be, for example:

```
JVMSHRC818I Total unstored bytes due to the setting of shared cache soft max
is 2195456. Unstored AOT bytes due to the setting of -Xscmaxaot is 0.
Unstored JIT bytes due to the setting of -Xscmaxjitdata is 0.
```

In this example, the cache size used resulted in 2195456 bytes not being able to be cached. Therefore, the cache size should be set to a value that allows those 2195456 bytes, plus a buffer of approximately 5% more cache size. See:

https://www.ibm.com/support/knowledgecenter/en/SSYKE2_8.0.0/com.ibm.java.vm.80.doc/docs/xxsharedcachehardlimit.html

Summary and References

Summary

This paper provided some ideas on shared class cache performance in a z/OS environment, and how to tune it to give best results. Making shared class cache too big will not help in performance but might hurt depending on your configuration, size and usage of memory.

References

There are articles, white papers and other documentation on shared class cache which gives detail information on the subject. Some of the references are:

https://www.eclipse.org/openj9/docs/xshareclasses/

https://www.ibm.com/developerworks/library/j-class-sharing-openj9/

Document Change History

Check the date in the footer of the document for the version of the document.

September 11, 2018	Original document at time of Techdoc creation
September 13, 2018	Updated with assigned WP102766 Techdoc number, plus made a few minor corrections.

End of WP102766