How to fix a collector Class Cache Explosion

Information:

Environment
Affected Versions: all

Description

Introduction

The collector is a crucial component of a AppMon installation and can under circumstances get into bad shape.

If you can see the following messages in the collector log (since dT 5.5)

```
INFO [ClassCacheWritingThread] Class cache size on disk:
35,151,172,526 bytes (33522.77M)
INFO [ClassCacheWritingThread] Inheritance map files size on disk:
36,990,428 bytes (35.28M)
```

you should consider to act, since these are only written when exceeding a specific threshold. To be more exact, it depends on how comprehensive all the byte-code of all the applications that are connected to this collector is. So the size might not be concerning but the tendency how fast it is growing after everything was instrumented once.

The reason why the class cache can constantly grow are more and more popular software frameworks that are in use in the monitored applications which are creating classes dynamically. Those classes are often just created and used for single requests and dropped afterwards, but the collector stores it in the class cache.

AppMon 5.5 already contains a cleanup mechanism that is dropping classes out of the class-cache that have not been in use for at least a week, but the health could also be affected by a huge inheritance map (IMAP cleanup targeted for dT 5.6).

Affects

Only Java applications, since AppMon is currently not able to instrument dynamic .NET assemblies.

Solution

The only solution we currently have is to exclude those dynamically created classes on the agent side so they are not sent to the collector anymore. After implementing those exclusions the class cache needs to be dropped to get rid of the old occurances.

Determine classes

With IMAP
pro: small set of files to copy
con: same named classes from same package or namespace only show up once

1. Copy the IMAP files (*.imap) from <DT_HOME>/collector/cache to a different location
2. Download and execute:

- ClassCacheImapDump.jar (dT class cache < 5.5)
- ClassCacheImapDump55.jar (dT 5.5 class cache)
- ClassCacheImapDump60.jar (dT 5.6, 6.0, 6.1, 6.2, 6.3, 6.5)

```
java –jar ClassCacheImapDump.jar ./<imap_copy_folder>/
```

You could redirect the output to a textfile for further analysis. BTW, a JVM is needed to execute the dump tool.

1. Find similar named classes like

```
com.foobar.package.DynamicallyCreatedClass$$EnhancerByCGLIB$$
c5477de5
com.foobar.package.DynamicallyCreatedClass$$EnhancerByCGLIB$$
a3275ce2
```

To do this efficiently, it's recommended to use a tool that is able to sort the lines. (eg. Notepad++ with TextFX plugin). Alternatively you could use the attached "ClassCacheInstanceUtility.jar" tool, which will aggregate dynamic class instance entries based on most common dynamic class patterns and sort them based on their frequency in the class cache. The usage for the tool is:

```
java -jar ClassCacheInstanceUtility.jar <IMAP/classes dump
output file> <OPTIONAL aggregated entry threshold>
```

The optional aggregated entry threshold argument determines what is the minimal number of aggregated instances of a pattern to display in the output, default is 50. The output of the tool is in the format: "<dynamic class patern root> : <number of occurances>"

With class cache
Alternatively you can execute the dump tool also on the `classes` subfolder (usually huge at that time -> copy maybe not easily possible) - same procedure as for the IMAP files. In case the by the monitored applications used frameworks for dynamically created classes is creating same named classes, this way is required to find out which one those are.
pro: all class cache content visible
con: big amount of data to copy

NB: on version 6.3 and higher, classes are stored in a folder called "set1" and "set2" instead of the "classes" folder. The dump tool does not recognize the "set1" and "set2" folders when specified on the command line. To proceed, the files from "set1" or "set2" must be moved into a folder called "classes".

Exclude classes

Based on the found similar named classes create exclusion rules:

- Restrictive

  ```
  -agentpath:....,exclude=starts:com/foobar/package/Dynamically
  CreatedClass$$EnhancerByCGLIB$$
  ```

  All package '.' separators need to be replaced by '/'.
- More comprehensive

  ```
  -agentpath:....,exclude=contains:$$EnhancerByCGLIB$$
  ```

- Use ';' to combine 2 or more rules. Eg.

  ```
  -agentpath:....,exclude=starts:com/dynatrace/diagnostics/foob
  ar;contains:Proxy
  ```

- Allowed are "starts", "ends" and "contains".

As can be seen on the above examples, it needs to be added to the -agentpath command line parameter for the application JVMs.

For details on excluding classes, please see the other KB articles

- How to excluding classes as an agent parameter
- Excluding classes as an agent parameter

Automatic clean-up

Automatic clean-up may be a sufficient remedy for growing class caches and an alternative to exclusions as described above. This depends on the rate of dynamically generated classes which get written to the class cache. For very fast growing class caches (caused by applications that generate a high number of classes dynamically), exclusions may be required. For moderately to slow growing class caches the clean-up mechanism should be sufficient.

AppMon 5.5 collectors (and higher versions) will automatically clean up the class cache (DT_HOME/collector/cache classes). This task is, with the out-of-the-box settings, maximum running every day and cleaning up classes older than 7 days. How often the clean-up is really running depends on multiple conditions. All of them need to be matched, otherwise the cleanup will not start.

Condition	Collector setting	Default value
Hour of day (check is done per minute, "cleanup start" is possible for the whole hour).	`com.dynatrace.diagnostics.classcache.cleanupTime`	3 (=3am, possible 0-23)
Minimum wait time after last cleanup.	`com.dynatrace.diagnostics.classcache.cleanupMinimumWaitTime`	12 (h)
Oldest item in the class cache needs to be older than the configured age setting. Additionally, a check is performed during clean-up if the class is still loaded by a connected agent - otherwise it is kept in the cache.	`com.dynatrace.diagnostics.classcache.cleanupItemAge`	168 (h, =7 days)
Growth rate since last cleanup.	`com.dynatrace.diagnostics.classcache.minGrowthBeforeCleanup`	500 (MB)

To adapt these settings, please see Setting AppMon Collector Debug Flags.

From AppMon 6.3 on, the partition where the class cache is located needs to have at least the same amount of space free than the current class cache consumes. Otherwise the cleanup will not start.

In addition, dynaTrace 5.6 collectors (and higher versions) will automatically clean up the inheritance maps (DT_HOME/collector/cache/*.imap). The clean up job is run every 7 days. Please note that this feature is disabled by default in App Mon 5.6 and can be enabled by adding the setting

```
-Dcom.dynatrace.diagnostics.imap.cleanupItemAge=604800000
```

to dtcollector.ini (followed by a collector restart). 604800000 is measured in milliseconds and is the maximum age of items in the class cache that will be kept during the clean-up job. A value of 604800000 equals 7 days and is the recommended setting. This will be the default with AppMon 6 and higher.