Conversion Tools - Bug #6642

improve parsing memory footprint

07/27/2022 11:39 AM - Constantin Asofiei

Status:	Review	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Constantin Asofiei	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**case_num:**	
vendor_id:	GCD	**version:**	

Description

Related issues:

Related to Conversion Tools - Bug #6627: improve conversion memory footprint ...	**Review**

History

#2 - 07/27/2022 11:39 AM - Constantin Asofiei

- Related to Bug #6627: improve conversion memory footprint for AstKey persistent maps added

#3 - 07/27/2022 11:59 AM - Constantin Asofiei

In #6627-4 it was found that the parsing phase keeps a lot of state in the heap. This required for a large customer application to use a 16GB of heap for parsing, which remained occupied for the full conversion, thus full conversion needed 22GB of heap.

Further analysis of the Java heap during parsing showed that almost all memory is consumed by ASTs being kept around:

- ClassDefinition.schemaDict.scopes - this keeps all the temp-table and permanent database ASTs (although the same ASTs are kept for permanent database)
- MemberData.var.def - the entire legacy class AST

The correct approach of reducing the heap is to reduce the parsing heap - otherwise the heap will still increase proportional with the number of converted .cls files. The approach to reduce it is:

- once a .cls file finished parsing:
  - cleanup the ClassDefinition.schemaDict.scopes of all private temp-tables and permanent tables (what ClassDefinition.cleanupObjectResources meant to do, but it was never called).
  - MemberData.var.def is set to a copy of itself, to detach it from the class AST
- once the parsing phase completes
  - release all ClassDefinition.schemaDict references by setting the schemaDict field to null.
  - release all MemberData.var references by setting the var field to null.

The result of this is:

- during parsing, from a top of ~14GB of heap, now is ~8GB. With these changes, the parsing heap now scales with the depth of the prescan parse classes (as ClassDefinition.schemaDict and MemberData.var.def has state released when a .cls finishes parsing completely).
- after parsing, as ClassDefinition.schemaDict and MemberData.var are released, the heap usage ends up to ~1-2GB, before post-parse-fixups starts.

The conclusion: for a large application, a full conversion should be possible with ~10GB of heap.

#4 - 07/27/2022 01:09 PM - Constantin Asofiei

- Status changed from New to Review

- % Done changed from 0 to 100

- Assignee set to Constantin Asofiei

The changes are in 6129a/14249. Please review.

This will be committed to 6129a.3821c.13813, too, once I'm finished testing the changes with some large apps.

The other issue with large apps: Java compile (javac) can still require 16-22GB of heap, but this can be reduced by compiling the Java source code in smaller batches.

#5 - 07/29/2022 06:51 AM - Constantin Asofiei

I've fixed some issues in 6129a/14249 in 12450, the customer's app now completes full conversion with 10g of heap.

There are also some misc fixes, full list is this:

- 4GL allows an empty body getter/setter to be explicitly defined at the interface property; FWD needs to drop the empty body for this getter/setter.
- Fixed conversion of property setter/getter name when is passed as an OUTPUT/INPUT-OUTPUT argument.
- More improvements to 'read_jtype' to not show "missing 'jtype' annotation" in 'read_jtype'" for cases where this annotation is not expected.
- Fixed 6129a/14249 - 'ClassDefinition.schemaDict' must preserve the global namespace, to allow buffer definitions for permanent tables to be resolved from sub-classes.