Base Language - Feature #6826

reduce the BDT usage in FWD runtime

10/06/2022 03:43 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	**version:**	
vendor_id:	GCD		

Description

Related issues:

Related to Base Language - Feature #6817: replace INSTANTIATING-PROCEDURE and...	**Test**
Related to Base Language - Feature #6827: reduce the memory footprint of Wrap...	**WIP**
Related to Base Language - Bug #8510: replace all 'new logical' in FWD runtim...	**New**

History

#1 - 10/06/2022 03:48 AM - Constantin Asofiei

There are resources like StaticTempTable where the state is kept as BDT, and converted APIs are being called to read this state, instead of accessing the state directly. BDT instances are heavy, and FWD runtime should use as few as possible, to reduce both BDT allocation count and CPU time.

This task is meant to:

- analyze BDT usage in FWD - class members being kept for runtime purposes only need to be changed to their Java equivalent. Any complex scenarios, like maybe datetime/tz, need to be discussed.
- converted API calls where for example a character attribute is being read by FWD runtime, just to get its Java value - the Java value needs to be read directly.
- do not pass handle instances around the FWD runtime - pass the WrappedResource directly. ExternalProgramWrapper instances will be covered in #6817

Some fixes will be in 6129a/3821c.

#2 - 10/06/2022 03:49 AM - Constantin Asofiei

- Related to Feature #6817: replace INSTANTIATING-PROCEDURE and other handle<ExternalProgram> with the actual referrent added

#3 - 10/06/2022 03:53 AM - Constantin Asofiei

- Related to Feature #6827: reduce the memory footprint of WrappedResource implementations: replace BDTs with Java types added

#4 - 10/06/2022 03:55 AM - Constantin Asofiei

The WrappedResource refactor will be done in #6827.

#5 - 10/06/2022 04:27 AM - Constantin Asofiei

Some changes are in 3821c/14272. Ovidiu, please review.

#6 - 10/06/2022 10:12 AM - Constantin Asofiei

Another idea is to replace new unknown (conversion and runtime) with a singleton instance.

#7 - 10/07/2022 08:48 PM - Ovidiu Maxiniuc

Constantin Asofiei wrote:

> Some changes are in 3821c/14272. Ovidiu, please review.

Global notes:

- the replacement of BDT (for getUniqueID and ADM-DATA attribute) is definitively a good thing: not only they take up less space but the operations are faster. I found no issues with the new code.
- some time ago, I also wrote some code for nano-timing pieces of code related to (new) persistence. I did not commit them (or rolled them back) because I was afraid of a possible performance penalty which come with the wrapping the productive code as lambdas. There is also extra code for handling non-final variables constraint of Java. I wonder whether we can find a solution to allow direct call to Impl methods in normal code, and activate the JMX wrappers only when doing profiling. Maybe as an aspect post-processing?
- replacing tableHandle().unwrapTempTable() with getParentTable() is very nice. Maybe we do something similar for default buffer? I wonder whether we can cache the result, though.

AnnotatedAst:

- private String getPath(int max): it does not look like an optimization to me. All characters are twice reverse d. If the original problem was the ArrayList iterator, then it cam be replaced with a (reversed) for and each element from list can be accessed directly using get(index) (it returns elementData[index]);
- private void putAnnotationImpl(String key, Object annotation): doesn't RulesTracing expect a lowercase key ?

#8 - 10/10/2022 05:29 AM - Constantin Asofiei

Ovidiu Maxiniuc wrote:

> AnnotatedAst:
>
> - private String getPath(int max): it does not look like an optimization to me. All characters are twice reverse d. If the original problem was the ArrayList iterator, then it cam be replaced with a (reversed) for and each element from list can be accessed directly using get(index) (it returns elementData[index]);

The changes want to reduce the number of object allocations - String instances in this case. I've tested with and without, and the change is marginally better than the previous one. I'll continue keeping an eye on this.

- private void putAnnotationImpl(String key, Object annotation): doesn't RulesTracing expect a lowercase key ?

Fixed in 3821c/14281

3821c/14281 contains other performance improvements:

- Performance improvements - avoid the ProcedureData lookup, by keeping a parallel stack of this data for THIS-PROCEDURE

#9 - 12/13/2022 09:45 AM - Greg Shah

- Assignee set to Stanislav Lomany

#10 - 01/10/2023 10:50 AM - Constantin Asofiei

Stanislav, did you start work on this?

#11 - 01/10/2023 10:50 AM - Stanislav Lomany

No.

#12 - 01/10/2023 10:51 AM - Constantin Asofiei

- Assignee changed from Stanislav Lomany to Constantin Asofiei

Stanislav Lomany wrote:

> No.

OK, I'm taking it, as is easier to look in the profiler and see what can be reduced.

#13 - 01/16/2023 03:53 PM - Constantin Asofiei

6129b/14366 contains a set of changes related to this, mainly from what profiling a large customer app showed.

#14 - 01/20/2023 09:22 AM - Constantin Asofiei

- Status changed from New to Review

- % Done changed from 0 to 100

Profiling and manual/automated code review was used to identify BDT field and local variable definitions. These were replaced with their Java counterpart. This task can be closed, as in performance terms, the parts which showed up in profiling were fixed, and the code review covers a large part of possible BDT usage.

#15 - 02/15/2023 04:46 AM - Constantin Asofiei

7826a/14811 fixes other cases to reduce BDT usage from within FWD runtime.

#16 - 03/24/2024 05:17 AM - Constantin Asofiei

- Related to Bug #8510: replace all 'new logical' in FWD runtime with logical.of or logical.UNKNOWN added

#17 - 03/25/2024 08:37 AM - Ovidiu Maxiniuc

There are many cases where a BDT is used as a short-lived constant value. The most of case are logical and interger s.

Can we implement a cache, similar to Integer? They have a small cache of the most used values (up to 256 IIRC) and return these instead of creating these short-lived constant values. The unknown values is just an example of such a constant. We must ensure they are immutable, or better said, use them only when they are immediately discarded (no assign or other mutation allowed).