Runtime Infrastructure - Bug #12168

Reduce FWD server heap: read DMO @Property without reflection, share proxy Method objects

11/13/2026 07:28 AM - Artur Şcolnic

Status:	Test	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Artur Şcolnic	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**reviewer:**	Constantin Asofiei
vendor_id:	GCD	**production:**	No
case_num:		**env_name:**	
version_reported:		**topics:**	
version_resolved:			

Description

Related issues:

Related to Base Language - Feature #6407: name_map.xml improvements	**Test**
Related to Runtime Infrastructure - Bug #12161: Live memory allocation optimi...	**Review**
Related to Base Language - Feature #12222: Remove name_map.xml by using annot...	**Review**

History

#1 - 11/13/2026 07:30 AM - Artur Şcolnic

- Status changed from New to WIP

- Assignee set to Artur Şcolnic

Motivation

Heap dumps of three FWD servers (different appps) after 15 minutes of testing showed two fixed-cost items common to all of them:

- *Property annotation objects (100–190 MB).* The JDK parses each DMO field's @Property@ annotation into an object that stores all 38 members, defaults included (~2.3 KB per field), and caches it for the life of the class. FWD reads it once in DmoMeta.getPropertiesList() and copies the values into orm.Property, but the cached object can never be freed, because the Method objects held by proxies and orm.Property keep it reachable.
- **Duplicated Method objects (23–74 MB).** ProxyFactory.gatherMethods() called getDeclaredMethods() / getMethods() for every generated proxy class. Each call returns fresh copies, so each proxy kept its own copy of ~600 BufferImpl methods (530k–842k copies per server).

Changes

- New DmoAnnotationReader reads @Property values directly from the class file with ASM, so the JDK never creates or caches the annotation objects. If no class file is available, it falls back to reflection.
- Dynamic temp-tables only exist in memory, so DynamicTablesHelper and SchemaCheck pass the generated class bytes through a new DmoMetadataManager.registerDmo(..., Map<String, byte[]>) overload.
- ProxyFactory caches each class's method arrays (ClassValue), so all proxy classes share the same Method instances.

Result (multi tenant app, same test, before/after)

- Live heap: 1,375 MB → 1,113 MB (**–263 MB, –19%**).
- @Property annotation objects: 79,573 → 0 (–187 MB).
- Method objects: 1.74 M → 1.04 M (–62 MB); BufferImpl copies: 530,901 → 401.

On a large production server the saving stays roughly the same in absolute terms: a fixed 300-600Mb rather than a percentage of the heap.

#2 - 11/13/2026 07:42 AM - Artur Școlnic

- Status changed from WIP to Review

- % Done changed from 0 to 100

- reviewer Constantin Asofiei added

Constantin, please review 12168a.

#3 - 11/13/2026 08:06 AM - Constantin Asofiei

Please commit DmoAnnotationReader

Can you check SourceNameMapper, what is the heap required for reading all the legacy annotations from the converted source code?

#4 - 11/13/2026 08:19 AM - Artur Școlnic

Constantin Asofiei wrote:

> Please commit DmoAnnotationReader

Done, sorry I missed it...

> Can you check SourceNameMapper, what is the heap required for reading all the legacy annotations from the converted source code?

Reading the legacy annotations through reflection costs about 2.1 KB per @LegacySignature and 0.85 KB per @LegacyParameter.
In the application dumps I used this amounts to 30-40Mb, worth fixing.

#5 - 11/13/2026 08:24 AM - Constantin Asofiei

Artur Școlnic wrote:

> Reading the legacy annotations through reflection costs about 2.1 KB per @LegacySignature and 0.85 KB per @LegacyParameter.
> In the application dumps I used this amounts to 30-40Mb, worth fixing.

How many java.lang.Class objects were there loaded? We have apps with 10k's of converted files.

I think is worth it, this is 'read and forget' code which is not useful one processed, so no need to keep it cached in the JVM. Consider making DmoAnnotationReader more generic to handle any kind of annotation.

#6 - 11/13/2026 08:28 AM - Artur Școlnic

- Status changed from Review to WIP

Constantin Asofiei wrote:

> How many java.lang.Class objects were there loaded?

70-90k, depending on the app.

> I think is worth it, this is 'read and forget' code which is not useful one processed, so no need to keep it cached in the JVM. Consider making DmoAnnotationReader more generic to handle any kind of annotation.

Will do.

#7 - 11/13/2026 09:28 AM - Greg Shah

- Related to Feature #6407: name_map.xml improvements added

#8 - 11/13/2026 09:42 AM - Artur Școlnic

- Status changed from WIP to Review

Constantin, please review rev 12168a/16851.

#9 - 11/13/2026 09:50 AM - Șerban Bursuc

- Related to Bug #12161: Live memory allocation optimizations. added

#10 - 11/13/2026 10:16 AM - Constantin Asofiei

- Status changed from Review to Internal Test

Artur Școlnic wrote:

> Constantin, please review rev 12168a/16851.

Thanks, lets get this tested.

#11 - 11/16/2026 03:26 AM - Artur Școlnic

Multi tenant app, large gui app unit tests, chui regression and etf testing passed. Can we merge?

#12 - 11/16/2026 03:28 AM - Constantin Asofiei

- Status changed from Internal Test to Merge Pending

Yes, go ahead now.

#13 - 11/16/2026 03:34 AM - Artur Școlnic

- Status changed from Merge Pending to Test

12168a was merged to trunk/16850.

#14 - 11/19/2026 02:46 AM - Dănuț Filimon

- Related to Feature #12222: Remove name_map.xml by using annotations added