Bugs - Bug #2249

PermGen leak due to dynamic conversion at runtime

02/27/2014 11:59 AM - Eric Faulhaber

Status:	Closed	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Eric Faulhaber	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:	Cleanup and Stablization for Server Features		
billable:	No	**case_num:**	
vendor_id:	GCD		

Description

Related issues:

Related to Database - Feature #2274: improve performance of new database feat...	**Closed**

History

#1 - 02/27/2014 12:51 PM - Eric Faulhaber

I've noticed as I've been debugging a TRPL error during runtime conversion, that the class names for TRPL expressions keep advancing. The convention for compiled TRPL expressions is CE (for "Compiled Expression") followed by a number suffix, which advances sequentially. In trying to track down an NPE in a specific init-rule in brew.xml, I noticed the class name for the same compiled expression kept changing, each time with a higher numeric suffix.

Currently, we cache each compiled expression instance, such that if we encounter the same expression multiple times, we fetch the cached instance of the compiled version and use that instead of taking the performance and memory hit of compiling and instantiating it again. Each instance is cached at a certain scope, which can be a rule set, a named function, or the pattern engine itself. The cache is a two-level cache. The outer level is a weak hash map whose keys are the scope objects (rule sets, named functions, pattern engine). For each scope, there is a map at the inner level of the cache, which has infix expression strings as its keys and the compiled expression instances as its values.

We cache the instances of the compiled expressions rather than the classes, because the initialization of these objects uses information (such as the libraries they reference and the variables that are in scope at the time), which is only available and known at the time we are parsing/compiling the expression.

This approach works OK for the most part (though it must also leak PermGen memory, since we never unload the classes themselves as the scope keys are garbage collected), as long as you only have one instance of the pattern engine and of each unique rule and named function. This is how it works during static conversion (for which it was designed).

Problem is, now we instantiate PatternEngine for every dynamic conversion operation and the rule sets and named functions are loaded as new instances constantly. So, now the cache is not useful across dynamic conversion runs (different instances of the pattern engine and all other scope key objects). Not only do we have the original class leak, but we are compiling the same expressions into new classes and instantiating them all the time. While the instances will be garbage collected when the scope objects are freed, the classes will add up in PermGen heap.

I don't want to unload the classes, since there are a finite number of them if we fix the re-compilation problem. Thus, PermGen should reach a steady state eventually. I think the fix lies in correcting the cache behavior, but I have to think more on how best to do that.

#2 - 03/30/2014 03:27 PM - Eric Faulhaber

- Assignee set to Eric Faulhaber

This is one source of the performance problems we are seeing with the dynamic database features, though it is not purely a performance issue. I am not yet sure of the degree to which we will address this issue in M11, since we are looking eventually to replace the expression compiler wholesale with a more efficient Scala implementation of TRPL.

It may be that some clever caching of dynamically converted artifacts (i.e., temp-tables and queries) may alleviate this issue to a large degree for the purposes of M11. However, I suspect that large scale testing will probably require we address this sooner rather than later, and it may be that I can address the expression caching with some relatively low-effort changes.

#3 - 04/22/2014 11:30 AM - Eric Faulhaber

- Status changed from New to Hold

With the implementation of the ConversionPool, this problem is largely alleviated, because we are re-using PatternEngine instances, rather than constantly creating new ones. So, the number of compiled expression classes, while large, is finite now. I am putting this issue in hold status, pending additional testing.

#4 - 01/06/2015 06:19 PM - Eric Faulhaber

- % Done changed from 0 to 100

- Status changed from Hold to Closed

The ConversionPool implementation seems to be holding up well with testing. We have a separate PermGen growth issue from all the dynamic query classes we are compiling, but that is not covered by this task.

#5 - 11/16/2016 12:06 PM - Greg Shah

- Target version changed from Milestone 11 to Cleanup and Stablization for Server Features