Database - Feature #2551

Feature # 2274 (Closed): improve performance of new database features

replace runtime compilation of DMOs with ASM bytecode implementation

04/16/2015 11:02 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	**vendor_id:**	GCD

Description

History

#1 - 04/16/2015 11:16 AM - Eric Faulhaber

- Tracker changed from Bug to Feature

The current implementation of dynamic temp-table support generates a DMO schema AST on the fly and uses TRPL to convert this to Java source code, then uses InMemoryCompiler (which is backed by the general purpose Java compiler via javax.tools) to compile this into DMO interface and implementation classes. This is both slow and causes memory issues. The memory problems are mitigated by effectively disabling soft references with the -XX:SoftRefLRUPolicyMSPerMB=0 JVM option, but this is not a production solution.

The new approach is to keep the process roughly the same up to the point where JASTs are created for the DMO interface and implementation classes, but then instead of "brew"-ing these to Java source code and using javac to compile them, use ASM to generate the class files directly and load them with a custom class loader. This should be considerably faster (profiling shows a good deal of time currently is spent compiling via InMemoryCompiler and searching for classes via InMemoryClassLoader), and it will avoid the memory overhead (particularly the soft reference problems) of the general-purpose compiler.

#2 - 04/16/2015 11:35 AM - Eric Faulhaber

Email discussion w/ Constantin regarding the custom class loader needed for this implementation:

Eric,

Another thought about class loaders: we can create a P2JClassLoader which, depending on some directory configuration, it delegates the work to the MultiClassLoader or to the new DynamicClassLoader (or some other loader, if in the future is found needed), before delegating it to the parent class loader. This way we can always set P2JClassLoader as the system class loader, and enable certain custom loaders as needed, without having to worry if a new loader needs to be integrated/reworked/etc.

Thanks,
Constantin

On 16.04.2015 11:22, Constantin Asofiei wrote:

> Eric,
>
> However, the more I've researched the way class loader chaining has to work in order for Hibernate to find the classes with Class.forName(String), it doesn't look like this is feasible, because it seems my class loader has to be the parent of the class loader used to load the P2J and Hibernate classes.

Yes, if not found in the ClassLoader used to load the current class (i.e. a Hibernate class), it delegates the search to the parent class loader, and so on, until the class is found or ClassNotFoundException is thrown. So your new class loader should be the parent of MultiClassLoader (or otherwise, specified as the system class loader).

> This means my class loader would have to stay around for the life of the server, and with it many dynamic DMO classes which will no longer be needed.

Yes, the class loader needs to stay alive, but the classes referenced by it can be discarded. As I understand, the reference chaining goes:

class loader <-> class <- objects <- object references

so as long as the chain remains "class <- objects" (as in, the class is removed from the class loader and all its objects are no longer referenced), the garbage collector should be able to pick it up and discard it, together with all its objects. I think this can be proved using MultiClassLoader: have a long enough loop which loads a jar via addJarLoader and a class from that file, creates instance(s) of that class, discards the instance(s), discards the jar via removeJarLoader. If a OOME is not encountered and the heap/permgen get garbage collected, then this proves that discarding the class from the class loader will remove it from the permgen, too.

I've found some interesting details about class loader issues here: http://www.dynatrace.com/en/javabook/class-loader-issues.html

Thanks,
Constantin

On 16.04.2015 10:54, Eric Faulhaber wrote:

> Constantin,
>
> Thanks for the reply.
>
> I was hoping to use multiple instances of my class loader, to allow garbage collecting these classes by releasing the loader when the classes were no longer in use (and after they aged out of a cache).
>
> However, the more I've researched the way class loader chaining has to work in order for Hibernate to find the classes with Class.forName(String), it doesn't look like this is feasible, because it seems my class loader has to be the parent of the class loader used to load the P2J and Hibernate classes. This means my class loader would have to stay around for the life of the server, and with it many dynamic DMO classes which will no longer be needed.
>
> Are you aware of any reasonable strategy to allow short-lived classes to be garbage collected granularly?
>
> Thanks,
> Eric
>
> On 04/16/2015 03:31 AM, Constantin Asofiei wrote:
>
>> Eric,
>>
>> I'm not aware of any other usage of InMemoryClassLoader expect what was done for the runtime compilation of queries + DMOs. So I think it can be removed from the MultiClassLoader chaining.
>>
>> Thanks,
>> Constantin
>>
>> On 16.04.2015 09:54, Eric Faulhaber wrote:
>>
>>> Constantin,
>>>
>>> Do we have any P2J dependency on InMemoryCompiler and InMemoryClassLoader for anything outside of the persist package (other than InMemoryClassLoader being in the chain of class loaders defined by MultiClassLoader)? I couldn't find anything else, but I don't know if there is anything subtle I'm missing, perhaps specific to Majic? I know the test harness uses InMemoryCompiler, but I'm not aware of any dependency back to P2J for that use.
>>>
>>> We've already removed the runtime compilation of dynamic queries, and now I'm rewriting the code which compiles dynamically defined temp-tables to generate the DMO interface and implementation class bytecode directly. I want to know how much freedom I have to rework the class loader hierarchy. I am not planning on touching MultiClassLoader, other than to restructure the parent chaining.
>>>
>>> Right now, Hibernate is unable to find my classes, since my class loader is outside the chain. I didn't have this problem with my proxy implementation, because I controlled the loading of those classes. However, Hibernate uses Class.forName(String) to retrieve the DMO classes, so I have to do more work to let it find them.
>>>
>>> Thanks,
>>> Eric

#3 - 04/28/2015 11:50 PM - Eric Faulhaber

- File ecf_upd20150427a.zip added

- Status changed from WIP to Closed

- % Done changed from 0 to 100

The attached update implements this feature. It passed regression testing and was committed to bzr rev. 10842.

When applying manually, the following should be deleted:

```
src/com/goldencode/proxy/Utils.java
src/com/goldencode/compile/ (the entire directory)
lib/asm.jar
```

If updating from bazaar, these changes will be taken care of automatically.

#4 - 11/16/2016 12:06 PM - Greg Shah

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

Files

ecf_upd20150427a.zip	267 KB	04/29/2015	Eric Faulhaber