Database - Feature #2276

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

create a work queue of PatternEngine instances for runtime conversion tasks

03/30/2014 04:00 PM - 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

Related issues:

Related to Database - Feature #2282: make number of pattern engine instances ...	**Hold**	**04/11/2014**

History

#1 - 03/30/2014 04:11 PM - Eric Faulhaber

Today, we instantiate multiple PatternEngine instances to perform the various stages of runtime conversion tasks for dynamically defined temp-tables and queries. It is expensive to load all the required XML rule-sets for each of these steps, but then we throw away this configuration, and the very next time we have to run the same stage for a different temp-table or query, we reload everything.

The idea of this task is to create a thread-safe work queue of PatternEngine instances, each preloaded (or lazily loaded) with the appropriate configuration and rule-sets, such that they just need to be handed ASTs to process.

This should help in two ways:

- avoid the overhead of reloading the XML TRPL rules every time they are needed;
- possibly improve the situation described in #2249, since the current caching strategy for compiled expressions is based on com.goldencode.expr.Scope instances, which are represented in the TRPL architecture by instances of PatternEngine and RuleContainer.

Careful investigation is needed to make sure state information is not kept within pattern engines and bled across sessions.

#2 - 03/30/2014 04:12 PM - Eric Faulhaber

- Assignee set to Eric Faulhaber

#3 - 04/03/2014 11:48 PM - Eric Faulhaber

- Status changed from New to WIP

- % Done changed from 0 to 20

To prove that this idea has merit, I've hacked up the dynamic conversion code and made some changes to PatternEngine to permit a different mode of operation. I basically create an open-ended TRPL pipeline for each conversion phase/profile we are interested in running and keep the pattern engine instance that is used for that phase around, instead of discarding it. We load the profile and run the global init-rules during construction. When client code needs to convert, it loads its target ASTs and processes them. Global post-rules are never called, but this is OK, because none of them are relevant to runtime conversion.

This approach means we take the hit of loading the profile (a lot of XML work) and running the global init-rules once per PatternEngine instance, instead of multiple times per dynamic conversion. It also resolves the issue of #2249 for the most part, because each pattern engine object is long-lived, and all the cached, compiled expressions it uses are bound to it. Granted, with a queue of pattern engine objects which are dedicated to the same conversion phase, we will have duplication, but the number of expression classes that will be compiled and loaded into PermGen will be finite, whereas it is ever-growing now.

This mode required some changes to AstSymbolResolver, to set and reset the context local instance of this class. There's still a few days of work to do, to make sure all the pattern workers can be safely loaded and used in multiple pattern engines. There are some potentially problematic static variables in some of them. I have to check the thread safety of this approach in a number of other places as well. I also have to actually implement the queue, as my proof of concept just used a short cut of some instance and class variables to make sure the idea could work. Then, regression testing...

The results so far are very promising. Previous to this change, one customer API typically was taking ~50 seconds to complete on the first pass, and ~43-44 seconds on subsequent passes. Now, the first pass takes ~22-24 seconds, and subsequent passes take ~5 seconds. So, we're seeing nearly an order of magnitude improvement. Once the TRPL code is "warmed up", each conversion step only takes 1-3 ms. A lot of the remaining time (up to hundreds of ms. for each class) is taken in the in-memory compile step. Caching previous conversion results (#2275) should help with this.

I believe I can cut down the first pass time considerably by preloading the pattern engines, instead of setting them up lazily/on-demand, which is how I did it for the POC code. There still will be a lag on the first pass, as expressions are compiled, but I think much of the time is taken up with the I/O and XML processing overhead of loading the rule-sets. I plan to do this in a short-lived thread launched at server startup.

#4 - 04/04/2014 05:13 AM - Constantin Asofiei

Eric,

> running the global init-rules once per PatternEngine instance

I'm a little concerned about this, as it will add a limitation for the runtime conversion mode. If the global init-rules are applied only once per PatternEngine instance and not for each PatternEngine.run call, then you need to make sure that global init-rules don't initiate any global state needed by the down-stream rules (i.e. global maps/sets/etc, which may survive a PatternEngine usage and the data re-used on the next run). Also, we will need to walk around and be aware of this limitation in the future, when making changes to the conversion code which is also ran from the runtime. So if the limitation remains, a comment added to the global init-rules in each pipeline, will be useful.

#5 - 04/04/2014 11:09 AM - Eric Faulhaber

Constantin, I implemented it this way because I was trying to mimic how we use the pattern engine during static conversion: at each conversion phase, we run the global init-rules once, then run against many ASTs. I originally attempted to bracket each run with the global init-rules and post-rules, but I encountered errors which seemed to be about the state of global variables set up in the global init-rules. However, I now think these were unrelated, because I've tried this bracketing again, and now it works without error (and there is no measurable performance impact). So, I will plan to follow your suggestion and keep each run bracketed by the global init-rules and post-rules.

#6 - 04/05/2014 12:21 PM - Eric Faulhaber

- File ecf_upd20140404a.zip added

Interim update. I am uploading it as a safety backup, but it still needs a lot of work and cleanup.

#7 - 04/05/2014 12:26 PM - Eric Faulhaber

- File deleted (ecf_upd20140404a.zip)

#8 - 04/05/2014 12:28 PM - Eric Faulhaber

- File ecf_upd20140404a.zip added

Missed a few files, so I'm re-uploading this update.

#9 - 04/11/2014 08:23 PM - Eric Faulhaber

- % Done changed from 20 to 100

- File ecf_upd20140410a.zip added

- Status changed from WIP to Closed

The attached update has passed regression testing and is committed to bzr rev. 10510.

#10 - 11/16/2016 12:07 PM - Greg Shah

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

Files

ecf_upd20140404a.zip	147 KB	04/05/2014	Eric Faulhaber
ecf_upd20140410a.zip	179 KB	04/12/2014	Eric Faulhaber