Base Language - Feature #6416

explore and implement ideas to make OO 4GL more efficient

05/26/2022 02:30 PM - Greg Shah

Status:	New	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**version:**	
vendor_id:	GCD		

Description

History

#1 - 05/26/2022 02:41 PM - Greg Shah

I think some of the implementation approaches we have had to take for compatibility purposes are mostly unnecessary things which slow down performance and/or hinder scalability. If a given application does not require those features, we should consider disabling them (with an optional configuration flag). Some items that come to mind:

- eliminate usage of FIRST-OBJECT/LAST-OBJECT/NEXT-SIBLING/PREV-SIBLING chaining
  - this can't scale well to millions of objects
  - it should not be a necessary feature for most applications, it is not a good design in my opinion
  - this could be disabled via runtime configuration
- direct construction (real constructors)
  - it seems to me that the indirect construction approach is more expensive than we would want
  - how critical is the error processing/recovery that is implemented by the current approach?
  - for any system that does many constructions of new objects, direct construction may have advantages
  - this would have to be configured at conversion (to emit real new usage and real ctors); I don't know if there is a runtime option needed, we probably would need to modify some runtime behavior

I'm interested in other ideas and thoughts about the above.

#2 - 08/01/2022 10:19 AM - Constantin Asofiei

A part of the problem is the instance initialization in 4GL; the constructors are called **after** the initialization code in the simulated 'execute' Java method is executed. Plus, we require to load the class (the static legacy constructor) before instantiating it.

And the hardest part: the constructors in 4GL act like any other top-level block, and they need to use the BlockManager API - and the super/this constructors must execute from within the BlockManager.internalProcedure context; which Java does not allow - Java super/this constructor calls must be first line in the Java constructor... so we can't call the legacy constructors properly.

#3 - 08/01/2022 10:35 AM - Greg Shah

What specific features of the constructor processing are impacted if we used direct construction (did not properly duplicate the block processing)? I recall that a big part was the control flow when an error occurs. But that should be a rare case. What other aspects are impacted?

I'm trying to evaluate how the OO 4GL code could be modified/hardened such that it could safely use direct construction.

#4 - 08/01/2022 10:44 AM - Constantin Asofiei

Greg Shah wrote:

> What specific features of the constructor processing are impacted if we used direct construction (did not properly duplicate the block processing)?

We can't remove everything, we need at least some processing for reference counting; there is real app code like new some.Test(1, 2, 3, 4)., on a single line, without any assignment; the reference may or may not be assigned directly by executing the constructor, and this 'dangling' reference needs to be registered as 'pending delete' when the current scope ends. And the still-initializing reference can be passed as arguments to other calls, which need to see it as 'valid', at least until the constructor call completes (OK or not).

> I recall that a big part was the control flow when an error occurs. But that should be a rare case. What other aspects are impacted?

well, is not just control flow - the pending-initialized instance needs to be automatically deleted when this happens.

At this time I don't see how we can get away with directly calling Java this/super, without any processing code required before this/super is executed. Plus the 'execute' instance initialization code.

#5 - 08/02/2022 09:57 AM - Marian Edu

Constantin Asofiei wrote:

> A part of the problem is the instance initialization in 4GL; the constructors are called **after** the initialization code in the simulated 'execute' Java method is executed. Plus, we require to load the class (the static legacy constructor) before instantiating it.

Wouldn't be possible to have the whole initialization (static legacy constructor and the 'execute' method) in the Java static constructor?

> And the hardest part: the constructors in 4GL act like any other top-level block, and they need to use the BlockManager API - and the super/this constructors must execute from within the BlockManager.internalProcedure context; which Java does not allow

Do we really need that? Wouldn't be possible for ObjectOps.newInstance to provide the right context? The idea is to do the constructor selection at conversion time probably to avoid doing it at runtime or am I missing something?

> Java super/this constructor calls must be first line in the Java constructor...

4GL works the same way, first line in a constructor must be a call to super/this.

> so we can't call the legacy constructors properly.

Thought the whole idea was to drop the 'legacy' constructors and just use plain Java ones instead?

#6 - 10/06/2022 02:35 AM - Constantin Asofiei

Marian Edu wrote:

> Constantin Asofiei wrote:
>
> > A part of the problem is the instance initialization in 4GL; the constructors are called **after** the initialization code in the simulated 'execute' Java method is executed. Plus, we require to load the class (the static legacy constructor) before instantiating it.
>
> Wouldn't be possible to have the whole initialization (static legacy constructor and the 'execute' method) in the Java static constructor?

No, as the 4GL classes are loaded/initialized per each user context - they are **not** JVM-wide.

> And the hardest part: the constructors in 4GL act like any other top-level block, and they need to use the BlockManager API - and the super/this constructors must execute from within the BlockManager.internalProcedure context; which Java does not allow
>
> Do we really need that? Wouldn't be possible for ObjectOps.newInstance to provide the right context? The idea is to do the constructor selection at conversion time probably to avoid doing it at runtime or am I missing something?
>
> > Java super/this constructor calls must be first line in the Java constructor...
> >
> > 4GL works the same way, first line in a constructor must be a call to super/this.
> >
> > > so we can't call the legacy constructors properly.
> >
> > Thought the whole idea was to drop the 'legacy' constructors and just use plain Java ones instead?

For all these, the tricky parts are:

- the static class initialization (which is context-level, not JVM wide)
- 'execute' method for both instance and static class, which need to be called before the constructor
- each converted constructor is a 4GL-style block - you can't call Java 'super' from it.

What we can improve is caching of the resolved constructor or method resolution (for dynamic-invoke) - we have a cache for RUN statement, but is not used for OO.

#7 - 10/06/2022 02:41 AM - Constantin Asofiei

Other notes:

- cache resolved object methods per type, arguments (see CFOps resolveLegacyEntry)
- cache 'method not found' cases, too.
- ObjectOps.getExecuteMethods
- BlockManager.returnWorker - ObjectOps.pendingInitializing - use a counter for pending initialized objects, instead of walking the stack when there are no pending initialized objects.

#8 - 10/06/2022 10:01 AM - Marian Edu

Constantin Asofiei wrote:

> No, as the 4GL classes are loaded/initialized per each user context - they are **not** JVM-wide.

Ah I see now, always forgetting about the user context in fwd server... then nothing can be really **static** but how about we add **constructor** methods in BaseObject - one for static and one for instance level and the default constructor in BaseObject calls them in that order: static then instance? Those are just no-op in BaseObject but overwrite as needed by OO classes.