Base Language - Bug #7460

implement Progress.Lang.Class:getMethods

06/22/2023 09:56 AM - Constantin Asofiei

Status:	Merge Pending	**Start date:**	
Priority:	High	**Due date:**	
Assignee:	Marian Edu	**% Done:**	80%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**case_num:**	
vendor_id:	GCD	**version:**	

Description

Related issues:

Related to Base Language - Feature #6410: implement additional built-in OO cl... **WIP**

History

#2 - 06/22/2023 09:57 AM - Constantin Asofiei

A customer project uses progress.lang.class:getMethods. We need to implement this.

Greg: is progress.lang.class supposed to be fully implemented in some other task?

#3 - 06/22/2023 03:34 PM - Greg Shah

> Greg: is progress.lang.class supposed to be fully implemented in some other task?

Yes, in #6410.

#4 - 06/22/2023 03:34 PM - Greg Shah

- Related to Feature #6410: implement additional built-in OO classes/interfaces added

#5 - 06/22/2023 03:35 PM - Constantin Asofiei

Marian, can this be implemented 'standalone', without additional APIs from progress.lang.classs (or others)?

#6 - 06/22/2023 03:42 PM - Greg Shah

That is a big reason we really need to get 6410a merged into trunk. It at least provides conversion support for all of those classes (or that is my understanding).

The runtime support also needs to be written.

#7 - 06/22/2023 03:44 PM - Constantin Asofiei

For this task, the priority is the runtime for progress.lang.class:getMethods. For current customers, we already have at least the stubs for the skeletons (otherwise the code would not compile).

#8 - 06/30/2023 11:08 AM - Greg Shah

- Assignee set to Marian Edu

What is the effort to implement this?

I would expect it to go into branch 6410a.

#9 - 07/25/2023 01:04 PM - Marian Edu

Did pushed local changes in #6410a - rev 14556, probably missed release notes.

The idea was to directly use the Java method reference in LegacyMethod, maybe InternalEntry could be a better approach but I don't see how that could work with getMethods. As opposed to getMethod where there is some method resolution based on name/parameters for getMethods we will have to start from the Java class and use pure Java reflection, really don't see how we could use the SourceMapper here. Anyway, we need to have Parameter implemented as well as part of this.

Did changed a bit the rules for access mode resolution - the default in 4GL is 'public', for 'package-protected' and 'package-private' I think we might need to extend the LegacySignature to keep that information since there is only one 'package' access mode in Java.

#10 - 09/11/2023 08:11 AM - Constantin Asofiei

Review for 6410a rev 14571:

- LegacyClass.java
  - we need to cache the Legacy4GLMethod instances and resolve them via the InternalEntry reference (which is unique when returned by ControlFlowOps.resolveLegacyEntry, so an identity map can be used).
  - ErrorManager.recordOrThrowError must be followed by a return.
  - getMethod (and others) are not using a BlockManager.function - I don't think this is OK, for i.e. ErrorManager.recordOrThrowError behavior. Is there a test I can look at?
- ParameterList.java
  - ErrorManager.recordOrThrowError must be followed by a return.
- Legacy4GLMethod.java
  - remove the import edu.emory.mathcs.backport.java.util.Arrays; line
  - fix the import to use i.e. import java.util.*.
  - this constructor:

    ```
    public Legacy4GLMethod(LegacyClass cls, Method javaMethod)
    {
       this.originatingClass = cls;
       this.ie = SourceNameMapper.buildInternalEntry(_getSignature(), javaMethod.getName());
    }
    ```

    must explicitly set ie.setMethod(javaMethod), so the call will not re-resolve the target. But considering the cache mention, ControlFlowOps.resolveLegacyEntry needs to be used. So this will no longer be a problem.
  - the invoke methods really need to be in their own BlockManager.function, as otherwise the error processing will break.
  - getDeclaringClass is using ObjectOps.getLegacyClass which can raise an ERROR condition - BlockManager.function is required.
  - cache the toString representation
- Parameter.java
  - did you check both dynamic extent and fixed extent?
  - the toString representation can be cached.
- ControlFlowOps.java
  - moving prepareArguments(ie, args); after the if (!validArguments is not correct; arguments may exist as i.e. BaseDataTypeVariable which need to be processed before validating them. Do you have a test?

#11 - 09/11/2023 10:52 AM - Marian Edu

Constantin Asofiei wrote:

> Review for 6410a rev 14571:
>
> - LegacyClass.java
> - we need to cache the Legacy4GLMethod instances

Not that I have anything against using cache in general but since getMethods uses flags - (aka a EnumSet) we can only use a cache if we build all methods regardless of the flags used and then filter that. I really don't expect any performance gain in using a cache here, when using reflection most common scenario is one gets a LegacyClass instance, find a method using getMethod or getMethods and call invoke on it. When using getMethods because of the flags we need to get all methods (static/instance, all public/protected/private) cause we do not know what was already cached and what not, wether or not building that cache and avoid reading methods again will give any actual improvement but it turns out the LagacyClass are cached as well so the cache we build is there to stay so I don't have anything against adding this is you think its worth it.

> and resolve them via the InternalEntry reference (which is unique when returned by ControlFlowOps.resolveLegacyEntry, so an identity map can be used).

The resolveLegacyEntry takes **actual parameters** to resolve (modes and arguments) and for getMethods we do not have anything like that, beside I know the methods and I don't need to call any resolve to get it - for getMethods I mean, for getMethod this is what we do already.

> - ErrorManager.recordOrThrowError must be followed by a return.
> - getMethod (and others) are not using a BlockManager.function - I don't think this is OK, for i.e. ErrorManager.recordOrThrowError behavior. Is there a test I can look at?

There are a bunch of unit tests for reflection, couldn't get the ABLUnit to work in FWD so had to write quick and dirty test procedure but as far as I've could see debuging it the errors seems to pop-out just fine, I've only used silent (no-error) and the errors were correctly reported in error-status. Otherwise I don't see why I couldn't put everything in BlockManager.function in cases where there are any errors thrown.

> - ParameterList.java
> - ErrorManager.recordOrThrowError must be followed by a return.

Will fix that, as said things seems to work correctly as-is at least when used with no-error & error-status.

> - Legacy4GLMethod.java
> - remove the import edu.emory.mathcs.backport.java.util.Arrays; line
> - fix the import to use i.e. import java.util.*.

Sure thing, will do.

> - this constructor:
> [...]
> must explicitly set ie.setMethod(javaMethod), so the call will not re-resolve the target.

That is correct, that one is protected so will probably need to add an override that takes the method as additional parameter.

> But considering the cache mention, ControlFlowOps.resolveLegacyEntry needs to be used. So this will no longer be a problem.
>
> - the invoke methods really need to be in their own BlockManager.function, as otherwise the error processing will break.
> - getDeclaringClass is using ObjectOps.getLegacyClass which can raise an ERROR condition - BlockManager.function is required.

Sure, will add the function block - was just following the approach in LegacyClass.invoke but you're the master in error management so one more lambda block won't hurt I guess.

- cache the toString representation

Will do, 'cache is a good thing'... until it isn't :)

- Parameter.java
  - did you check both dynamic extent and fixed extent?

Yes, dynamic as in not set - extent returns unknown in that case, 0 if it's not an extent and the extent value for fixed extent.

- the toString representation can be cached.

Sure thing :)

- ControlFlowOps.java
  - moving prepareArguments(ie, args); after the if (!validArguments is not correct; arguments may exist as i.e. BaseDataTypeVariable which need to be processed before validating them. Do you have a test?

The issue was with NPE when one sends less attributes than in the LegacySignature, I can try to fix that without changing the order of method calls since what you say does make sense.

Thanks

#12 - 09/11/2023 11:24 AM - Constantin Asofiei

Marian Edu wrote:

- ErrorManager.recordOrThrowError must be followed by a return.

Will fix that, as said things seems to work correctly as-is at least when used with no-error & error-status.
...
Sure, will add the function block - was just following the approach in LegacyClass.invoke but you're the master in error management so one more lambda block won't hurt I guess.

Related to my concern about this and others - I see now that LegacyClass.new_, LegacyClass.invoke all do not use BlockManager. My worry is about structured exceptions - if Progress.Lang.Class assumes ROUTINE-LEVE ON ERROR UNDO, THROW, then this may not work properly in FWD without BlockManager. Please create a separate task about this to not delay this task.

- cache the toString representation

Will do, 'cache is a good thing'... until it isn't :)

These Parameter, Legacy4GLMethod, etc, instances are immutable, right?

- ControlFlowOps.java
  - moving prepareArguments(ie, args); after the if (!validArguments is not correct; arguments may exist as i.e. BaseDataTypeVariable which need to be processed before validating them. Do you have a test?

The issue was with NPE when one sends less attributes than in the LegacySignature, I can try to fix that without changing the order of method calls since what you say does make sense.

If you have trouble fixing this, please post a simple test.

Regarding the cache of Legacy4GLMethod instances - my point was to have a Map<InternalEntry, Legacy4GLMethod cache, so that these are created only once. But I think I understand your concern now, the resolution of the methods is done via Java and not via SourceNameMapper; please take a look if you can rework the getMethods code to rely on SourceNameMapper - all the info is there, although not necessarily accessible via existing APIs.

#13 - 09/15/2023 11:13 AM - Greg Shah

Marian: Are you close to committing the next version of this?

#14 - 09/15/2023 01:41 PM - Marian Edu

Greg Shah wrote:

> Marian: Are you close to committing the next version of this?

I'll try to push that by Monday, got sidetracked by some sikuli stuff to help with mouse usage :(

#16 - 09/18/2023 01:18 PM - Marian Edu

Constantin Asofiei wrote:

> These Parameter, Legacy4GLMethod, etc, instances are immutable, right?

There seems to be an issue with this one, since the toString method is override one can't tell from the 4GL side if two objects are actually the same instance - Equals can well return true even if those aren't really immutable. Anyway, for Parameter those doesn't look like being cached at all - calling GetParameters on the same Legacy4GLMethod instance always return different instances of parameters, at least Equals returns false when trying to compare them.

Equals on the Legacy4GLMethod on the other hand returns true for methods declared in the class itself and false if declared in one of the base class in the hierarchy. I have no idea how this works, even if it will only cache the declared methods then the one in the base classes will be cached in their corresponding classes so will ultimately be cached too, how come those are not equals is a mystery to me :(

The classes does seems to be immutable, calling GetClass again returns the same instance - or at least this is what Equals seems to imply.

#17 - 09/18/2023 01:51 PM - Constantin Asofiei

Marian, what happens if you use = operator, and not Equals/ToString methods? If they are the same reference, this will be true.

#18 - 09/18/2023 02:00 PM - Marian Edu

Constantin Asofiei wrote:

> Marian, what happens if you use = operator, and not Equals/ToString methods? If they are the same reference, this will be true.

Did tried that, returns the same as Equals so really different instances and looks like no override for Equals - the only things that seems to be cached in the declared methods inside the class, everything else that come from the base classes are new instances.

#19 - 09/18/2023 02:02 PM - Marian Edu

All this doesn't look like expected behaviour imho and since those are all internal objects anyway let me know if you want me to cache all/some/none of them. At this point I can probably go the 4GL way even if it doesn't make much sense to me :(

#20 - 09/18/2023 02:04 PM - Constantin Asofiei

Marian, at this point, lets leave this behavior on the side for now. We need the implementation of progress.lang.method rather sooner than later. Please finish the other parts of the review (limit to this branche's changes, i.e. what I mentioned about invoke to use BlockManager.functio).

#21 - 09/18/2023 03:08 PM - Marian Edu

Constantin Asofiei wrote:

> what I mentioned about invoke to use BlockManager.functio).

Yeah, as said the invoke in LegacyClass does not have that block either and now I see why it does not... what would be the return class type in this case, care to share some light on this one maybe?

The previous block used the object.class but then when the method returns anything but an P.L.O (aka base data type like character) it just throws an invalid data type found during runtime conversion error... the method gets executed fine, it returns the character value but the the block tries to cast that to object which obviously fails so what generic class type should I use for those invoke methods?

For LegacyClass if you can come up with the generic return type for block function I can probably add those blocks but then those will have to be tested, imho since those were used before without those mandatory blocks it might be we're just trying to fix a pain that doesn't exist :)

#22 - 09/18/2023 03:24 PM - Marian Edu

Tried the obvious choice BaseDataType as return type and this could work if we update the blocks to create a proxy if the return data type is BaseDataType, otherwise the new instance call on declaredConstructor will obviously fail :(

Let me know if this is something you think it could work.

#23 - 09/19/2023 03:59 AM - Marian Edu

- Status changed from New to WIP

- % Done changed from 0 to 80

Constantin Asofiei wrote:

> Marian, at this point, lets leave this behavior on the side for now. We need the implementation of progress.lang.method rather sooner than later. Please finish the other parts of the review (limit to this branche's changes, i.e. what I mentioned about invoke to use BlockManager.functio).

Pushed revision #14578 with changes as per code review, did updated the BlockManager to use the 'wrapper' for BaseDataType return type as mentioned before, I've couldn't find any other way to make it work with a function block without that change.

Reverted the changes in ControlFlowOps.java so the parameters are prepared before validation, the error happens when you pass more arguments than the method expects - line #5187 (not an NPE but some IndexOutOfBound):

```
char mode = modes.charAt(i);
...
LegacyParameter lpar = lsig.parameters()[i];
```

A later commit rev #14579 might be a quick fix for this, not sure if that adds any side-effects though :(

The invoke methods in LegacyClass does still **eat** the errors, at least when running with no-error the error-status is not being set - tried without no-error and no error pop-up either :(

Why is invokeImpl behaving like this - assuming no-error and then fiddling with the ErrorManager I do not know but I imagine this could be already used in many places so changing anything here needs more testing, proved one understand the logic behind it in the first place :)

#24 - 09/19/2023 04:00 AM - Marian Edu

- Status changed from WIP to Review

Please review.

#25 - 09/20/2023 10:12 AM - Constantin Asofiei

Marian, I'm working on the review. I'll do any required changes directly on the branch and will prepare it for merge.

On a side note, I'm trying to run the tests from tests.oo.progress.reflect for constructor, method and parameter, but (some of the) failures look like the tests are still WIP.

#26 - 09/20/2023 02:10 PM - Marian Edu

Constantin Asofiei wrote:

> On a side note, I'm trying to run the tests from tests.oo.progress.reflect for constructor, method and parameter, but (some of the) failures look like the tests are still WIP.

Indeed, not even WIP, in our system the status is NEW, it looks like only the skeleton test were made to check conversion support in FWD, only a few hours recorded for each of those classes so definitively no real testcases.

We did ‘convert’ everything to ABLUnit although many of the OO classes does not have complete tests but only mock-ups. There was a very limited set of OO classes that were implemented and we have extensive tests for - core string, memptr, bytebucket, collections and net package as far as i remember.

#27 - 09/22/2023 10:52 AM - Constantin Asofiei

Greg, I think 6410a rev 14777 can be merged to trunk.

#28 - 09/22/2023 10:55 AM - Greg Shah

Go ahead now.

#29 - 09/22/2023 11:12 AM - Constantin Asofiei

Branch 6410a was merged to trunk rev 14751 and archived.

#30 - 09/22/2023 11:50 AM - Greg Shah

Do we need a follow-up issue to resolve the remaining features/open items? Or should we clear them now, here?

#31 - 09/22/2023 12:06 PM - Constantin Asofiei

For what we've been discussing about Progress.Reflect.Method related to caching (if is needed or not) and to the = operator, we can delay in another task.

But, I'd like to have a stable suite of OEUnit tests related especially to invoke (at Method or Class) and be able to run that in FWD, and fix any issues. I think this is the priority related to this.

#32 - 11/30/2023 12:52 PM - Constantin Asofiei

Created task branch 7460a from trunk rev 14850.

7460a rev 14851 has a regression fix: Do not use 'TypeFactory.object' in cases where the skeleton class is not instantiated via ObjectOps (like the reflection classes), as this will force pending ObjectOps scopeable registration, which never gets executed, as the legacy constructors are not used.

#33 - 11/30/2023 01:38 PM - Greg Shah

- Status changed from Review to Merge Pending

Code Review Task Branch 7460a Revisions 14850 and 14851

No objections.

#34 - 11/30/2023 01:53 PM - Constantin Asofiei

Branch 7460a was merged into trunk as rev. 14851 and archived.

#35 - 12/04/2023 02:50 AM - Marian Edu

Constantin Asofiei wrote:

> Created task branch 7460a from trunk rev 14850.
>
> 7460a rev 14851 has a regression fix: Do not use 'TypeFactory.object' in cases where the skeleton class is not instantiated via ObjectOps (like the reflection classes), as this will force pending ObjectOps scopeable registration, which never gets executed, as the legacy constructors are not used.

Constantin, does this mean the 6410 branch needs to be updated with those changes?

Thanks

#36 - 12/04/2023 03:00 AM - Constantin Asofiei

Yes, I'll need to rebase. Are all changes in 6410b?

#37 - 12/04/2023 03:19 AM - Marian Edu

Constantin Asofiei wrote:

> Yes, I'll need to rebase. Are all changes in 6410b?

I have some local changes but not ready to push them, just rebase it and will merge the local changes on my side.

Thanks

#38 - 12/04/2023 03:25 AM - Constantin Asofiei

6410b rev 14828 was rebased from trunk rev 14856 - new rev 14860