Database - Bug #3471

incremental conversion is regressed

02/08/2018 11:44 AM - Ovidiu Maxiniuc

Status:	Closed	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Constantin Asofiei	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**case_num:**	
vendor_id:	GCD		

Description

Related issues:

Related to Conversion Tools - Feature #4644: add incremental reporting support	**New**	
Related to Conversion Tools - Bug #4867: Incremental conversion: change detec...	**New**	**08/31/2020**

History

#1 - 02/08/2018 12:03 PM - Ovidiu Maxiniuc

Eric tried to convert incrementally with f1+cb using a branch based on trunk 11218. This technique has been valuable when it is needed to convert just a subset of files in a previously converted application, so we don't have to wait many hours for a full re-conversion in order to test a fix candidate.

However, this is now broken, apparently related to a recent change regarding the schema-triggers.xml file:

```
------------------------------------------------------------------------------
Business Logic Base Structure
------------------------------------------------------------------------------

EXPRESSION EXECUTION ERROR:
---------------------------
xmlTriggerRoot = xml.parse("schema-triggers.xml", false, null)
                     ^  { schema-triggers.xml (No such file or directory) }
---------------------------
Elapsed job time:  00:00:00.020
ERROR:
java.lang.RuntimeException: ERROR!  Active Rule:
-----------------------
      RULE REPORT
-----------------------
Rule Type :   INIT
Source AST:  null
Copy AST  :  null
Condition :  xmlTriggerRoot = xml.parse("schema-triggers.xml", false, null)
Loop      :  false
--- END RULE REPORT ---

    at com.goldencode.p2j.pattern.PatternEngine.run(PatternEngine.java:1078)
    at com.goldencode.p2j.convert.ConversionDriver.processTrees(ConversionDriver.java:1128)
    at com.goldencode.p2j.convert.ConversionDriver.back(ConversionDriver.java:989)
    at com.goldencode.p2j.convert.ConversionDriver.main(ConversionDriver.java:2017)
Caused by: com.goldencode.expr.ExpressionException: Expression execution error @1:22
    at com.goldencode.expr.Expression.execute(Expression.java:484)
    at com.goldencode.p2j.pattern.Rule.apply(Rule.java:497)
    at com.goldencode.p2j.pattern.Rule.executeActions(Rule.java:745)
    at com.goldencode.p2j.pattern.Rule.coreProcessing(Rule.java:712)
    at com.goldencode.p2j.pattern.Rule.apply(Rule.java:534)
    at com.goldencode.p2j.pattern.PatternEngine.applyGlobal(PatternEngine.java:1680)
    at com.goldencode.p2j.pattern.PatternEngine.run(PatternEngine.java:1014)
    ... 3 more
Caused by: java.io.FileNotFoundException: schema-triggers.xml (No such file or directory)
    at java.io.FileInputStream.open0(Native Method)
    at java.io.FileInputStream.open(FileInputStream.java:195)
```

```
    at java.io.FileInputStream.<init>(FileInputStream.java:138)
    at java.io.FileInputStream.<init>(FileInputStream.java:93)
    at com.goldencode.util.XmlHelper.parse(XmlHelper.java:412)
    at com.goldencode.p2j.xml.XmlPatternWorker$Library.parse(XmlPatternWorker.java:355)
    at com.goldencode.expr.CE6467.execute(Unknown Source)
    at com.goldencode.expr.Expression.execute(Expression.java:391)
    ... 9 more
```

Technical details:

The (intermediary) file schema-trigger.xml carries some information (namely the procedures designed as triggers) from one stage of the conversion (M0/p20.xml) to another (CB/base_structure.xml). These are trigger-procedure candidates only, the respective procedures are not known to exist at the moment when the intermediary xml file is created and even more, the Java-name is also not know yet.

In CB stage the intermediary file is read. Each file that is coded as a trigger is checked against this list and if there is a match the entry is updated with final data. Unmatched entries are logged and dropped. The final list is saved to destination location. The base_structure.xml expected the file to be present in normal conversion, and intentionally abends when file was missing.

Assuming the M0 stage was already executed at least once, we should already have the intermediary schema-trigger.xml. If we are executing a fX+CB then the database schema (including its trigger configuration) remains unchanged. The CB will reread it and check whether new triggers were added or removed and recreate the final schema-trigger.xml accordingly. OTOH, it soesn't make sense to run the conversion with in fX+CB mode if database schema was altered.

There is also the case when the project does not use a permanent database schema. This is a bit strange, but not unseen. In this case the M0 phase is indeed not required so the schema-trigger.xml is also useless. So, in this case, the CB should detect this and not create its final version. The build.xml should also be aware of whether the file was generated or not.

To fix the incremental conversion I propose the following steps:

1. rename the intermediary file to schema-trigger-candidates.xml (the intermediary marking can be kept although not directly used);
2. if the intermediary file is missing, the CB stage will not crash but will treat this as a normal situation and will not create the final schema-trigger.xml in destination directory. I think DatabaseTriggerManager will handle naturally the absence of the file. However, a prominent warning in the log/console should be issued if the intermediate file is expected but missing;
3. do not drop the intermediary file. The final version will keep current name and will be used by build.xml as usual. Also, the ant clean-generated target should clean up the intermediate file as part of full conversion.

#2 - 02/09/2018 07:06 PM - Ovidiu Maxiniuc

I had the chance to do some incremental conversion today and I realized that the implementation of schema-trigger.xml was broken from the very beginning, the fact that I removed the intermediary file only surfaced one of the aspects. How is it broken? It is broken in the sense that if only a subset of files are reconverted with F_+CB, the refreshed schema-trigger.xml will only contain information about these files. The file is rebuild on intermediary data, the old 'final' information is overwritten. This is probably why, about a month ago, Eric provided me with a set of already converted sources, but unknowingly that the schema-triggers file contained in the archive was corrupted as result of the broken incremental conversion. If F_+CB is executed for all files from project, then the file is fine.

How to fix this?
Probably we will need to keep both version of the file synchronized. Or better keep only the file located in the root of the project and delegate the copy operation (to /dmo directory, where the runtime expects it) to ant (in build.xml). This is not perfect because the content is incremental, removing triggers from project will leave 'rotten' data. A clean/full reconvert is needed to be sure the file is correctly generated.

#3 - 02/10/2018 09:34 AM - Greg Shah

> This is not perfect because the content is incremental, removing triggers from project will leave 'rotten' data. A clean/full reconvert is needed to be sure the file is correctly generated.

For now this seems like a reasonable restriction.

#4 - 10/02/2019 01:09 PM - Constantin Asofiei

Full convert/compile will always be required when someones updates FWD, and there are parser changes where the token IDs have changed.

I'll need to create a test suite which shows the incremental conversion/compile issues.

#5 - 10/02/2019 01:09 PM - Constantin Asofiei

- Assignee set to Constantin Asofiei

- Status changed from New to WIP

#6 - 10/03/2019 05:34 PM - Constantin Asofiei

I'm still checking, but the current issues are:

- for OO, the tricky part is keeping the Java converted method name consistent. If you recall, we need to keep this consistent in all method overrides. I'm not sure yet how to solve this, but some way or another, I need all the application's .cls files loaded in memory (in terms of ClassDefinition instances).
- the 'signature change' issue in name_map.xml - this is not consistent if a procedure/function signature gets changed.
- shared frames and shared menus - these have dependancies on the master definition.

Another assumption is that the conversion will be ran in F1+CB mode, to avoid the schema parsing/generation.

> for OO, the tricky part is keeping the Java converted method name consistent. If you recall, we need to keep this consistent in all method overrides. I'm not sure yet how to solve this, but some way or another, I need all the application's .cls files loaded in memory (in terms of ClassDefinition instances).

I think it is reasonable to require that the entire project be converted at least once as a batch. Given that assumption, I think the problem can be solved by:

1. Using the existing ASTs to obtain any information needed about converted names, API signatures, temp-table definitions and anything else needed.
   - If we have to save off more information into the AST, that is OK.
   - For OO we will have to implement an alternate load (from AST) of the ClassDefinition, ConvertedClassName and whatever other state is needed to lookup the names. We used to do that and it seems pretty straightforward as an approach.
   - Using ASTs for this is the natural approach to solve the problem because it matches how we handle things already.
2. For anything that cannot be calculated from existing ASTs, we can store additional data on a "cross-project" basis.
   - This would help for items that must be disambiguated from existing values.
   - For example, during temp-table conversion, we need to store additional state that is outside of any given AST. The mapping of the temp-table name (for a specific procedure/class) to the unique DMO name is needed at a minimum.
3. Create a tool to calculate the incremental list of files to be converted as a batch.
   - On each successful run, we store state about every ACTUALLY USED source file (procedures, classes AND includes) in the project.
   - This state would include the normalized filename, timestamp of last change, file size and checksum/hash.
   - We calculate the changed files. If the timestamp/size has not changed, there are no edits. If the timestamp is different, we check the checksum/hash. Different size always means a changed file.
   - For each changed file, we calculate those files that are dependent upon it and add them to the list.
     - For an include, any procedure or class that references it. The pphints data stores this, but I think it would be better to aggregate this across the project.
     - For a procedure, just that procedure itself. I don't think we need to consider calling locations here.
     - For a class, all of its subclasses and procedures/classes that use the class. We would need to store a list of the referenced classes on a cross-project basis to make this efficient.
   - We can expose this as a command line utility so a developer can see the implications of their changes.
   - We also should add an "incremental conversion" mode to ConversionDriver to calculate and use that list automatically.
4. Any cross-project data should be stored in an H2 database. Over time we are going to move all of our conversion processing into a database (H2 and possibly Janus Graph). This seems like a natural place to start, instead of adding more one-off conversion artifacts.

If I haven't missed something, this would solve the problem permanently. What do you think?

#8 - 01/27/2020 09:32 AM - Constantin Asofiei

I think it will take ~4-6 weeks to complete and stabilize. I've been working on identifying and analyzing the parts which need to end up in the DB (XML files, TRPL and pattern worker collections); I still have some unknowns for the schema part (both perm and tt).

This is without:

- the 'automatic incremental conversion' - I plan to rely on an explicit list of files to 'incrementally convert'. I think this needs some extra ~3-4 days of work and test, maybe less if things "clear up" during implementation and I can plug this in easily.

- 'incrementally convert' a single file - my plan is to run full F2+M0+CB on the entire file, and do not skip steps.

#9 - 01/27/2020 09:49 AM - Greg Shah

> automatic incremental conversion

By this do you mean item 3 in #3471-7/?

> I plan to rely on an explicit list of files to 'incrementally convert'.

We would build this list automatically? Or would the user have to give us the list (which would not be automatic)?

> 'incrementally convert' a single file - my plan is to run full F2+M0+CB on the entire file, and do not skip steps.

I'm not clear about what this means.

#10 - 01/27/2020 10:29 AM - Constantin Asofiei

Greg Shah wrote:

> > automatic incremental conversion

> By this do you mean item 3 in #3471-7/?

Yes.

> I plan to rely on an explicit list of files to 'incrementally convert'.

> We would build this list automatically? Or would the user have to give us the list (which would not be automatic)?

First phase, manual; second phase, automatic.

> 'incrementally convert' a single file - my plan is to run full F2+M0+CB on the entire file, and do not skip steps.

> I'm not clear about what this means.

I mean I plan to rely on full conversion of that single file, and not F0+CB or CB only.

#11 - 01/27/2020 11:13 AM - Greg Shah

> I plan to rely on an explicit list of files to 'incrementally convert'.

> We would build this list automatically? Or would the user have to give us the list (which would not be automatic)?

> First phase, manual; second phase, automatic.

The first phase is part of the 4-6 weeks? The second phase is the extra 3-4 days?

> I mean I plan to rely on full conversion of that single file, and not F0+CB or CB only.

I was assuming this is the idea in general for this task. What part of this is not included with the 4-6 weeks? How much extra time is estimated to be needed for this? Isn't this just the same thing as the first phase approach and providing an explicit file list which only has 1 file?

#12 - 01/27/2020 11:26 AM - Constantin Asofiei

Greg Shah wrote:

> The first phase is part of the 4-6 weeks?

Yes.

> The second phase is the extra 3-4 days?

Yes, as I haven't thought in depth about this.

> > I mean I plan to rely on full conversion of that single file, and not F0+CB or CB only.
>
> I was assuming this is the idea in general for this task.

Yes, I agree to this.

> What part of this is not included with the 4-6 weeks? How much extra time is estimated to be needed for this? Isn't this just the same thing as the first phase approach and providing an explicit file list which only has 1 file?

If you are referring to the automatic way of determining 'the changed files so they are to be incrementally converted', then at this time I don't have something which is not included.

When I mention F0+CB, I mean 'partial conversion' of a file, where we skip i.e. the temp-table schema. I don't plan to rely on intermediary ASTs/artifacts when 'incrementally converting' a batch of files (on top of a fully converted project).

#13 - 01/27/2020 11:38 AM - Ovidiu Maxiniuc

I believe the original issue of the task is gone now as recently Adrian replaced the schema-triggers two-stage trigger scan from conversion to runtime lookup of the trigger methods based on the propath at the moment of the event.

#14 - 01/31/2020 11:45 AM - Greg Shah

Constantin: Are you planning to use an H2 database for storing project state?

#15 - 01/31/2020 11:45 AM - Constantin Asofiei

Greg Shah wrote:

> Constantin: Are you planning to use an H2 database for storing project state?

Yes

#16 - 02/04/2020 04:08 PM - Constantin Asofiei

Greg, I have a question about the DictionaryWorker. In FWD, the pattern worker instances are global for each conversion phase (annotations, core conversion, etc - aka the .xml being ran). For DictionaryWorker, we have this:

```
   /** Stores all named dictionaries that are active. */
   private Map<String, ScopedSymbolDictionary<Object>> dictList = null;
```

Considering that a pattern worker is the same instance for every AST file being processed in this conversion phase, I need to 'keep in the DB' any state which can be shared by the ASTs (like name mappings in NameMappingWorker). In the DB, for example when there is a set, I store both the set and the individual entries added to it (a kind of history, if you add bogus twice to the set, I keep both entries, in a separate table, plus the AST/file which saved it). The reason for this is, when we are doing incremental conversions, these sets/maps/etc (in the DB as tables) need to be re-calculated (before conversion starts) so that the files included in the current incremental conversion have their data 'purged' (and the state of the sets/maps is (almost) as if conversion was ran initially without these files).

Now, back to the DictionaryWorker - do you see any usage where the dictList state is shared among different AST files? Because if we keep any state in i.e. the global scope which is required to be used by other files, then I need to move this to the DB - which I'm really hoping not to do. And I couldn't find yet a usage where this would be needed... that's why I need a second pair of eyes on this.

#17 - 02/04/2020 05:22 PM - Greg Shah

I think the core thing to worry about here are cares where deleteDictionary() is not called OR where dictionaries are used in the rulesets with

multi-AST state such as annotations.xml.

The following is the only case I have found which seems to deliberately use the DictionaryWorker for storage/retrieval across different ASTs:

- this_proc_func_defs is modified in annotations.xml and used inside annotations/functions.rules. Please note that there is a typo in the deleteDictionary('this-proc_func_defs') that is in annotations.xml. If we fixed this typo, it probably would break this code.

The following cases do not call deleteDictionary() at the top of the ruleset, so they are collecting data across all ASTs for that pattern engine run, BUT I don't think it is deliberate. These are potential memory leaks and should be fixed.

- annotations/constant_expressions.rules use of "constant"
- annotations/process_reachable_refs.rules use of "reach"
- annotations/query_associations.rules use of "queryNames"
- annotations/record_scoping_post.rules use of "bufferconflicts"
- convert/database_access.rules use of "PreselectQueryName"
- convert/dereference.rules use of "defererence_stack"
- convert/variable_definitions.rules use of "toplvlnblock"
- unreachable/unreachable.xml use of "proc" (via variable procId)

#18 - 02/05/2020 03:32 AM - Constantin Asofiei

Greg Shah wrote:

> I think the core thing to worry about here are cares where deleteDictionary() is not called OR where dictionaries are used in the rulesets with multi-AST state such as annotations.xml.

Thanks, I think this is what I wasn't following properly...

> The following is the only case I have found which seems to deliberately use the DictionaryWorker for storage/retrieval across different ASTs:
>
> - this_proc_func_defs is modified in annotations.xml and used inside annotations/functions.rules. Please note that there is a typo in the deleteDictionary('this-proc_func_defs') that is in annotations.xml. If we fixed this typo, it probably would break this code.

I really think this is a typo - per the comment above, add a dictionary to save the return type for all functions defined in this-procedure, the dictionary needs to be reset; and leaving it as is, it doesn't make sense during conversion, as we can't rely on the conversion's order of processing ASTs to determine the function return type, when is in another AST.

> The following cases do not call deleteDictionary() at the top of the ruleset, so they are collecting data across all ASTs for that pattern engine run, BUT I don't think it is deliberate. These are potential memory leaks and should be fixed.
>
> - annotations/constant_expressions.rules use of "constant"
> - annotations/process_reachable_refs.rules use of "reach"
> - annotations/query_associations.rules use of "queryNames"
> - annotations/record_scoping_post.rules use of "bufferconflicts"
> - convert/database_access.rules use of "PreselectQueryName"
> - convert/dereference.rules use of "defererence_stack"
> - convert/variable_definitions.rules use of "toplvlnblock"
> - unreachable/unreachable.xml use of "proc" (via variable procId)

I'll review these, too, and I'll add the deleteDictionary.

Thanks again!

#19 - 02/06/2020 08:24 AM - Constantin Asofiei

Greg Shah wrote:

> I think the core thing to worry about here are cares where deleteDictionary() is not called

I've fixed the cases where deleteDictionary() was not called.

> OR where dictionaries are used in the rulesets with multi-AST state such as annotations.xml.

In all cases, the dictionary name is not 'per-pipeline', but 'per-AST' (the dictionary is deleted in the per-tree init-rules). So I think we are OK in leaving DictionaryWorker unchanged.

#20 - 02/09/2020 02:50 PM - Constantin Asofiei

Greg, about identifying the 'modified files' from a full conversion project. What I'm thinking is to:

- add all files without an .ast
- for all files with .ast, compute the hash for:
  - the file itself
  - go through the .pphints hierarchy and compute a hash for each one
  - if the hash differs for the file itself or any included file, then add it to the list

I don't think is reliable to use the file timestamp, because if someone copies over a new code source into the abl/ folder, the timestamp might not be correct.

#21 - 02/10/2020 10:59 AM - Greg Shah

> add all files without an .ast

Yes, this makes sense. I presume that this really means "add all files (that would normally be included in a full run) without an .ast".

> for all files with .ast, compute the hash for

Yes.

What about OO dependencies?

#22 - 02/10/2020 11:05 AM - Constantin Asofiei

Greg Shah wrote:

> add all files without an .ast

Yes, this makes sense. I presume that this really means "add all files (that would normally be included in a full run) without an .ast".

Exactly. Not sure how to handle the files which are deleted (or removed from the conversion).

> What about OO dependencies?

My goal is that the incremental conversion will produce the same Java method names as the full run, for these files. But I haven't reached this part yet.

#23 - 02/10/2020 11:34 AM - Greg Shah

> Not sure how to handle the files which are deleted (or removed from the conversion).

If the 4GL source files are deleted, then we should remove the artifacts associated with the deleted files.

#25 - 04/13/2020 07:27 AM - Constantin Asofiei

Greg, 3471a rev 11373 is ready for review

#26 - 04/13/2020 04:34 PM - Greg Shah

Code Review Task Branch 3471a Revision 11373

So far, I've very impressed.

Part 1

I'm posting the review in pieces since there is so much to review. This part is about all the classes in com.goldencode.p2j.convert.db.

1. In the calling locations of helper.persistCollection() it would be better if the types of the passed lambda were not inferred ((ps, key) -> ...). It was not obvious on first reading that ps was a PreparedStatement. An example is in LoggedCollection.persist(String, DBHelper, BiFunction<AstKey, Object, Object[]>).

2. I don't understand the purpose of the work tables and the matching regular tables. For example, in the H2Map constructor the code with // load the main table is never executed (it is commented out), while the data does seem to be loaded from the work table. But in the persist() method, we store the code data in the main table only. So we never read that data back in. The changes seem to be handled more consistently (the work table is used for persisting changes in persist() and it is read back in the constructor from the work table. Can you explain the design a bit?

3. Why does AstExternalizable store a duplicate of the AST substree and deregister the tree? We still have the AST in memory, but we just can't do a direct lookup because it is not in the astMap.

4. Is com.goldencode.p2j.convert.db better as sub-package of com.goldencode.ast?

The ConversionData should not move. But for the other classes, my initial sense is that they seem more like a generic service rather than TRPL conversion support. The reason why I am thinking about this is that we plan to move all of the conversion into a database at some point.

I only see two real dependencies.

- This would require moving the basic AST processing of AstSymbolResolver into a base class com.goldencode.ast.AstSymbolResolver. I would expect things like registerAst(), registerTree(), deregisterTree(), getAst() would move. For sure, the core processing of AST nodes from the astMap would move. Anything related to rules or pattern processing would stay in the current class which would be renamed com.goldencode.p2j.pattern.PatternSymbolResolver.
- The DBHelper uses p2j.cfg.Configuration which is definitely a conversion thing. I guess we could just pass this value in to the constructor instead.

On the other hand, I don't want to disrupt things at this point. So we may want to just keep this in mind as an idea. What do you think?

#27 - 04/13/2020 06:33 PM - Constantin Asofiei

Greg Shah wrote:

> 1. In the calling locations of helper.persistCollection() it would be better if the types of the passed lambda were not inferred ((ps, key) -> ...). It was not obvious on first reading that ps was a PreparedStatement. An example is in LoggedCollection.persist(String, DBHelper, BiFunction<AstKey, Object, Object[]>).

I agree, that's simple to change, I'll fix it.

> 2. I don't understand the purpose of the work tables and the matching regular tables. For example, in the H2Map constructor the code with // load the main table is never executed (it is commented out), while the data does seem to be loaded from the work table. But in the persist() method, we store the code data in the main table only. So we never read that data back in. The changes seem to be handled more consistently (the work table is used for persisting changes in persist() and it is read back in the constructor from the work table. Can you explain the design a bit?

Some details about how these work:

- the main table is used as a replacement for state which in trunk we have in files (like ui_strings.txt, duplicate function return types, all function return types, etc). Without the main table, you can't access these easily.
- these db-backed collections went through more than 1 iteration before I managed to get to a common point. The goal of the 'work' tables is to keep a history for each collection entry, with this in mind:
  - a put or add performs the operation and logs the AST from where it was performed, and the data which is being stored
  - a containsKey or contains operation logs this, too: consider for example a certain key in a map is being put by one source AST, and other ASTs just check the key and bypass some action. Without logging this access, when can't
    restore the map state after removing the logging from the work table associated with the currently 'incrementally converted' files. For incremental conversion, these collections need to be restored and have a state as if the conversion (previously) was done without these files.
  - a remove operation will clear all logged changes for that key, too.
- there was a case when for a map key, the value can be different, which is solved by the forceMapValue API - for an example, this is the tmpTabNodes map in p2o.xml, which is a AstKey, Ast map. In this case, if the map already has the authoritative peer CLASS element for a temp-table, then that value is used. But, for each temp-table definition, we need to know even the non-authoritative temp-tables, if the file with the authoritative temp-table gets 'incrementally converted'. Here I'm trying to keep the same DMO interface/class names, but is still not perfect yet - for example, to compute the physical H2 temp-table name, a sequence is used; the result will behave the same, but two identical temp-tables may end up in different physical tables after incremental conversion.

About why the collection restore is done only from the work table - if you look in ConversionData.clean(String), this deletes all history in the work table for a currently converted file. And the main collection needs to 'go back in time' and reflect the reality as if these programs were not included in the conversion, initially. Hope it makes sense.

And no, the data is stored from both the main and work tables - see the changes.persist call.

> 3. Why does AstExternalizable store a duplicate of the AST substree and deregister the tree? We still have the AST in memory, but we just can't do a direct lookup because it is not in the astMap.

This is a performance issue, to not keep in memory the full AST file, but only this subtree. What I needed was to remove the parent reference (which hooks the full tree in memory), but I think just setting the AST parent to null could have work. OTOH, this is used only for tmpTabNodes in p2o.xml, which doesn't do any complex lookups in this AST, is just read-only to read something.

Argh, but I think you are right, a subsequent incremental conversion would break this, and the AST will not be able to be loaded a second time (as the AST IDs have changed). I'll fix it.

> 4. Is com.goldencode.p2j.convert.db better as sub-package of com.goldencode.ast?
>
> The ConversionData should not move. But for the other classes, my initial sense is that they seem more like a generic service rather than TRPL conversion support. The reason why I am thinking about this is that we plan to move all of the conversion into a database at some point.
>
> I only see two real dependencies.
>
> - This would require moving the basic AST processing of AstSymbolResolver into a base class com.goldencode.ast.AstSymbolResolver. I would expect things like registerAst(), registerTree(), deregisterTree(), getAst() would move. For sure, the core processing of AST nodes from the astMap would move. Anything related to rules or pattern processing would stay in the current class which would be renamed com.goldencode.p2j.pattern.PatternSymbolResolver.
> - The DBHelper uses p2j.cfg.Configuration which is definitely a conversion thing. I guess we could just pass this value in to the constructor instead.
>
> On the other hand, I don't want to disrupt things at this point. So we may want to just keep this in mind as an idea. What do you think?

I need to think about it :) but yes, these are more like a generic service than TRPL support.

#28 - 04/13/2020 06:54 PM - Greg Shah

Code Review Task Branch 3471a Revision 11373

Part 2

5. The design is quite elegant. The fact that the maps and sets in conversion have been replaced with DB backed versions completely hides the DB in most cases. The exception seems to be this: "the 'get' ensures the set is mapped to DB". I'm not enthusiastic about this, but I can accept it since you have left behind a comment. Overall, it minimized the TRPL changes which I think is the right idea at this time. It is especially powerful in its ability to support the runtime-only TRPL case.

6. Yes, the files in rules/dbref/ are dead. They were never finished. We left them behind in case they were useful, but I don't know if we would use them now.

7. I see that ClassDefinition instances can be "loaded from an AST" which is the "old school" way we did things before pre-scan. I like that we use annotations for the built-in class support. Overall, the OO changes are hard to evaluate. I guess we really need to rely on the testing to confirm that the changes are OK.

8. I see that the externalizable stuff is used in multiple places, for FrameKey and AstKey not just AstExternalizable. Can you explain the idea here? I must have missed where it is actually being flattened/restored.

#29 - 04/13/2020 07:02 PM - Constantin Asofiei

Greg Shah wrote:

> 5. The design is quite elegant. The fact that the maps and sets in conversion have been replaced with DB backed versions completely hides the DB in most cases. The exception seems to be this: "the 'get' ensures the set is mapped to DB". I'm not enthusiastic about this, but I can accept it since you have left behind a comment. Overall, it minimized the TRPL changes which I think is the right idea at this time. It is especially powerful in its ability to support the runtime-only TRPL case.

About the exception: this is needed because for a 'map with a child set/map as value', the put transforms the set/map to a 'child' one. And the caller doesn't know that, so the 'get' ensures the caller has the proper reference to use. I agree, is not that clean, but can't see yet a better way to solve this.

> 7. I see that ClassDefinition instances can be "loaded from an AST" which is the "old school" way we did things before pre-scan. I like that we use annotations for the built-in class support. Overall, the OO changes are hard to evaluate. I guess we really need to rely on the testing to confirm that the changes are OK.

No, is not the same way - the AST is a fully parsed, and converted AST (except for builtin skeleton classes, now I leave behind an AST for them, too). Why I say 'converted' - because I need the converted java name, accessor methods for properties and some other info.

> 8. I see that the externalizable stuff is used in multiple places, for FrameKey and AstKey not just AstExternalizable. Can you explain the idea here? I must have missed where it is actually being flattened/restored.

AstKey is the key for tmpTabNames in p2o.xml and FrameAstKey is the key for frameInterfaces in frame_generator_pre.xml. I don't need a CustomExternalizable as I need for Ast values, as these are already containers for an AST member, and it was easy to make them externalizable, so they can be stored in the db-backed table. H2 will take care automatically of flattening/restoring them.

#30 - 04/14/2020 11:15 AM - Constantin Asofiei

3471a rev 11376 contains a fix for #4384-187 issue related to converting references for builtin classes with no FWD implementation (it was a TODO left behind which I forgot to address...).

#31 - 04/14/2020 11:27 AM - Constantin Asofiei

The lambda issues were addressed in 3471a 11375.

I think this can be merged to trunk, after I finish another conversion for a small project (2 hours or so).

#32 - 04/14/2020 11:58 AM - Greg Shah

Code Review Task Branch 3471a Revision 11376

I'm fine with the changes. You can merge to trunk when your testing succeeds.

What is the list of known todos/limitations for this task?

#33 - 04/14/2020 05:43 PM - Constantin Asofiei

Phase 2 planning:

- move the h2-related code as a service in goldencode.ast
- tmpTabNodes usage - find a better way to reduce the memory footprint (maybe a lazy loading approach?)
- improve to get rid of the required .ast for the builtin classes. The dependency is mainly from ClassDefinition$MemberData.astId field, which is used by ConvertedClassName.addMethod(String, String, Long astId).
- enum support
- some issues related to temp-table for incremental conversion:
  - the physical table name is currently 'incremental only', following the tt%d pattern. So, even if previously a temp-table was mapped to the same physical H2 temp-table as other schema-identical ones, after incremental conversion it will use its own.
  - find other cases where the temp-table DMO interface and impl do not 'remain the same' after the incremental conversion (especially when the authoritative temp-table gets re-converted...)
- try to find a better way to solve the problem with the 'child collection' after a put operation, as this requires a get, for the caller to use the correct reference.

Also, I think there are some issues when converting a class in the middle of a hierarchy, as I've noticed them in a project where some methods got the name changed

#34 - 04/14/2020 06:06 PM - Constantin Asofiei

Branch 3471a was merged to trunk rev 11346 and archived. This adds support for incremental conversion. Not mandatory for current projects to be reconverted.

Some details about how incremental conversion works:

- during conversion, a H2 database is kept with details about various cross-AST state which needs to be preserved
- a new mode was added to ConversionDriver, 'i'. If you use this, only changed files (from a previous conversion with incremental conversion support) will be reconverted. This mode can be enabled always, as if you delete the cvtdb/ folder during cleanup, it will pick up all files.

Also, for this incremental conversion to work, it is mandatory for a set/map var which is global for all ASTs, to be initiated in TRPL via CommonAstSupport APIs, like:

```
createString2IntMap(String)
createString2LongMap(String)
createString2StringMap(String)
createStringSet(String)
createStringMapToStringMap(String)
createStringMapToStringSet(String)
createStringMapToLongSet(String)
createMapAstKeyToString(String)
createMapAstKeyToInt(String)
createMapFrameAstKeyToString(String)
createMapAstKeyToAst(String)
```

For global 'sequence-like' variables, use nextSequenceValue(String).

Also, in case of maps which have as value another collection (set or map), after you performed the 'put' operation, do a 'get' on the same key and re-assign the var referencing the key's value, so that this var is backed to db.

#35 - 04/15/2020 09:46 AM - Greg Shah

- % Done changed from 0 to 100

- Status changed from WIP to Closed

#36 - 04/16/2020 04:03 PM - Greg Shah

Additional notes which may be useful:

The core idea is that the FWD ConversionDriver has an incremental conversion mode ("I") which will detect the list of files which have changed OR which must be reconverted because a dependency has changed. Dependencies are include files. Only that list of files is converted.

This means you can edit the files as needed (or just put new versions of them in place) and FWD can detect the changes. It does not look at the date/time stamps. It uses MD5 hashes of the files to detect a change. Don't just change whitespace because that will change the hash!

In our standard configuration, one can add the 'i' mode to convert.blacklist (as in <arg value="-IXd2"/>) in the build.xml.

After that, run only ant convert.blacklist compile jar (to avoid cleaning the project) to get incremental mode.

If you want to go back to a clean conversion, delete the cvtdb/ folder and this same incremental mode will convert all files. The build.xml can also add this directory to the clean target.

Conversion now requires more memory because we have an in-memory database and more AST files loaded at one time. Where a project needed a minimum of 8GB before, you may need 16GB now. In other applications, we now use 12GB where we used to use 8GB previously. Regardless, more memory will be needed. Make sure you don't stave the conversion process, otherwise the performance will be exceptionally slow. We have an idea of how to reduce this, but it is not implemented yet.

#37 - 04/16/2020 04:21 PM - Constantin Asofiei

4231b rev 11462 contains some performance fixes related to the memory usage.

The idea here was:

- 'unpin' the AST sub-tree reference from memory and keep the AST ID only, when the map's value is an AST (the tmpTabNodes again)
- for the other usage, like AstKey or FrameAstKey maps, we keep registered the sub-tree for this key. Any other keys will re-use this from the registry, and if someone else wants another sub-tree from the same root AST, then it will just load the entire AST (and this is how FWD already behaves).

#38 - 04/23/2020 09:33 AM - Constantin Asofiei

4231b rev ~~11481~~ 11482 contains more performance fixes (again, related to the tmpTabNodes and AstKey):

- collections are cleared after they are persisted. I think this helps the garbage collector to identify them faster.
- records in db-backed collections are inserted 2000 at a time
- AstKey and FrameAstKey keep a copy of the AST instead of the real AST. The idea here is to not pin in memory the entire AST file, as these keys are used across the entire project.
- AstKey computes the hash only once.

The last two items assume the keys are immutable once they used in the map, and this makes sense, as the criteria used to extract info from the AST is for annotations which are previously set (by other rules) and should not be changed by further TRPL rules.

#39 - 05/05/2020 03:44 AM - Constantin Asofiei

Eric, I've noticed something while looking at #4633. The SchemaWorker.joins does not survive for incremental conversion. Considering that for incremental conversion only a subset of files is used, this information gets either lost.

How important is this in a business logic POV? Does it affect the logic (i.e. records will not be retrieved correctly as the joins are not available), or is just a performance issue?

#40 - 05/13/2020 09:33 AM - Constantin Asofiei

Some other parts which are required for incremental conversion:

- the .wsdl generation for exported SOAP services (#3310)
- a TODO in ClassDefinition(SymbolResolver sym, String qname) related to legacy Enum's GetEnum implicitly added methods (these need to be re-registered, as they do not appear in the AST).

#41 - 05/14/2020 04:08 PM - Constantin Asofiei

- Related to Feature #4644: add incremental reporting support added

#42 - 06/20/2020 12:01 PM - Greg Shah

Task branch 4231b has been merged to trunk as revision 11347.

#43 - 08/31/2020 12:16 PM - Vladimir Tsichevski

- Related to Bug #4867: Incremental conversion: change detection algorithm ignores .ext-hints files added

#44 - 04/24/2021 02:52 PM - Constantin Asofiei

3821c rev 12326 fixes incremental conversion when, in full conversion, the non-new shared temp-table exists in the last file (from the conversion file list). This set the 'tmpTabNodes' map to have the non-new shared temp-table AST, which will be missing the DMO implementation name (as that is not required for non-new shared temp-table).

The solution was two-fold:

- do not save in 'tmpTabNodes' non-new shared temp-tables
- for incremental conversion, add all programs which are dependent on that tmpTabNodes AST, and re-convert them.