Runtime Infrastructure - Bug #7443

performance of CentralLogger

06/15/2023 08:08 AM - Constantin Asofiei

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

Description

Related issues:

Related to Runtime Infrastructure - Bug #5703: rationalize, standardize and s...	**Closed**
Related to Runtime Infrastructure - Feature #6692: move FWD to Java 17	**Internal Test**

History

#1 - 06/15/2023 08:08 AM - Constantin Asofiei

- Related to Bug #5703: rationalize, standardize and simplify the client-side log file name configuration added

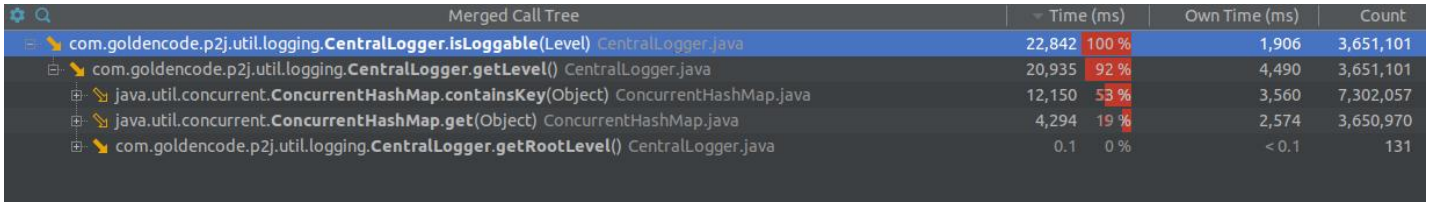
#2 - 06/15/2023 08:13 AM - Constantin Asofiei

- File backtraces.png added

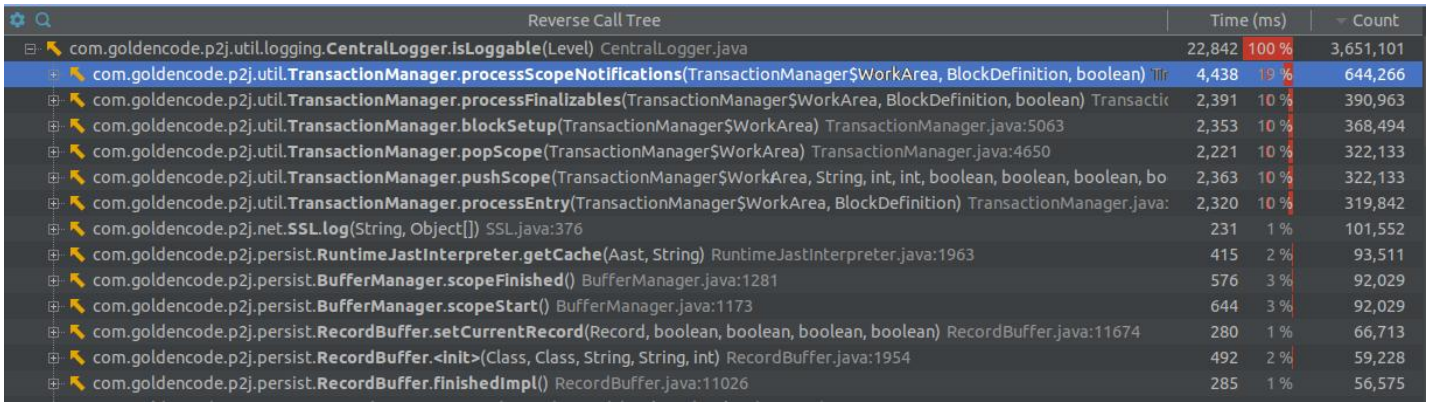
- File callees.png added

CentralLogger.isLoggable is called 3.6 million times when testing an application. In the profiler, this highlights as ~1%.

Its callees are:



and the backtraces:



#4 - 06/15/2023 09:31 AM - Galya B

~1% of resources for 3.6 million executions sounds great to me!

Jokes aside...

We can lower it to 1.8 million executions removing the unnecessary if (LOG.isLoggable(Level.FINE)) that hides concatenation, that can be reworked as passing in the formatting arguments directly to the formatting logger method.

#5 - 06/15/2023 09:32 AM - Constantin Asofiei

Galya B wrote:

> We can lower it to 1.8 million executions removing the unnecessary if (LOG.isLoggable(Level.FINE)) that hides concatenation, that can be reworked as passing in the formatting arguments directly to the formatting logger method.

We need to remove the map lookup. The log level needs to be cached at the logger instance.

#6 - 06/15/2023 09:33 AM - Galya B

Constantin Asofiei wrote:

> We need to remove the map lookup. The log level needs to be cached at the logger instance.

Prejudice! There is no map lookup, it's cached.

#7 - 06/15/2023 09:39 AM - Galya B

Actually the cache map should have been the first lookup in the method. This is my bad.

#8 - 06/15/2023 09:40 AM - Galya B

And we know the complexity of accessing string keyed map. Do you think the concurrency of the map adds to the CPU usage?

#9 - 06/15/2023 09:44 AM - Galya B

There is no free lunch, it's either the CPU or the RAM. If I need to work on this task, I'll remove isLoggable from the logger interface and make sure all concatenations and formattings outside of the logger are moved to the logger after the check is performed. Then I'm sure we will have achieved 0.5%.

#10 - 06/15/2023 09:50 AM - Galya B

Also let's not forget that resource consumption is part of execution of applications and this feature was not existing before, so it's expected the usage to raise somewhat.

That being said, I've been against the excessive use of isLoggable since the beginning.

#11 - 06/15/2023 10:08 AM - Galya B

Constantin, what is the logging level you tested with on this screenshot?

#12 - 06/15/2023 10:13 AM - Constantin Asofiei

This is the entire logging node. The loggers levels is the same from before testing with CentralLogger.

```
        <node class="container" name="logging">
          <node class="container" name="file">
            <node class="string" name="path">
              <node-attribute name="value" value="server_%g.log"/>
            </node>
            <node class="integer" name="rotationLimit">
              <node-attribute name="value" value="0"/>
            </node>
            <node class="integer" name="rotationCount">
              <node-attribute name="value" value="1"/>
            </node>
          </node>
          <node class="container" name="loggers">
            <node class="string" name="com.goldencode.p2j.util">
              <node-attribute name="value" value="INFO"/>
            </node>
            <node class="string" name="com.goldencode.p2j">
              <node-attribute name="value" value="WARNING"/>
            </node>
            <node class="string" name="com.goldencode.p2j.net">
              <node-attribute name="value" value="INFO"/>
            </node>
            <node class="string" name="com.goldencode.p2j.persist">
              <node-attribute name="value" value="INFO"/>
            </node>
          </node>
        </node>
```

#13 - 06/15/2023 10:14 AM - Greg Shah

Bypassing everything but the logging level check has served us well. There is a performance reason those checks were there. Yes, they are ugly. But they are fast. And the cached local flags are even faster. This is why I mentioned that an approach where we use local flags and a listener model to notify in the (very rare case) where the logging levels change is something we need to consider.

#14 - 06/15/2023 10:20 AM - Constantin Asofiei

Galya, the problem is not isLoggable being called excessively. canLog relies too on isLoggable, so getting rid of isLoggable will not solve anything, this will just mean that log will call canLog which will call isLoggable.

The solution is to cache the logging level at the logger instance, and not only for the logger's name. It doesn't matter if the same logger name is being used by multiple logger instances, the logger instance must have its level precomputed.

Regarding map usage: maps are slow 'by design', and concurrent maps just adds more overhead. And where we can do without maps, we must do so.

Keep in mind that 3.6 million calls for isLoggable is not from stress testing this application, you can consider this 'normal usage'.

#15 - 06/15/2023 10:20 AM - Galya B

Greg Shah wrote:

> Bypassing everything but the logging level check has served us well. There is a performance reason those checks were there. Yes, they are ugly. But they are fast. And the cached local flags are even faster. This is why I mentioned that an approach where we use local flags and a listener model to notify in the (very rare case) where the logging levels change is something we need to consider.

I understand, but I don't understand. I need to understand to comply.

1. Have you done performance testing on Java isLoggable implementation? Think about it carefully.
2. What is the performance reason? Memory consumption as far as I know. That is concatenation and formatting. I solved that in a better way that removes one invocation of the same method, when the level is enabled.
3. Local flags and listeners are objects, they take up memory and create non-standard (in that context) complex design. And we don't like objects (I mean lambdas).

I know what you want, but I fear implementing it myself, the same way I feared implementing hundreds of isLoggable calls.

"Performance" should be measurable. I don't see any base measurement here, because noone measured Java isLoggable.

#16 - 06/15/2023 10:29 AM - Galya B

Constantin Asofiei wrote:

> It doesn't matter if the same logger name is being used by multiple logger instances, the logger instance must have its level precomputed.

It has it pre-computed in CACHED_LOGGER_NAME_LEVELS. It's in a static field of type map in the same class. I did a mistake for some reason

though. I had to make the check to the cached map as a first statement in getLevel.

> Regarding map usage: maps are slow 'by design', and concurrent maps just adds more overhead.

I can agree about the concurrent part, because I haven't measure it. I don't agree about the map part. Red-black trees in Java 8 are quick, but let's not get into sorting, because getting a String key is with complexity of 0(1).

> Keep in mind that 3.6 million calls for isLoggable is not from stress testing this application, you can consider this 'normal usage'.

The number is striking, but not more than the number of calls to these base methods, so the percentage will be a constant even with stress testing.

#17 - 06/15/2023 10:44 AM - Galya B

Even if I bend due to pure pressure and implement it, I don't recommend using isLoggable just for the sake of looking safe especially with high logging levels like WARNING and SEVERE.

#18 - 06/15/2023 10:46 AM - Constantin Asofiei

isLoggable is meant to be used when there is a cost with resolving the arguments passed to the log method. That's why I insisted on it - even passing a lambda expression is costly.

Will something like this work with current implementation? (ignore the listener approach required if the parent changes the logging level).

```
--- old/src/com/goldencode/p2j/util/logging/CentralLogger.java
+++ new/src/com/goldencode/p2j/util/logging/CentralLogger.java
@@ -226,6 +226,9 @@
    /** Flag to prevent deadlock on SecurityManager. */
    private boolean excludeSMContext;

+   /** Cached logging level. */
+   private Level level;
+
    /** The bootstrap config reader. */
    private static BootstrapConfig bc;

@@ -244,6 +247,8 @@
       this.loggerName = loggerName;
       this.checkParentLevels = checkParentLevels;
       this.excludeSMContext = excludeSMContext;
+
+      this.level = getLevel();
    }

    /**
@@ -1035,8 +1040,7 @@
     */
    public boolean isLoggable(Level logLevel)
    {
-      Level loggerLevel = getLevel();
-      return loggerLevel == null || logLevel.intValue() >= loggerLevel.intValue();
+      return level == null || logLevel.intValue() >= level.intValue();
```

```
    }

    /**
```

#19 - 06/15/2023 10:49 AM - Galya B

Constantin, I started with a similar implementation, where the state also included the level, but there are other caveats as well. The level comes late for any process - only after the configs are resolved, so we support dynamic levels for that purpose. Also there are (only one for the moment I think) classes that set their own levels.

#20 - 06/15/2023 10:51 AM - Galya B

Give me 1h to rework TransactionManager to my liking and see if it fixes something.

#21 - 06/15/2023 10:52 AM - Galya B

Do we have a branch here?

#22 - 06/15/2023 10:54 AM - Constantin Asofiei

Galya B wrote:

> Give me 1h to rework TransactionManager to my liking and see if it fixes something.

Reworking code is not the way to go. The fix must be in CentralLogger.

> The level comes late for any process - only after the configs are resolved, so we support dynamic levels for that purpose. Also there are (only one for the moment I think) classes that set their own levels.

OK, then we need this implemented, too.

#23 - 06/15/2023 10:57 AM - Galya B

To me removing 50% of the calls sounds quite right. Or am I misreading your screenshot?

#24 - 06/15/2023 10:58 AM - Galya B

Also as I've said, I have a mistake to fix.

#25 - 06/15/2023 10:59 AM - Galya B

Galya B wrote:

> To me removing 50% of the calls sounds quite right. Or am I misreading your screenshot?

I didn't say it right. Improving 50% of the calls.

#26 - 06/15/2023 11:13 AM - Constantin Asofiei

- File isLoggable.png added

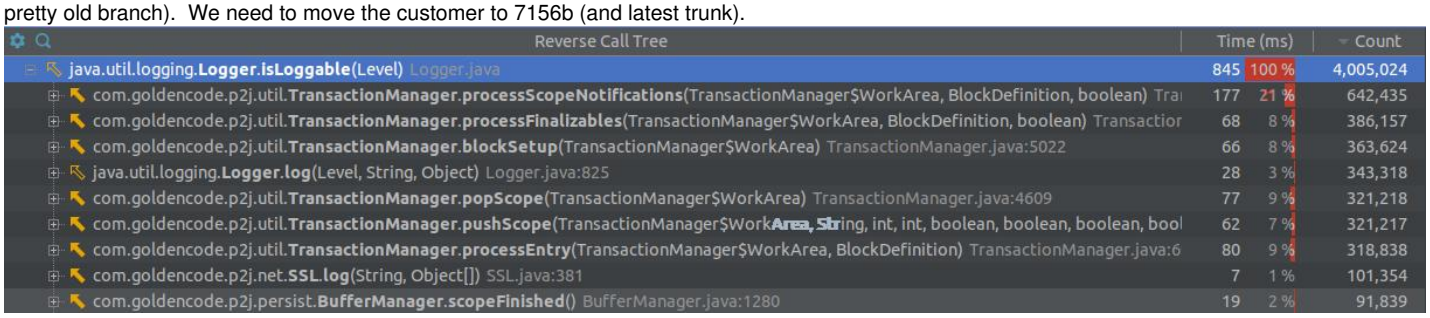
Galya B wrote:

> Galya B wrote:
>
> > To me removing 50% of the calls sounds quite right. Or am I misreading your screenshot?
>
> I didn't say it right. Improving 50% of the calls.

Any API call (from the profiling I do) which is over ~10_000ms is something of concern which needs to be investigated and possible improved. We need a way to make these 3.6 million calls go down to 1000ms or less (so 50% will not do).

Note that canLog (which is called from the actual log method) is called only 30k times. But this is only because of the isLoggable protection, which gets called before actually going down the log method.

You asked about the isLoggable from Java logging. Here it is - 4million calls in 843ms. This is from profiling the same application, but with 7156a (a pretty old branch). We need to move the customer to 7156b (and latest trunk).



#27 - 06/15/2023 11:24 AM - Galya B

I made an expensive mistake, that removed will improve things 3 times... Now getLevel checks in 3 concurrent maps, while it should check only one. That will make the duration about 7614 (22842 / 3).

#28 - 06/15/2023 11:33 AM - Galya B

Also Java logger has a tree :(I wanted to implement a tree as well, but didn't have the capacity of a Java team.

#29 - 06/15/2023 11:41 AM - Galya B

What I'm trying to say is that listeners and more state are a bad idea. It should be implemented with WeakReference although most instances are saved in static fields, but this shouldn't be presumed. And do push of the change instead of listening. Then also consider the implications on all logger methods, especially in the context of logger wrapper classes like CentralLoggerNoModeWrapper.

#30 - 06/15/2023 11:47 AM - Galya B

I'm not sure if that got understood, but all loggers should listen to the change, this means a few hundred objects wrapped in WeakReference objects. Not that getting someone scared by empty numbers is my goal here. Just mentioning.

#31 - 06/15/2023 12:37 PM - Constantin Asofiei

Galya, beside the FWD server startup, what are the condition under which a logger's level can be changed? Here I mean in existing FWD trunk, not the new feature which will allow to change the logging level at runtime, for any logger.

#32 - 06/15/2023 12:38 PM - Greg Shah

1. Have you done performance testing on Java isLoggable implementation? Think about it carefully.

All of our previous code and now these new changes are being profiled extensively.

2. What is the performance reason? Memory consumption as far as I know.

No, this is not about memory. It is 100% about reducing CPU cycles to do the same 4GL compatible work.

Some day, we will worry about memory but not until we've solved known CPU cycle issues. Eliminating map lookups and string concatenation has been proven to make a measurable difference in real application code. That is why we are focused on these specific things.

3. Local flags and listeners are objects, they take up memory and create non-standard (in that context) complex design. And we don't like objects (I mean lambdas).

The costly part of lambdas is when there is variable capture, especially where that capture is of instance data. Regardless of the concerns we have with lambdas, the listener approach for this use case is quite excellent since nearly every server run will have 0 calls to dynamically set levels after server startup. Even if we set the levels once or twice during a server run, who cares? It won't be millions of times. On the other hand, long running servers will have billions of calls to map lookups or string concat that are not needed if only we precalculate and cache the logging levels.

#33 - 06/15/2023 12:39 PM - Greg Shah

> Galya, beside the FWD server startup, what are the condition under which a logger's level can be changed? Here I mean in existing FWD trunk, not the new feature which will allow to change the logging level at runtime, for any logger.

Even after we implement it, the dynamic levels will almost never be used. We do not need to worry about that case.

#34 - 06/16/2023 02:44 AM - Galya B

When invalidating the levels in all instances that will be a performance bottleneck, because all loggers will need to access the concurrent map to get their new levels. If I've implemented a tree for the loggers it would be way better. But I kind of don't want to get into creating a red-black tree in Java myself...

#35 - 06/16/2023 09:01 AM - Greg Shah

Galya B wrote:

> When invalidating the levels in all instances that will be a performance bottleneck, because all loggers will need to access the concurrent map to get their new levels. If I've implemented a tree for the loggers it would be way better. But I kind of don't want to get into creating a red-black tree in Java myself...

This is a non-issue because it will almost never be called. In other words, we are OK if this notification/reset levels is slow.

What can't be slow or even as slow as a single map lookup is the isLoggable().

#36 - 06/16/2023 09:02 AM - Galya B

- Assignee set to Galya B

Greg Shah wrote:

> What can't be slow or even as slow as a single map lookup is the isLoggable().

Roger that. I'll see how to fix it on Monday.

#37 - 06/19/2023 04:19 AM - Galya B

- Status changed from New to WIP

7443a created.

#38 - 06/19/2023 10:40 AM - Galya B

- Status changed from WIP to Review

- % Done changed from 0 to 100

7443a r14628 based on trunk r14627 tested and ready for review.

#39 - 06/19/2023 11:50 AM - Constantin Asofiei

Galya B wrote:

> 7443a r14628 based on trunk r14627 tested and ready for review.

The changes look reasonable, but they need to be tested.

Alexandru: please apply (locally) the 7443a changes to 7156b and run a new round of testing.

#40 - 06/20/2023 01:47 AM - Alexandru Lungu

Constantin Asofiei wrote:

> Alexandru: please apply (locally) the 7443a changes to 7156b and run a new round of testing.

Sure. Enqueuing now.

#41 - 06/20/2023 03:40 AM - Alexandru Lungu

7443a/rev. 14628 doesn't compile:

```
[ant:javac] /home/al2/gcd/branches/7443a/src/com/goldencode/p2j/ui/chui/ThinClient.java:16177: error: exceptio
n ConfigurationException is never thrown in body of corresponding try statement
[ant:javac]       catch (ConfigurationException e)
```

I temporarily removed the catch block to be able to profile.

#42 - 06/20/2023 04:23 AM - Galya B

I pushed r14629 to fix the compilation error. I haven't rebuilt with all.

#43 - 06/20/2023 08:48 AM - Galya B

r14630 reverts LogHelper#createThreadName, requested in #7143-234.

#44 - 06/20/2023 09:14 AM - Greg Shah

Code Review Task Branch 7443a Revision 14630

The change is good. I like that the code is in Utils instead of CentralLogger. Perhaps it would be better for it to be in the security package but Utils is OK for now.

#45 - 06/20/2023 09:42 AM - Alexandru Lungu

I've done one set of profiling tests and got a slight improvement comparing to the state before #7443 (which was +3.1%), but still not close to the original trunk code before CentralLogger (still +1.9%).

However, I did a second set of profiling tests: in the first round of tests, I got a ~26MB log file. I commented an WARNING (that is already fixed on another branch) that popped a lot of times and got to ~5MB of log file. With this second profiling round that generated ~5MB of logs, I got a result that is a bit slower than my trunk base-line before CentralLogger (only +0.45%). If I disable logging, I reach the times I had before CentralLogger, but of course this is not the solution here.

Even if this mostly seems like an IO slow-down, I haven't done any tracing similar to #7443-2 to confirm the cause. Could this be related to more complex formatting than before? We certainly have more detailed logging now (including thread, session, user) and this may cost us time.

#46 - 06/20/2023 09:54 AM - Constantin Asofiei

Alexandru Lungu wrote:

> I commented an WARNING (that is already fixed on another branch)

Is this already in trunk?

> Even if this mostly seems like an IO slow-down,

I'll do another set of profiling and see what happens.

#47 - 06/20/2023 10:00 AM - Alexandru Lungu

Constantin Asofiei wrote:

> Alexandru Lungu wrote:
>
> > I commented an WARNING (that is already fixed on another branch)
>
> Is this already in trunk?

It is about BaseDataTypeFactory instantiating ObjectVar and it is not in trunk - only in 7026e. Without it, you may see lots of Type class com.goldencode.p2j.util.ObjectVar instantiated via reflection, as it is not registered in BaseDataTypeFactory.

```
=== modified file 'src/com/goldencode/p2j/util/BaseDataTypeFactory.java'
--- old/src/com/goldencode/p2j/util/BaseDataTypeFactory.java    2023-05-26 14:37:01 +0000
+++ new/src/com/goldencode/p2j/util/BaseDataTypeFactory.java    2023-06-01 13:09:20 +0000
@@ -8,6 +8,7 @@
 ** 001 RAA 20230525 Created initial version.
 **     AL2 20230525 Ordered types alphabetically.
 **     AL2 20230525 Moved to CentralLogger.
+** 002 RAA 20230601 Added check for ObjectVar.
 */

 /*
@@ -163,6 +164,10 @@
       {
          return (T) new object<>();
       }
+      else if (cls == ObjectVar.class)
+      {
+         return (T) new ObjectVar<>();
+      }
       else if (cls == raw.class)
       {
          return (T) new raw();
```

> > Even if this mostly seems like an IO slow-down,
>
> I'll do another set of profiling and see what happens.

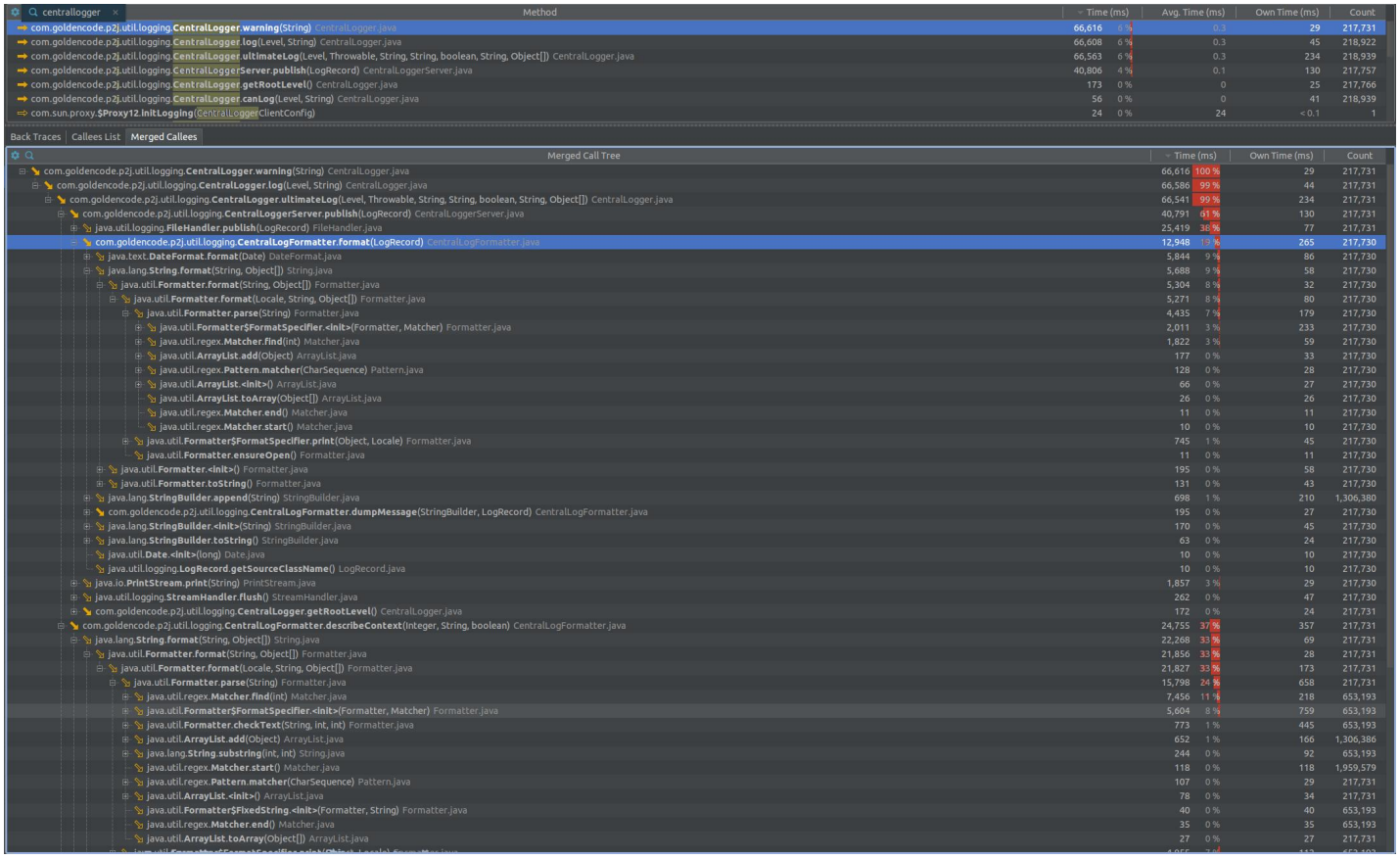
Thank you!

#48 - 06/20/2023 04:41 PM - Constantin Asofiei

- File log_format.png added

Galya, Alexandru I think is right, formatting the message is expensive. Please see how this can be improved. I've attached a profile snapshot of a simple program which just uses LOG.warning some ~217k times.

From this, all the parsing of the format (DateFormat.format and String.format) stands out.



What I've used was this logtest.p code:

```
def var i as int.

do i = 1 to 10000000:
   message i.
end.
```

and replaced the message i. in the converted Java so it looks like this:

```
public class Logtest
{
   /** Logger */
   private static final CentralLogger LOG = CentralLogger.get(Logtest.class);

   @LegacySignature(type = Type.VARIABLE, name = "i")
   integer i = UndoableFactory.integer();

   @LegacySignature(type = Type.MAIN, name = "Logtest.p")
   public void execute()
   {
      externalProcedure(Logtest.this, new Block((Body) () ->
      {
         for (int i = 0; i < 1000000; i++)
         {
            LOG.warning("bla");
         }
      }));
```

```
    }
}
```

Add some code to capture the total time spent (in nanos) for this loop (you can even convert it to a simple Java for loop,), so you can compare with any changes you make.

#49 - 06/21/2023 01:41 AM - Constantin Asofiei

Alexandru Lungu wrote:

> Constantin Asofiei wrote:
>
> > Alexandru Lungu wrote:
> >
> > > I commented an WARNING (that is already fixed on another branch)
> >
> > Is this already in trunk?
>
> It is about BaseDataTypeFactory instantiating ObjectVar and it is not in trunk - only in 7026e. Without it, you may see lots of Type class com.goldencode.p2j.util.ObjectVar instantiated via reflection, as it is not registered in BaseDataTypeFactory.

Alexandru, please commit this change to 7443a. I expect 7443a to be merged to trunk before 7026e.

#50 - 06/21/2023 02:42 AM - Alexandru Lungu

> Alexandru, please commit this change to 7443a. I expect 7443a to be merged to trunk before 7026e.

Done! Committed #7443-47 to 7443a as rev. 14631.

#51 - 06/21/2023 02:42 AM - Galya B

Alexandru, can you please try to run with the following configs:

```
        <node class="container" name="logging">
          <node class="boolean" name="serverSide">
            <node-attribute name="value" value="TRUE"/>
          </node>
        </node>
```

I'm curious to see the difference in performance.

#52 - 06/21/2023 02:50 AM - Galya B

Alexandru Lungu wrote:

> I got a result that is a bit slower than my trunk base-line before CentralLogger (only +0.45%).

Is someone surprised that using logger is more expensive than System.err.print with +0.45%?

Can I run the profiling procedure myself?

#53 - 06/21/2023 03:14 AM - Galya B

Let's set the requirements first before jumping into fixing. If I make CentralLogger#describeContext (24,755%) and CentralLogFormatter#format (12,948%) take no time in execution, this will be an improvement of 37,703% of the +0.45% or the performance degradation of using logger instead of System.err will come down to +0,28035% (improvement with +0.1696635%). If we're happy with that number, then I'm on it to make those two methods take 0ms.

Did I get the task right?

P.S. Numbers edited.

#54 - 06/21/2023 04:01 AM - Constantin Asofiei

Galya, please measure the following, for 1mm log lines:

- the LogTest modified program with CentralLogger and 7443a
- the LogTest modified program with FWD before CentraLogger (using what was before CentraLogger)

This will give us an idea of the time differences.

#55 - 06/21/2023 04:39 AM - Galya B

Constantin Asofiei wrote:

> using what was before CentraLogger

You mean System.err? server.log was "generated" by System.err

#56 - 06/21/2023 07:58 AM - Constantin Asofiei

I don't understand what you mean by System.err. Previously, FWD was using java.util.logging for logging support on server-side.

What we need to compare is com.goldencode.p2j.util.LogHelper.getLogger and CentraLogger. And we need to improve any parts from CentralLogger that can be improved, to get to as close as possible to java logging.

#57 - 06/21/2023 08:02 AM - Galya B

- File logging-performance.zip added

Here it is, the groundbreaking reality (in src.com.galya.LoggingTest):

```
      // 2174ms - 3424ms, redirected 4573ms
      // one of the traditional FWD ways of logging
      systemErr();

      // 32912ms, redirected 21647ms, 21600ms
      // one of the traditional FWD ways of logging
      javaLoggerSystemErr();

      // 9235ms, 9062ms, 9523ms
      // never used in FWD
      javaLoggerFileSystem();

      // 13911ms, 13560ms
      // the new way
      centralLogger();
```

Source attached.

For the test CentralLoggerServer has not lost any formatting, just a few calls to get params for the context. Basically the whole server logging logic is preserved.

#58 - 06/21/2023 08:07 AM - Galya B

A little bit of history:

Once upon a time FWD used direct System.err.print and throwable.printStackTrace() statements excessively. It also used java.util.logging.Logger here and there with a ConsoleHandler that by default does System.err.print. server.log was "generated" by redirecting stderr to a file. Files were never generated by java.util.logging.Logger, nor all logs were logged by java.util.logging.Logger.

System.err compensated for java.util.logging.Logger bad performance giving a good result on average.

In that utopic environment CentralLogger appeared and ruined the performance of System.err by introducing file writes, but in reality it wanted to save FWD from java.util.logging.Logger poor performance.

#59 - 06/21/2023 08:09 AM - Constantin Asofiei

From your results, are you saying that Java logger is 33seconds and CentraLogger is 14seconds?

#60 - 06/21/2023 08:10 AM - Galya B

Constantin Asofiei wrote:

> From your results, are you saying that Java logger is 33seconds and CentraLogger is 14seconds?

Please read the story carefully and then we'll discuss my supposedly heretic statement again.

#61 - 06/21/2023 08:13 AM - Galya B

What I'm implying is: Console writes seem quite expensive, so the java logger performed poorly before, but the day was saved by the fact it wasn't used everywhere. CentralLogger is less performant than java logger in writing files, which unfortunately wasn't the way java logger was used. But I'll add more info in a few min, since I haven't tested java logger with our formatter.

#62 - 06/21/2023 08:34 AM - Constantin Asofiei

Galya, I think your tests are biased because they do not use the FWD runtime. My point was to test from within the FWD server. I've used the program I posted and:

- CentraLogger with console: 21037ms
- CentraLogger with file: 17032ms
- Java logging with stderr: 10649ms

Please spend your effort to find how to improve CentralLogger further. For example, can String.format be replaced with something else, which parses the format string only once? I understand that CentraLogger adds new features to logging which didn't exist before, and these add an overhead, but we must try to improve it.

#63 - 06/21/2023 08:41 AM - Galya B

Biased is a synonym for isolated? I have no idea what settings you use in your tests, while I shared my samples. Aren't then your tests biased?

I asked a while ago what percentage of improvement is expected from reworking the formatting. I haven't heard on it ever since.

#64 - 06/21/2023 08:48 AM - Galya B

Constantin Asofiei wrote:

> For example, can String.format be replaced with something else, which parses the format string only once?

Biased is the word I have for this statement. Let me prove you why.

java.util.logging.ConsoleHandler:

```
public class ConsoleHandler extends StreamHandler {
    // Private method to configure a ConsoleHandler from LogManager
    // properties and/or default values as specified in the class
    // javadoc.
    private void configure() {
        LogManager manager = LogManager.getLogManager();

        [..]

        setFormatter(manager.getFormatterProperty(cname +".formatter", new SimpleFormatter()));

        [..]
    }
```

java.util.logging.SimpleFormatter:

```
public class SimpleFormatter extends Formatter {

    [..]

    public synchronized String format(LogRecord record) {
        [..]

        String message = formatMessage(record);
        [..]

        return String.format(format,
                             dat,
                             source,
                             record.getLoggerName(),
                             record.getLevel().getLocalizedLevelName(),
                             message,
                             throwable);
    }
}
```

#65 - 06/21/2023 09:02 AM - Galya B

- File LoggingTest.java added

Here is the java logger to file with the new formatter. There is slight degradation of performance (+6.3%), because there is a lot more operations / data to be formatted, while the formatter itself is the same. This data is not very useful though, since this method (java logger to file) has never been used in FWD.

```
      // 10141, 9888
      // never used in FWD
      javaLoggerFileSystemOurFormatter();
```

So the rough data from the few runs can be summarized so (percentages are degradation compared to the previous method):

System.err (base) -> +50% System.err redirected to file -> +108% Java logger to file -> +6.3% Java logger to file with CentralLogFormatter -> +36% CentralLogger -> +58.8% Java logger to console redirected to a file -> +48% Java logger to console.

In red are methods used in FWD, in blue not used in FWD.

As you can see System.err is very low in the progression.

Instead of shooting blindly, this matter should be considered carefully and the actual issue (if there is any) found, before applying random "fixes".

At the end of the day, Greg may decide logging is not worth the performance degradation and ask me to convert all logs to System.err writes. This is an option. We should stay with open minds.

#66 - 06/21/2023 09:15 AM - Galya B

Another summary:

- CentralLogger (used in new trunk) is 300% slower than System.err redirected to file (used in old trunk).
- java.util.logging.Logger with ConsoleHandler redirected to a file (used in old trunk) is 480% slower than System.err redirected to file (used in old trunk).

Sorry for the edits.

#67 - 06/21/2023 11:00 AM - Greg Shah

- % Done changed from 100 to 50

- Status changed from Review to WIP

- Priority changed from Normal to Urgent

We know we need proper logging because there are 4GL language features that depend upon it. In addition, we must provide a supportable environment which means our log files must have common features like size limits and rotation. We aren't going to eliminate the logging or move back to System.err. There is no reason to consider it.

Independently of all this logging work, we have been making a substantial effort on performance optimization since last summer. This effort has succeeded in making consistent improvements in certain real 4GL converted code over that time period. The changes are substantial, with the current result being twice as fast as the original (in other words, more than 50% improvement in overall time). And we are not done, we need to make an additional improvement of another 25% (from the original number) over the coming months.

I fully understand that the older environment is different and was doing less work than the CentralLogger approach. With that said, we are watching and measuring this very carefully. The performance work was being tested in a branch slightly behind trunk. When we rebased that branch to trunk and picked up the logging changes, we now saw the impact of the new approach.

In regard to this task, it doesn't matter that this extra logging overhead is due to new code. The profiling done by Alexandru and Constantin have already identified areas that can be improved. This isn't random, it is based on profiling of real converted code running with these changes. Please investigate those areas and make improvements.

It is understandable that any new subsystem is going to have some code that needs optimization. That is what this task is about. This is pretty urgent because we don't want to introduce an apparent performance degradation by rebasing from trunk. If we can make significant improvements by making changes with formatting or whatever, let's please focus on getting that done. We are holding up releasing this rebased branch to a customer while this is worked.

#68 - 06/23/2023 01:53 AM - Galya B

If we blindly introduce "improvements" in iterations without any clearly expressed expectations, this may not work asap as requested.

#69 - 06/23/2023 02:02 AM - Galya B

Constantin Asofiei wrote:

> Galya, I think your tests are biased because they do not use the FWD runtime. My point was to test from within the FWD server. I've used the program I posted and:
>
> - CentraLogger with console: 21037ms
> - CentraLogger with file: 17032ms
> - Java logging with stderr: 10649ms

If you haven't yet understood, I'll do another summary:
stderr (+/- Java logging with stderr) < file (CentraLogger with file), aka 10649 < 17032
stderr (+/- Java logging with stderr) < stderr + file (CentraLogger with console), aka 10649 < (17032 + 10649) 21037

#70 - 06/23/2023 02:04 AM - Galya B

I'll test it in a unbiased way now, in FWD. Wait for results.

#71 - 06/23/2023 02:07 AM - Constantin Asofiei

Galya B wrote:

> If we blindly introduce "improvements" in iterations without any clearly expressed expectations, this may not work asap as requested.

The expectations are these:

- reduce the total number of allocated objects by CentralLogger during logging a message (string format comes into play here)
- reduce the overall total time - you can use a profiler in trace mode (VisualVM works).

Convert this logtest.p program:

```
def var i as int.

do i = 1 to 10000000:
   message i.
end.
```

and replace its .java with this content:

```
package com.goldencode.testcases;

import static com.goldencode.p2j.util.BlockManager.externalProcedure;

import java.util.logging.Level;
import java.util.logging.Logger;

import com.goldencode.p2j.util.Block;
import com.goldencode.p2j.util.Body;
import com.goldencode.p2j.util.InternalEntry.Type;
import com.goldencode.p2j.util.LegacySignature;
import com.goldencode.p2j.util.logging.CentralLogger;

public class Logtest
{
   /** Logger */
   private static final CentralLogger LOG1 = CentralLogger.get(Logtest.class);

   private static final Logger LOG2 = Logger.getLogger(Logtest.class.getName());

   @LegacySignature(type = Type.MAIN, name = "logtest.p")
   public void execute()
   {
      externalProcedure(Logtest.this, new Block((Body) () ->
      {
         long n1 = System.nanoTime();
         for (int i = 0; i < 1_000_000; i++)
         {
            LOG1.log(Level.WARNING, "something");
            // LOG2.log(Level.WARNING, "something");
         }
         long n2 = System.nanoTime();
         System.out.println((n2 - n1) / 1_000_000l);
      }));
   }
}
```

Run it from the FWD server via the FWD client using logtest.p program. Uncomment and run the LOG2 line separately. We are interested in profiling only the LOG1 (CentraLogger), LOG2 (java logging) is only to get a 'reference time' to which we need to get close.

#72 - 06/23/2023 02:56 AM - Galya B

- File StandardServer.java added

CleanFormatter and the java logger init code is taken from an old trunk version.

```
public class StandardServer
{

   [..]

   public void bootstrap(BootstrapConfig config, Supplier<String> diagnostics, String cfgProfile)
   throws Exception
   {
      Runtime.getRuntime().addShutdownHook(new Thread(() -> {

         [..]

      }));

      long n1 = System.nanoTime();
      for (int i = 0; i < 1_000_000; i++)
      {
         LOG.log(Level.WARNING, "something");
         // LOG2.log(Level.WARNING, "something");
      }
      long n2 = System.nanoTime();
      System.out.println("centralLogger " + (n2 - n1) / 1_000_000L);

      Logger rootLogger = Logger.getLogger("");
      ConsoleHandler consoleHandler = new ConsoleHandler();
      consoleHandler.setFormatter(new CleanFormatter());
      rootLogger.addHandler(consoleHandler);

      Logger anotherLogger = Logger.getLogger("random");

      n1 = System.nanoTime();
      for (int i = 0; i < 1_000_000; i++)
      {
         anotherLogger.log(Level.WARNING, "something");
      }
      n2 = System.nanoTime();
      System.out.println("java console logger " + (n2 - n1) / 1_000_000L);

      try
      {
         // wake up any waiting threads, the non-network portions of the
         // server are fully initialized now
         ready.countDown();

         [..]

      }
   }

   /**
    * Simple formatter to emit a very clean, condensed output.
    */
   public static class CleanFormatter
   extends Formatter
   {
      /** Date format string. */
      private static final String FMT = "MM/dd/yyyy HH:mm:ss z";

      /** Date and time formatter. */
      private static final ThreadLocal<SimpleDateFormat> sdf = new ThreadLocal<SimpleDateFormat>()
      {
         @Override
         protected SimpleDateFormat initialValue()
         {
            return new SimpleDateFormat(FMT);
         }
      };
```

```
    /** Platform-specific line separator. */
    private static final String SEP = System.getProperty("line.separator");

    /**
     * Format the given log record and return the formatted string.
     *
     * @param    record
     *           The log record to be formatted.
     *
     * @return   The formatted log record.
     */
    public String format(LogRecord record)
    {
       String spec = "[%s] (%s:%s) %s%s";

       Date dat = new Date(record.getMillis());

       Object[] parms = new Object[]
       {
          sdf.get().format(dat),
          record.getSourceClassName(),
          record.getLevel().toString(),
          record.getMessage(),
          SEP
       };

       String out = String.format(spec, parms);

       Throwable t = record.getThrown();

       if (t != null)
       {
          StringBuilder sb = new StringBuilder(out);
          dumpThrowable(t, sb);
          out = sb.toString();
       }

       return out;
    }

    /**
     * Dump all details of the throwable into the given string buffer.
     * Includes exception type, message, complete stack trace, and chained
     * throwables, if any.
     *
     * @param    t
     *           The throwable to dump.
     * @param    buf
     *           String buffer into which details are written.
     */
    private void dumpThrowable(Throwable t, StringBuilder buf)
    {
       Throwable cause = t;
       do
       {
          if (cause != t)
          {
             buf.append("Caused by: ");
          }

          buf.append(cause.getClass().getName());
          String msg = cause.getMessage();
          if (msg != null && msg.trim().length() > 0)
          {
             buf.append(": ");
             buf.append(msg);
          }
          buf.append(SEP);

          StackTraceElement[] trace = cause.getStackTrace();

          for (int i = 0; i < trace.length; i++)
          {
             buf.append("        at ");
             buf.append(trace[i]);
```

```
                buf.append(SEP);
            }

            cause = cause.getCause();
        }
        while (cause != null);
    }
  }
}
```

centralLogger 15329
java console logger 21277

I don't mind working on improving the performance of CentralLogger, but doing tacky wacky stuff (like reinventing String.format) because of unrealistic expectations is what concerns me.

I hope numbers speak for themselves.

Also I want you to understand that we don't have base performance measurements to compare the new state, because previously it didn't use any logger.+

That being said... I'm on it... whatever it is... the issue... the improper logging.

#73 - 06/23/2023 05:11 AM - Galya B

r14632 16% improvement of the +0.45% degradation by replacing String.format in formatting logging context.

I also tried to replace SimpleDateFormat with DateTimeFormatter (popular as a better implementation), but it led to worse numbers.
I also tried to move around FileHandler.flush() which unexpectedly didn't lead to any change in numbers.

What else?

#74 - 06/23/2023 05:14 AM - Constantin Asofiei

Please do a profiling and see what else stands out which can be improved.

#75 - 06/23/2023 08:51 AM - Galya B

r14633:

The appserver kept the spawner live, which kept the stderr parsing / logging loop on the server live. It probably contributed to the numbers, but I'm not sure how much exactly. To stop it the spawner now sends a message before launching the client process, that will trigger a timer and stop the logging

after 2min. The client is supposed to log itself, so the stderr forwarding should not be relevant after the client is started. It seems winspawn.c has some retry logic and I'm not sure if it works well there, but it is tested and works for Linux.

#76 - 06/23/2023 08:52 AM - Galya B

If it's not a trouble, can we have a perf test running now, while I wonder what else to change.

P.S. with updated spawner

#77 - 06/23/2023 10:26 AM - Galya B

- % Done changed from 50 to 100

- Status changed from WIP to Review

r14634: Fixed Java (java.util.logging.LogRecord#needToInferCaller set with reflection to stop heavy checks unused by CentralLogger).

Performance of CentralLogger is 3+ times quicker now.

After all changes the function that initially took ~14510ms now takes ~4200ms.

#78 - 06/23/2023 11:42 AM - Constantin Asofiei

Galya, this is a review for 7443a rev 14634:

- CentraLogger - please leave only 006 number and remove 007 and 008 (leave the user initials, date and message there, for both '007' and '008' lines; remove just the numbers, we need to have only one history number per branch). The same for CentralLoggerServer.
- CentraLogger.logStream
  - I don't really like that we have this if (line.equals("started")) condition from the spawner. It seems fragile. At least add comments to both the native spawner code and here that they are linked (i.e. changing in one place it must be changed in the other place, too).
  - the 'self distruct' timeout I think should be the same as ClientSpawner.TIME_OUT.
  - what is the purpose of TimeUnit.MILLISECONDS.sleep(0);? 0 just does nothing.
- CentraLogger.invalidateLoggerTreeLevelCache - shouldn't logger.loggerName.contains(loggerName) be logger.loggerName.startsWith(loggerName)? AFAIK the logger hierarchy is log1 is parent for log1.log2.log3, but not for log2.log1.
- LoggingUtil.addLeadingSymbols - instead of doing:

```
new String(new char[leadingSymbolsCount]).replace('\0', symbol);
```

  try Arrays.fill(char[], symbol), to avoid String.replace overhead.

Eugenie: please review the native changes.

Alexandru: please do another round of performance testing with 7156b + 7443a. Please note that I've rebased 7156b.

#79 - 06/23/2023 11:52 AM - Constantin Asofiei

Constantin Asofiei wrote:

> Alexandru: please do another round of performance testing with 7156b + 7443a. Please note that I've rebased 7156b.

With this in mind, please use this patch (the app is not yet reconverted with rebased 7156b):

```
=== modified file 'src/com/goldencode/p2j/util/BlockManager.java'
--- old/src/com/goldencode/p2j/util/BlockManager.java   2023-06-15 09:30:19 +0000
+++ new/src/com/goldencode/p2j/util/BlockManager.java   2023-06-23 14:25:16 +0000
@@ -7097,7 +7097,19 @@

       throw lex;
    }
-
+      /**
+    * Trigger a rollback of the current block and then throw a legacy condition.
+    *
+    * @param    label
+    *           The name of the block that is the target of the undo and throw operations.
+    * @param    error
+    *           The legacy error to throw.
+    */
+   public static void undoThrow(String label, object<? extends LegacyError> error)
+   {
+      undoThrow(error);
+   }
+
    /**
     * Trigger a rollback of the current block and then throw a legacy condition.
     *
```

#80 - 06/23/2023 12:33 PM - Eugenie Lyzenko

Constantin Asofiei wrote:

> Eugenie: please review the native changes.

I have no objection for code itself. However the comment log in file history confuses me a bit:

```
** 018 GBB 20230623 Adding "started" print to stderr to notify the server logger to stop listening for logs.
```

I see the log call used when process **started**, not stopped. So it is not clear for me why words **...to stop listening...** are used here.

#81 - 06/26/2023 02:29 AM - Galya B

Eugenie Lyzenko wrote:

> I see the log call used when process **started**, not stopped. So it is not clear for me why words **...to stop listening...** are used here.

The loop in the listening Thread created by CentralLogger#logStream stops work when isProcessRunning is false, which happens on receiving the keyword started from the spawner.
Have I missed to cleanup something else?

#82 - 06/26/2023 02:44 AM - Galya B

Eugenie Lyzenko wrote:

> I see the log call used when process **started**, not stopped. So it is not clear for me why words **...to stop listening...** are used here.

Ah, I got it. The problem is that the message can be send only before the client process is started and it's indicating the process is started, while the logger Thread decides to stop listen on this event, because client processes like appserver are keeping the logging loop alive for unnecessarily long time otherwise.

P.S. It's not like command from the spawner to the listener, but an event.

#83 - 06/26/2023 03:13 AM - Galya B

Constantin Asofiei wrote:

> we need to have only one history number per branch

You say that often, but when I add headers for each change (for code reviews), I forget the previous one was on the same branch. Thanks for pointing out the needed changes.

- I don't really like that we have this if (line.equals("started")) condition from the spawner.

I don't too. The alternative solution is to separate spawner logs in their own files, but from usability perspective it might be better to keep this imperfect solution.

> At least add comments to both the native spawner code and here that they are linked (i.e. changing in one place it must be changed in the other place, too).

OK, added.

- the 'self distruct' timeout I think should be the same as ClientSpawner.TIME_OUT.

I lowered it to ClientSpawner.TIME_OUT which is 30 sec. The reason for the 2min was that I'm not sure if less will be enough for the retry in winspawn.c. I don't understand the retry there, because appserver launcher has its own retry mechanism in Java.

- what is the purpose of TimeUnit.MILLISECONDS.sleep(0);? 0 just does nothing.

Leftover, removing it.

- CentraLogger.invalidateLoggerTreeLevelCache - shouldn't logger.loggerName.contains(loggerName) be logger.loggerName.startsWith(loggerName)?

Right.

> try Arrays.fill(char[], symbol), to avoid String.replace overhead.

Done.

r14635 up.

#84 - 06/26/2023 03:32 AM - Constantin Asofiei

Galya, one more small optimization, please: this test in LoggingUtil.addLeadingSymbols can be leadingSymbolsCount <= 0:

```
      if (leadingSymbolsCount < 0)
      {
         return originalValue;
      }
```

#85 - 06/26/2023 03:35 AM - Galya B

Constantin Asofiei wrote:

> Galya, one more small optimization, please: this test in LoggingUtil.addLeadingSymbols can be leadingSymbolsCount <= 0:

Up in r14636.

#86 - 06/26/2023 04:47 AM - Alexandru Lungu

Attempting to profile with rev. 14636. I've got:

```
[ant:javac] /media/al2/gcd/branches/7443a/src/com/goldencode/p2j/util/logging/CentralLogger.java:393: error: e
xception InterruptedException is never thrown in body of corresponding try statement
[ant:javac]             catch (InterruptedException ignored)
```

#87 - 06/26/2023 06:12 AM - Galya B

Compilation error fixed with r14637.

#88 - 06/26/2023 10:11 AM - Constantin Asofiei

Greg: I think we can merge this to trunk. Alexandru reported a 3.5% improvement for 7443a (from a 3.5% loss without it).

#89 - 06/26/2023 10:56 AM - Greg Shah

Please merge 7443a to trunk. Good work!

#90 - 06/26/2023 12:29 PM - Galya B

Task branch 7443a was merged to trunk as rev 14633 and archived.

#91 - 06/26/2023 12:52 PM - Galya B

- Status changed from Review to Test

#92 - 06/26/2023 01:22 PM - Greg Shah

- Status changed from Test to Closed

#93 - 07/14/2023 04:43 AM - Constantin Asofiei

Constantin Asofiei wrote:

> Greg: I think we can merge this to trunk. Alexandru reported a 3.5% improvement for 7443a (from a 3.5% loss without it).

This is just to document some findings in the last two days: in trunk rev 14629, there was a bug introduced which leaked a RETURN statement to the caller block - thus the caller block was returning early, and skipping application code which should have been executed; it's weird that this did not appear in normal regression tests like ETF. This bug was related to #7300 (7300a branch).

In trunk rev 14649, for #7450, this bug was partially fixed; with this fix, there was a performance loss in an application, because this 'skipped application code' was being executed (correctly) again. It took me a while to find this explanation, and the conclusion was the changes in trunk rev 14629 as the problem. The full fix will be merged via 7300b.

I've profiled again the application and the conclusion is this:

- the previous '3.5% additional improvement from the 3.5% loss' was incorrect and was due to the 7300a bug
- there are no more spikes for CentraLogger or LOG-MANAGER usage; so we are 'on par' (or better) with the previous Java logging

#94 - 01/25/2024 03:05 AM - Tomasz Domin

Galya B wrote:

> r14634: Fixed Java (java.util.logging.LogRecord#needToInferCaller set with reflection to stop heavy checks unused by CentralLogger).
>
> Performance of CentralLogger is 3+ times quicker now.
>
> After all changes the function that initially took ~14510ms now takes ~4200ms.

May I ask why such decision was taken instead of creating custom LogRecord that does not do inferCaller as JDK LogRecord does ?
Setting private field of LogRecord violated access constraints and causes issues starting from JDK 17+.
Was is to minimize code refactoring ?
EDIT: I found CentralLogRecord.

#95 - 01/25/2024 03:08 AM - Tomasz Domin

- Related to Feature #6692: move FWD to Java 17 added

#96 - 01/25/2024 03:11 AM - Galya B

Tomasz Domin wrote:

> Galya B wrote:
>
> > r14634: Fixed Java (java.util.logging.LogRecord#needToInferCaller set with reflection to stop heavy checks unused by CentralLogger).
> >
> > Performance of CentralLogger is 3+ times quicker now.
> >
> > After all changes the function that initially took ~14510ms now takes ~4200ms.
>
> May I ask why such decision was taken instead of creating custom LogRecord that does not do inferCaller as JDK LogRecord does ?
> Setting private field of LogRecord violated access constraints and causes issues starting from JDK 17+.

Replacing "simply" LogRecord means implementing a new FileHandler. This is work that is to be scheduled with upgrading to a new version of Java. It's been discussed. I can show references, if any concerns.

#97 - 01/25/2024 03:20 AM - Tomasz Domin

Galya B wrote:

> Tomasz Domin wrote:
>
> > Galya B wrote:
> >
> > > r14634: Fixed Java (java.util.logging.LogRecord#needToInferCaller set with reflection to stop heavy checks unused by CentralLogger).
> > >
> > > Performance of CentralLogger is 3+ times quicker now.
> > >
> > > After all changes the function that initially took ~14510ms now takes ~4200ms.
> >
> > May I ask why such decision was taken instead of creating custom LogRecord that does not do inferCaller as JDK LogRecord does ?
> > Setting private field of LogRecord violated access constraints and causes issues starting from JDK 17+.
>
> Replacing "simply" LogRecord means implementing a new FileHandler. This is work that is to be scheduled with upgrading to a new version of Java. It's been discussed. I can show references, if any concerns.

Please do, I am in a process of upgrading FWD to support new Java versions, probably that needs to be done as this is the only stopper in entire FWD now.

#98 - 01/25/2024 03:52 AM - Galya B

#7143-253, #7143-255.

Instead of replacing java.util.logging.LogRecord there could be another option, extend it and Override getSourceClassName and getSourceMethodName to return the values without calling inferCaller. Then replace new LogRecord with the new object in CentralLogger and CentralLogRecord. Potentially even in the SQL loggers.

And finally the block in CentralLogFormatter can be removed:

```
try
      {
         Field f1 = record.getClass().getDeclaredField("needToInferCaller");
         f1.setAccessible(true);
         f1.set(record, false);
      }
      catch (NoSuchFieldException | IllegalAccessException ignored)
      {
      }
```

Then performance tests need to be run again.

#99 - 01/25/2024 03:57 AM - Galya B

The new class should in be in the package com.goldencode.p2j.util.logging. As for the name (since CentralLogRecord is already used), it can be PerformantLogRecord or ExtendedLogRecord.

#100 - 01/26/2024 03:51 AM - Tomasz Domin

Galya B wrote:

> The new class should in be in the package com.goldencode.p2j.util.logging. As for the name (since CentralLogRecord is already used), it can be PerformantLogRecord or ExtendedLogRecord.

Thank you Galya very much.

I was considering that option but it seems java.util.logging.LogRecord was not really meant for extending it as it has all fields private so overriding class cant access nor modify them directly.

Have you considered calling setSourceClassName and setSourceMethodName unconditionally (even when they are set to null) in ultimateLog ? That would clear needToInferCaller flag.

Another possible solution might be to make use of serialization and generate log message on LogRecord serialization, but there would be some overhead related to that.

I guess I will try both.

#101 - 01/26/2024 05:12 AM - Galya B

Tomasz Domin wrote:

> Galya B wrote:
> I was considering that option but it seems java.util.logging.LogRecord was not really meant for extending it as it has all fields private so overriding class cant access nor modify them directly.

The field needToInferCaller is private indeed, but it doesn't have to be overridden itself, if the methods that use it are overridden. Please try the method described in #7443-98.

#102 - 01/26/2024 05:28 AM - Tomasz Domin

Galya B wrote:

> Tomasz Domin wrote:
>
> > Galya B wrote:
> > I was considering that option but it seems java.util.logging.LogRecord was not really meant for extending it as it has all fields private so overriding class cant access nor modify them directly.
>
> The field needToInferCaller is private indeed, but it doesn't have to be overridden itself, if the methods that use it are overridden. Please try the method described in #7443-98.

The issue is not only needToInferCaller being private but also all other java.util.logging.LogRecord fields being private, so there is no way to read them except calling superclass getters. So simple overriding of getSourceClassName wont let access overriding class to access private String sourceMethodName, it still needs to get it via super.getSourceClassName(), which still will check needToInferCaller.
I could override private fields with superclass fields, but that would lead to loosing all benefits of using a standard java.util.logging.LogRecord, most probably also issues with serialization for distributed use cases.

#103 - 01/26/2024 05:31 AM - Galya B

Tomasz Domin wrote:

> Galya B wrote:
>
> > Tomasz Domin wrote:
> >
> > > Galya B wrote:
> > > I was considering that option but it seems java.util.logging.LogRecord was not really meant for extending it as it has all fields private so overriding class cant access nor modify them directly.
> >
> > The field needToInferCaller is private indeed, but it doesn't have to be overridden itself, if the methods that use it are overridden. Please try the method described in #7443-98.
>
> The issue is not only needToInferCaller being private but also all other java.util.logging.LogRecord fields being private, so there is no way to read them except calling superclass getters. So simple overriding of getSourceClassName wont let access overriding class to access private String sourceMethodName, it still needs to get it via super.getSourceClassName(), which still will check needToInferCaller.
> I could override private fields with superclass fields, but that would lead to loosing all benefits of using a standard java.util.logging.LogRecord, most probably also issues with serialization for distributed use cases.

We'll be overriding only those two methods, the rest can continue working as it was before. But you're right that we'll also need setters for sourceMethodName and sourceClassName and declaration of the two fields in the new class.

#104 - 01/31/2024 05:36 AM - Tomasz Domin

- File 6692a_inferCaller_fix.diff added

Galya B wrote:

> We'll be overriding only those two methods, the rest can continue working as it was before. But you're right that we'll also need setters for sourceMethodName and sourceClassName and declaration of the two fields in the new class.

I took a different approach skipping all overriding.
As calling to LogRecord.setSourceClassName or LogRecord.setSourceMethodName implicitly clears inferLogger flag I allowed it to be set even when it would be set to NULL - except a case where logging level is ALL - in case someone would be interested in source class/method names in logs despite its cost.

Having look at logs dont see any difference.

Attached a patch that will be part 6692a branch.

#105 - 01/31/2024 06:01 AM - Galya B

Tomasz Domin wrote:

> As calling to LogRecord.setSourceClassName or LogRecord.setSourceMethodName implicitly clears inferLogger flag I allowed it to be set even when it would be set to NULL

Nice find. The diff looks good. Now the reflection can be removed from CentralLogFormatter.

Files

backtraces.png	158 KB	06/15/2023	Constantin Asofiei
callees.png	55 KB	06/15/2023	Constantin Asofiei
isLoggable.png	110 KB	06/15/2023	Constantin Asofiei
log_format.png	509 KB	06/20/2023	Constantin Asofiei
logging-performance.zip	76.4 KB	06/21/2023	Galya B
LoggingTest.java	3.79 KB	06/21/2023	Galya B
StandardServer.java	89.9 KB	06/23/2023	Galya B
6692a_inferCaller_fix.diff	1.15 KB	01/31/2024	Tomasz Domin