Bugs - Bug #2483

Fix issues identified by static code analysis

01/09/2015 09:02 AM - Igor Skornyakov

Status:	Closed	**Start date:**	01/09/2015
Priority:	Normal	**Due date:**	
Assignee:	Igor Skornyakov	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:	Cleanup and Stablization for Server Features		
billable:	No	**case_num:**	
vendor_id:	GCD	**version:**	

Description

Related issues:

Related to FWD - Bug #3002: static code analysis issues	**Closed**

History

#1 - 01/09/2015 10:04 AM - Greg Shah

- Project changed from Liberty to Bugs

#2 - 01/09/2015 10:10 AM - Igor Skornyakov

- Description updated

- Subject changed from Bugs project to Fix issues identified by static code analysis

#3 - 01/09/2015 10:16 AM - Igor Skornyakov

1. Eclipse reports potential NPE at line 1216 in the DirectoryCopy.java
2. FindBugs reports 250+ suspicious places in P2J

#4 - 01/09/2015 10:17 AM - Igor Skornyakov

1. DirectoryCopy, line 1216
if (att != null || att.length > 0)
Potential NPE. I think a correct version is: if (att != null && att.length > 0)

2. AppServerManager, line 837
appServers.remove(pool);
Argument type (AgentPool) is incompatible with the map key type (String).
A suggested change:
appServers.remove(pool.getName());

3. Bad attempt to compute absolute value of signed random integer in
com.goldencode.p2j.directory.DirectoryService.closeBatchInt(DirectoryService$BatchRef, boolean, boolean) line 4728
Math.abs(new Random().nextInt())
Will not produce a negative value if new Random().nextInt() == Integer.MIN_VALUE.
A suggested change:
new Random().nextInt(Integer.MAX_VALUE)

4. Call to com.goldencode.p2j.schema.SchemaConfig$Metadata.equals(String) in com.goldencode.p2j.schema.SchemaConfig.getSchemas(boolean)
[Scariest(1), High confidence] SchemaConfig.java line 125
if (metadata.equals(next))
A suggested change:
if (metadata.getName().equals(next))

5. Call to equals(null) in com.goldencode.p2j.convert.ExpressionConversionWorker.expressionType(Aast, boolean, boolean) line 1047
if ("date".equals(stype) ||
stype can only be null at this point
A suggested change: initialiaze stype as in case of PLUS above (?)

6 Call to method of static java.text.DateFormat in com.goldencode.p2j.directory.DateValue.toString() line 258.
format.format(getDate(), buf, new FieldPosition(DateFormat.YEAR_FIELD));
format is a static instance of DateFormat which is not thread-safe.

A suggested change: use ThreadLocal or make all methods using these field synchronized.
The same issue in TimeValue, LogHelper.CleanFormatter

7. A broken implementation of Object.equals() in a number of places.
The most common error is not checking for null"
com.goldencode.cache.LFUAgingCache$NodeImpl
com.goldencode.cache.LRUCache$NodeImpl
com.goldencode.expr.Expression$CacheKey
com.goldencode.expr.SymbolResolver$CacheKey
...
Suggested fix - add check for null (or, even better for instanceof).

Also - the FloatConstant.equals() implementation violates the contract making equals non-transitive.

Also - DateValue.equals() and TimeValue.equals() allow compare with instances of Calendar and Date respectively which is risky as makes equals realtion non-symmetric (contract violation).

8. Broken synchronization:
com.goldencode.p2j.net.Queue.getNodeAddress() is unsynchronized, com.goldencode.p2j.net.Queue.setNodeAddress(int) is synchronized.

com.goldencode.p2j.net.Queue.getRemoteAddress() is unsynchronized, com.goldencode.p2j.net.Queue.setRemoteAddress(int) is synchronized.

com.goldencode.p2j.security.SecurityManager.getDebugLevel() is unsynchronized,
com.goldencode.p2j.security.SecurityManager.setDebugLevel(Level) is synchronized.

9.com.goldencode.p2j.ui.ColorRgb is Externalizable but doesn't define a void constructor.

It is possible to have Externalizable class w/o void constructor but it requires some additional efforts (with auxiliary class and readResolve/writeReplace - typically this trick is used for classes with final filelds). ColorRgb does not implement readResolve/writeReplace. The simplest solution is to add void constructor.

#5 - 01/09/2015 10:18 AM - Igor Skornyakov

- File ias_upd20150109a.zip added

I've fixed the issues we discussed recently except the one regarding the ExpressionConversionWorker (I understand it is unclear how to deal with it).

I've removed the inconsistent synchronized clause in the com.goldencode.p2j.net.Queue as the field it protects is volatile.

In the com.goldencode.p2j.security.SecurityManager I've added ReentrantReadWriteLock to guard access to debugLevel and debugLevelRequest fields (this imposes less contention comparing to a regular monitor).

#6 - 01/09/2015 10:28 AM - Igor Skornyakov

TableResultSet.java, line 603.
((BigDecimal) res).setScale(scale);
Looks like a misprint, should be: res = ((BigDecimal) res).setScale(scale);

#7 - 01/09/2015 10:31 AM - Igor Skornyakov

/home/ias/p2j/src/com/goldencode/p2j/e4gl/E4GLPreprocessor.java:307 Self assignment of field E4GLPreprocessor.opts in com.goldencode.p2j.e4gl.E4GLPreprocessor.setOptions(Options)
public void setOptions(Options options) {
this.opts = opts;
}

Should be: this.opts = options;

#8 - 01/09/2015 10:36 AM - Igor Skornyakov

/home/ias/p2j/src/com/goldencode/p2j/ui/WindowWidget.java:276 There is an apparent infinite recursive loop in com.goldencode.p2j.ui.WindowWidget.setMaxWidth(NumberType).

```
@Override
   public void setMaxWidth(NumberType maxWidth)
{
      this.setMaxWidth(maxWidth);
   }
```

Should be: this.setMaxWidthChars(maxWidth); (?)

/home/ias/p2j/src/com/goldencode/p2j/util/SocketImpl.java:1335 There is an apparent infinite recursive loop in com.goldencode.p2j.util.SocketImpl.setSocketOption(String, character).

```
public logical setSocketOption(String optionName, character optionVal)
{
      return setSocketOption(optionName, new character(optionVal));
   }
```

Should be (?)

#9 - 01/09/2015 10:48 AM - Igor Skornyakov

/home/ias/p2j/src/com/goldencode/p2j/persist/hql/HQLParser.java:147 Incorrect lazy initialization and update of static field com.goldencode.p2j.persist.hql.HQLParser.tokenLookup in com.goldencode.p2j.persist.hql.HQLParser.lookupTokenType(String).

The tokenLookup initialization is broken for a number of reasons. If we really want to initialize it lazily it is better to use the lazy initialization holder class idiom (see. "Effective Java", 2nd edition, page 283)

A similar issue: /home/ias/p2j/src/com/goldencode/p2j/util/LogHelper.java:557 Incorrect lazy initialization and update of static field com.goldencode.p2j.util.LogHelper.fileHandler in com.goldencode.p2j.util.LogHelper.initWorker(Class)

#10 - 01/09/2015 10:56 AM - Igor Skornyakov

/home/ias/p2j/src/com/goldencode/p2j/uast/CallbackResolver.java:355 Possible null pointer dereference of method in com.goldencode.p2j.uast.CallbackResolver.invokeCallback(String, Class, Class[], Object[])

The statement
return method.invoke(cb.getTargetInstance(), args);
will be invoked even if method null

Suggested fix:
return method null ? null : method.invoke(cb.getTargetInstance(), args);

#11 - 01/09/2015 10:58 AM - Igor Skornyakov

/home/ias/p2j/src/com/goldencode/p2j/persist/type/DatetimeTzUserType.java:239 Possible null pointer dereference of value in com.goldencode.p2j.persist.type.DatetimeTzUserType.nullSafeSet(PreparedStatement, Object, int, SessionImplementor)

Suggested fix:

```
if (value == null)
{
          st.setObject(index, null);
          st.setObject(index + 1, null);
          return;
      }
```

#12 - 01/09/2015 11:01 AM - Igor Skornyakov

/home/ias/p2j/src/com/goldencode/p2j/util/integer.java:359 Possible null pointer dereference of value in new com.goldencode.p2j.util.integer(Boolean)

The problem is with super.setValue(value) as in superclass setValue() accepts boolean. The unboxing can cause NPE.

Suggested fix: (?)

#13 - 01/09/2015 11:40 AM - Igor Skornyakov

/home/ias/p2j/src/com/goldencode/p2j/util/EnvironmentOps.java:1313 Non-virtual method call in com.goldencode.p2j.util.EnvironmentOps.load(character, character, boolean) passes null for nonnull parameter of load(character, character, logical, character).

/home/ias/p2j/src/com/goldencode/p2j/util/EnvironmentOps.java:1105 Non-virtual method call in com.goldencode.p2j.util.EnvironmentOps.load(character) passes null for nonnull parameter of load(character, character, logical, character)

/home/ias/p2j/src/com/goldencode/p2j/util/EnvironmentOps.java:1121 Non-virtual method call in com.goldencode.p2j.util.EnvironmentOps.load(String) passes null for nonnull parameter of load(character, character, logical, character)

/home/ias/p2j/src/com/goldencode/p2j/util/EnvironmentOps.java:1335 Non-virtual method call in com.goldencode.p2j.util.EnvironmentOps.load(String, character, boolean) passes null for nonnull parameter of load(character, character,

/home/ias/p2j/src/com/goldencode/p2j/util/EnvironmentOps.java:1357 Non-virtual method call in com.goldencode.p2j.util.EnvironmentOps.load(character, String, boolean) passes null for nonnull parameter of load(character, character, logical, character)

/home/ias/p2j/src/com/goldencode/p2j/util/EnvironmentOps.java:1140 Non-virtual method call in com.goldencode.p2j.util.EnvironmentOps.load(character, boolean) passes null for nonnull parameter of load(character, character, logical, character).

/home/ias/p2j/src/com/goldencode/p2j/util/EnvironmentOps.java:1379 Non-virtual method call in com.goldencode.p2j.util.EnvironmentOps.load(String, String, boolean) passes null for nonnull parameter of load(character, character, logical, character).

/home/ias/p2j/src/com/goldencode/p2j/util/EnvironmentOps.java:1159 Non-virtual method call in com.goldencode.p2j.util.EnvironmentOps.load(String, boolean) passes null for nonnull parameter of load(character, character, logical, character)

The above mentioned calls explicitly provide null as last argument (baseKey). This argument is de-referenced in the invoked static method w/o check.

Suggested fix: (?)

#14 - 03/23/2016 11:25 PM - Eric Faulhaber

Igor, I'm not sure if ias_upd20150109a.zip was ever integrated into P2J. If not, please integrate these changes into task branch 2483a now. Also go ahead and make the fixes you've suggested for the additional items you detailed above. One exception: HQLParser.java (note 9) is generated by ANTLR, so it doesn't make sense to edit that file.

Now that you have greater familiarity with the P2J code, do you have any suggested fixes for the items above which you were not sure how to address before?

#15 - 03/24/2016 07:40 AM - Igor Skornyakov

Created task branch 2483a

#16 - 03/28/2016 04:51 PM - Eric Faulhaber

- Target version set to Milestone 11

#17 - 03/30/2016 06:54 PM - Igor Skornyakov

I've fixed a number of issues(and some additional ones). Will commit the changes tomorrow morning after adding history records.
I've also noticed a number of issues which do not have obvious fix:

1. TemporaryAccountPool uses synchronization on the a pool variable which is ConcurrentLinkedQueue.
2. RouterSessionManager.deregisterSession: checks queue variable for null after a synchronization block on it
3. LFUAgingCache.NodeImpl: the hits field is modified in the access() method which is invoked from the superclass constructor
4. FloatConstant.equals - non-transitive. The idea is understandable but the implementation obviously violates the contract. In addition there are multiple places where float point number are compared for equality. I think we need a consistent approach for this. For example we can use the approach similar to one in the java.lang.Float.equals() possibly ignoring
5. AclRow.equals - non symmetric e.g. when comparing AclRow instance with other non AclRow TaggedName.
6. AdminLogon - non safe assignment to logon static field; assignment to an argument of the terminate() method - it is a noop but a comment claims that it destroys session
7. MultiClassLoader - wrong idiom for the managerLibsDir lazy initialization
8. DirectoryService - incorrect lazy initialization of the LOG field
9. LockManager - incorrect while/wait combination in the enterLock method (line 131); incorrect lazy initialization of the LOG field
10. E4GLPrepocessor.setOptions: currently is a noop. Should it be this.opts = options?
11. StreamConnector.close() - wait() call not in a loop.
12. ClientSpawner.spawn - not checking return value of the latch.await(TIME_OUT, TimeUnit.SECONDS). Is it correct?
13. ReportWorker.addMatchMultiplexed - String.replaceAll accepts regexp as first argument but File.separator is used
14. TempTableConnection - incorrect lazy initialization of delegate field
15. GlobalEventManager - possible NPE in the updateEntityRefCount (if count == null && delta <= 0)
16. HQLParser - incorrect lazy initialization of the tokenLookup;

#18 - 03/31/2016 12:00 AM - Eric Faulhaber

Igor Skornyakov wrote:

> TemporaryAccountPool uses synchronization on the a pool variable which is ConcurrentLinkedQueue.

Don't know. Greg?

> RouterSessionManager.deregisterSession: checks queue variable for null after a synchronization block on it

This is in an assert, and we don't normally run with asserts enabled. I guess it doesn't hurt to keep this; I would move it up just under the line which defines queue.

> LFUAgingCache.NodeImpl: the hits field is modified in the access() method which is invoked from the superclass constructor

Intentional; please leave as is.

> FloatConstant.equals - non-transitive. The idea is understandable but the implementation obviously violates the contract. In addition there are multiple places where float point number are compared for equality. I think we need a consistent approach for this. For example we can use the approach similar to one in the java.lang.Float.equals() possibly ignoring

The "equality" comparisons are not direct value comparisons, but rather they are epsilon tests. Perhaps not the best solution, admittedly. Anyway, this is an old library that was repurposed to back TRPL, where I don't think we really use floating point values. I would leave this as is.

> AclRow.equals - non symmetric e.g. when comparing AclRow instance with other non AclRow TaggedName.
> AdminLogon - non safe assignment to logon static field; assignment to an argument of the terminate() method - it is a noop but a comment claims that it destroys session
> MultiClassLoader - wrong idiom for the managerLibsDir lazy initialization
> DirectoryService - incorrect lazy initialization of the LOG field
> LockManager - incorrect while/wait combination in the enterLock method (line 131); incorrect lazy initialization of the LOG field
> E4GLPrepocessor.setOptions: currently is a noop. Should it be this.opts = options?
> StreamConnector.close() - wait() call not in a loop.
> ClientSpawner.spawn - not checking return value of the latch.await(TIME_OUT, TimeUnit.SECONDS). Is it correct?
> ReportWorker.addMatchMultiplexed - String.replaceAll accepts regexp as first argument but File.separator is used

Not my areas of the code; anyone else have input here?

> TempTableConnection - incorrect lazy initialization of delegate field

IIRC, only one instance of TempTableConnectionProvider is ever created, so this has been safe so far. Also, I intend to replace the use of this class soon. Nevertheless, I see your point. Looks like delegate could be an instance variable.

> GlobalEventManager - possible NPE in the updateEntityRefCount (if count == null && delta <= 0)

Yes. Suggested fix:

```
if (count == null)
{
   if (delta <= 0)
   {
      throw new IllegalArgumentException("Cannot decrement ref count for unknown entity");
   }

   count = delta;
}
else
{
   count += delta;
}
```

> HQLParser - incorrect lazy initialization of the tokenLookup;

Generated by ANTLR; do not edit.

#19 - 03/31/2016 05:08 AM - Igor Skornyakov

Eric Faulhaber wrote:

> Don't know. Greg?
>
> > RouterSessionManager.deregisterSession: checks queue variable for null after a synchronization block on it
>
> This is in an assert, and we don't normally run with asserts enabled. I guess it doesn't hurt to keep this; I would move it up just under the line which defines queue.

Done.

> > LFUAgingCache.NodeImpl: the hits field is modified in the access() method which is invoked from the superclass constructor
>
> Intentional; please leave as is.

I see.

> > FloatConstant.equals - non-transitive. The idea is understandable but the implementation obviously violates the contract. In addition there are multiple places where float point number are compared for equality. I think we need a consistent approach for this. For example we can use the approach similar to one in the java.lang.Float.equals() possibly ignoring
>
> The "equality" comparisons are not direct value comparisons, but rather they are epsilon tests. Perhaps not the best solution, admittedly. Anyway, this is an old library that was repurposed to back TRPL, where I don't think we really use floating point values. I would leave this as is.

OK. What about comparing float primitives (coordinates, sizes) for an equality (==)?

> > TempTableConnection - incorrect lazy initialization of delegate field
>
> IIRC, only one instance of TempTableConnectionProvider is ever created, so this has been safe so far. Also, I intend to replace the use of this class soon. Nevertheless, I see your point. Looks like delegate could be an instance variable.

Thank you. I will think about that.

> > GlobalEventManager - possible NPE in the updateEntityRefCount (if count == null && delta <= 0)
>
> Yes. Suggested fix:
> [...]

May be it better to use AtomicInteger instead of Integer? This will be more efficient. The Exception throwing you suggest doesn't seem a logical solution as we do not enforce the count to be non-negative.

> > HQLParser - incorrect lazy initialization of the tokenLookup;

> Generated by ANTLR; do not edit.

OK.

#20 - 03/31/2016 07:26 AM - Greg Shah

> TemporaryAccountPool uses synchronization on the a pool variable which is ConcurrentLinkedQueue.

Yes, the synchronization is incorrect. I think using a separate private static Object lock = new Object(); would be sufficient.

Also note that createTemporaryAccounts() needs to be protected with this same lock.

#21 - 03/31/2016 07:27 AM - Igor Skornyakov

Greg Shah wrote:

> > TemporaryAccountPool uses synchronization on the a pool variable which is ConcurrentLinkedQueue.
>
> Yes, the synchronization is incorrect. I think using a separate private static Object lock = new Object(); would be sufficient.
>
> Also note that createTemporaryAccounts() needs to be protected with this same lock.

Got it. Thank you.

#22 - 03/31/2016 07:32 AM - Constantin Asofiei

Igor Skornyakov wrote:

1. AclRow.equals - non symmetric e.g. when comparing AclRow instance with other non AclRow TaggedName.

Sub-classes of TaggedName may use different conditions when checking an equality - an AclRow can't be compared to another implementation of TaggedName. So the code is OK.

1. AdminLogon - non safe assignment to logon static field; assignment to an argument of the terminate() method - it is a noop but a comment claims that it destroys session

I don't understand this one.

1. MultiClassLoader - wrong idiom for the managerLibsDir lazy initialization

Yes, this can be fixed. Also, you can change the type for managerLibsDir from String[] to String.

1. ClientSpawner.spawn - not checking return value of the latch.await(TIME_OUT, TimeUnit.SECONDS). Is it correct?

I'm not sure that this is even needed for the remote launch case:

```
            // spawn remote
            BrokerSpawnResult result = cb.remoteStart(brokers, environmentMap);

            // wait for notification
            latch.await(TIME_OUT, TimeUnit.SECONDS);
```

The cb.RemoteStart will not return until the remote side finished the launch work (regardless if fail or succeed in launching).
For the other case, in local start, is OK not to check, as the code doesn't care if the latch's wait time elapsed or it was notified.

#23 - 03/31/2016 07:42 AM - Igor Skornyakov

Constantin Asofiei wrote:

> 1. AdminLogon - non safe assignment to logon static field; assignment to an argument of the terminate() method - it is a noop but a comment claims that it destroys session
>
> I don't understand this one.

It is unsafe to publish the reference to this from inside a constructor. If we need a lazy initialization of the AdminLogon **singleton** there is a standard idiom for this.
The terminate() method assigns null to the session argument which is noop. Are some actions are really required to destroy the session as claimed in the comment?

#24 - 03/31/2016 07:49 AM - Constantin Asofiei

Igor Skornyakov wrote:

> Constantin Asofiei wrote:
>
> > 1. AdminLogon - non safe assignment to logon static field; assignment to an argument of the terminate() method - it is a noop but a comment claims that it destroys session
> >
> > I don't understand this one.
>
> It is unsafe to publish the reference to this from inside a constructor. If we need a lazy initialization of the AdminLogon **singleton** there is a standard idiom for this.

OK, now I understand. We don't need a lazy initialization, the logon static field is just needed to access the singleton instance from other static methods (and only if the singleton was created). It can be fixed.

> The terminate() method assigns null to the session argument which is noop. Are some actions are really required to destroy the session as claimed in the comment?

The null assignment should have been for the AdminLogon.session field, not the session parameter. Is enough to change the line from session = null; to this.session = null;.

#25 - 03/31/2016 07:51 AM - Igor Skornyakov

Constantin Asofiei wrote:

> Igor Skornyakov wrote:
>
> > Constantin Asofiei wrote:
> >
> > 1. AdminLogon - non safe assignment to logon static field; assignment to an argument of the terminate() method - it is a noop but a comment claims that it destroys session
>
> I don't understand this one.
>
> It is unsafe to publish the reference to this from inside a constructor. If we need a lazy initialization of the AdminLogon **singleton** there is a standard idiom for this.
>
> OK, now I understand. We don't need a lazy initialization, the logon static field is just needed to access the singleton instance from other static methods (and only if the singleton was created). It can be fixed.
>
> The terminate() method assigns null to the session argument which is noop. Are some actions are really required to destroy the session as claimed in the comment?
>
> The null assignment should have been for the AdminLogon.session field, not the session parameter. Is enough to change the line from session = null; to this.session = null;.

I see. Thank you Constantin.

#26 - 03/31/2016 10:08 AM - Eric Faulhaber

Igor Skornyakov wrote:

> Eric Faulhaber wrote:
>
> > FloatConstant.equals - non-transitive. The idea is understandable but the implementation obviously violates the contract. In addition there are multiple places where float point number are compared for equality. I think we need a consistent approach for this. For example we can use the approach similar to one in the java.lang.Float.equals() possibly ignoring
>
> The "equality" comparisons are not direct value comparisons, but rather they are epsilon tests. Perhaps not the best solution, admittedly. Anyway, this is an old library that was repurposed to back TRPL, where I don't think we really use floating point values. I would leave this as is.

I just checked, and FloatConstant is actually dead code. It is never referenced outside of itself. I think I implemented it long ago for completeness, in terms of the various constant types defined for the Java virtual machine specification, but I never ended up using it for the purposes of the expression compiler.

> OK. What about comparing float primitives (coordinates, sizes) for an equality (==)?

This is unrelated. The use of the equality (==) operator in TRPL code does not invoke the equals methods defined in the constant classes in the com.goldencode.expr package. We just use the hashCode and equals methods here when creating the constant pool for the class file of a dynamically compiled expression. The Compiler.assembleComparatorOp method would assemble bytecode instructions for an actual equality test expressed in TRPL code.

#27 - 03/31/2016 10:10 AM - Igor Skornyakov

Eric Faulhaber wrote:

> OK. What about comparing float primitives (coordinates, sizes) for an equality (==)?

> This is unrelated. The use of the equality (==) operator in TRPL code does not invoke the equals methods defined in the constant classes in the com.goldencode.expr package. We just use the hashCode and equals methods here when creating the constant pool for the class file of a

> dynamically compiled expression. The Compiler.assembleComparatorOp method would assemble bytecode instructions for an actual equality test expressed in TRPL code.

I mean that we often use == for comparing float point data in our code (e.g. for sizes and coordinates). This looks suspicious.

#28 - 03/31/2016 10:22 AM - Eric Faulhaber

Igor Skornyakov wrote:

> Eric Faulhaber wrote:
>
> > GlobalEventManager - possible NPE in the updateEntityRefCount (if count == null && delta <= 0)
>
> Yes. Suggested fix:
> [...]
>
> May be it better to use AtomicInteger instead of Integer? This will be more efficient. The Exception throwing you suggest doesn't seem a logical solution as we do not enforce the count to be non-negative.

I don't have a problem with AtomicInteger per se, but atomicity isn't the issue here -- this private method is only called from synchronized code already. Yes, AtomicInteger may be more efficient, though so far, this code has not manifested as a bottleneck in any profiling I've done. How would this address the possible NPE?

#29 - 03/31/2016 10:27 AM - Greg Shah

> I mean that we often use == for comparing float point data in our code (e.g. for sizes and coordinates). This looks suspicious.

In the places in the client where we do this, there should be comments in each location stating that it has been checked and is safe. We are

supposed to only do this when the value has come from a known source (like a well known constant or where the value is known to have been cast from an integer).

If you have found any cases to the contrary, then we will want to consider those.

#30 - 03/31/2016 10:36 AM - Igor Skornyakov

Eric Faulhaber wrote:

> May be it better to use AtomicInteger instead of Integer? This will be more efficient. The Exception throwing you suggest doesn't seem a logical solution as we do not enforce the count to be non-negative.

> I don't have a problem with AtomicInteger per se, but atomicity isn't the issue here -- this private method is only called from synchronized code already. Yes, AtomicInteger may be more efficient, though so far, this code has not manifested as a bottleneck in any profiling I've done. How would this address the possible NPE?

Well, it doesn't address the NPE issue by itself. However using of the AtomicInteger will make a check for zero reference count a little bit easier (and in a thread safe way). As I mentioned before it seems a little bit inconsistent that we reject the counter initialization with negative value but allow it to become negative later.

#31 - 03/31/2016 10:45 AM - Eric Faulhaber

Igor Skornyakov wrote:

> Well, it doesn't address the NPE issue by itself. However using of the AtomicInteger will make a check for zero reference count a little bit easier (and in a thread safe way). As I mentioned before it seems a little bit inconsistent that we reject the counter initialization with negative value but allow it to become negative later.

OK, please implement a solution that fixes the possible NPE as you see fit. If you want to discuss further, post it here, otherwise I'll just review the full update when ready.

#32 - 03/31/2016 12:33 PM - Igor Skornyakov

Fixed the 'old' issues from this task and some of the 'new' ones. The issues from #3002 have been fixed as well (those one which are really issues).

Committed to the task branch 2483a revision 10991.

Findbugs (and Eclipse) still report a number of warnings - mostly about potential NPEs and resource leaks. The later are mostly about the situations when a resource is opened in one method and closed in another one.

#33 - 03/31/2016 01:47 PM - Igor Skornyakov

The task branch 2483a was rebased from the trunk revision 10990. Committed to revision 10992.

#34 - 03/31/2016 06:19 PM - Igor Skornyakov

Eric,
Should I continue fixing issues reported by Findbugs/Eclipse static code analysis?
Thank you.

#35 - 03/31/2016 06:27 PM - Eric Faulhaber

Igor Skornyakov wrote:

> Eric,
> Should I continue fixing issues reported by Findbugs/Eclipse static code analysis?
> Thank you.

If you think any are serious, then yes. Based on your earlier assessment of the remaining findings, however, it seems they do not fall into this category. I will review the task branch tonight. Thanks.

#36 - 04/01/2016 12:00 AM - Eric Faulhaber

Code review 2483a/10992:

Lots of good changes. Also lots of format problems. Some comments on substance:

- I don't recognize any of the new items in .bzrignore. Why have these been added?
- Unnecessary import statement added to LRUCache.java.
- I don't understand the point of the changes to Configuration. Why is the additional createConfiguration method necessary, and why is it synchronized when it is only called from the synchronized block in getInstance? Also, the javadoc for createConfiguration is incomplete.
- Not sure about the use of ThreadLocal in DateValue and TimeValue. If these really are used across user sessions, we must use ContextLocal instead. Greg, Constantin: are these objects used across sessions?
- GlobalEventManager: I agree the IllegalArgumentException in updateEntityRefCount isn't necessary; I removed it.
- LogHelper: use ContextLocal instead of ThreadLocal.
- I think we still want to clear the value in integer(Boolean) if the argument is null.
- I expected to see changes to FileStream and AppServerHelper, based on #3002-1. Were these not real issues after all?

I have made a number of adjustments and committed them to 2483a, including addressing some of the above items (though not all). Please review the changes.

Greg, Constantin: although I reviewed all the files, many of them are not ones I normally maintain. Will you please review those files you normally maintain, especially in the UI area?

Current version is 2483a/10993.

#37 - 04/01/2016 04:50 AM - Igor Skornyakov

Eric Faulhaber wrote:

> Code review 2483a/10992:
>
> Lots of good changes. Also lots of format problems. Some comments on substance:
>
> - I don't recognize any of the new items in .bzrignore. Why have these been added?

Sorry, I will remove them.

- Unnecessary import statement added to LRUCache.java.
- I don't understand the point of the changes to Configuration. Why is the additional createConfiguration method necessary, and why is it synchronized when it is only called from the synchronized block in getInstance? Also, the javadoc for createConfiguration is incomplete.

I've added this method because the configuration code updates the instance values may times and the existed code effectively published partially constructed object. It was possible of course to use local variable and assign the instance only when the object is completely assembled, but it looks more error prone.
I've removed the synchronized from the createConfiguration() method and fixed the Javadoc.

- Not sure about the use of ThreadLocal in DateValue and TimeValue. If these really are used across user sessions, we must use ContextLocal instead. Greg, Constantin: are these objects used across sessions?

The only reason to use ThreadLocal is that format is an instance of the non-thread safe class. As it is stateless it is safe to share it between sessions.

- LogHelper: use ContextLocal instead of ThreadLocal.

Please see the comment above. The reason is the same.

- I think we still want to clear the value in integer(Boolean) if the argument is null.

Sorry, I do not understand. Normally we treat null as unknown so the value == null || value.isUnknown() is an idiom. The existed code didn't check for null which is a potential NPE. What do you mean by "clear the value@.?

- I expected to see changes to FileStream and AppServerHelper, based on #3002-1. Were these not real issues after all?

These are not really issues. Eclipse complains about the cases when the resource (Stream or ResultSet) is created in one method ans used in the other one. I do prefer to avoid such situations but there are too many places in our code like this to fix as any such fix typically requires more or less serious logic refactoring.

Committed yo the task branch 2843a revision 10995.

> I have made a number of adjustments and committed them to 2483a, including addressing some of the above items (though not all). Please review the changes.

Thank you, I'm looking on it.

#38 - 04/01/2016 05:25 AM - Igor Skornyakov

Eric Faulhaber wrote:

> If you think any are serious, then yes. Based on your earlier assessment of the remaining findings, however, it seems they do not fall into this category. I will review the task branch tonight. Thanks.

It is difficult to say how serious they are without a thorough analysis. In any case I believe that static code analysis and fixing should be an ongoing process.

#39 - 04/01/2016 06:13 AM - Constantin Asofiei

My review for 2483a rev 10995:

1. ExpressionConversionWorker.expressionType:1056 - adding the stype = expressionType(source.getChildAt(1), infer, fuzzy); for the MINUS branch activates a code which was not active until now. The problem here is this: the code might have been working correctly previously, and with this, we might broke some case. Greg: any thoughts?
2. ExpressionConversionWorker.compareClientWhereNodes - you are changing the code from while (same && (node1 != null || node2 != null)) to while (same && (node1 != null && node2 != null)): at least, I think same should become false if one of node1 or node is null.
3. KeyImport.main - "try with resources" will close the fk stream, so the fk.close() is not needed in the block
4. ConfigSyncManager - this class is not only client-side (as it is used by ConfigManager#syncConfigChanges(Runnable) which is called on server-side widgets), so the ThreadLocal should be moved to ContextLocal. Hynek: please confirm this
5. the LogicalTerminal.messageBox change is incorrect - we need to log the message via UnimplementedFeature.todo, but still execute the code. You can't add the return - fix in another way, i.e. asume the title is empty string, if is null.
6. PushMessagesWorker.waitForMessages change is incorrect - the javadoc states that the caller must place this in a loop and check the running flag., which the caller does. So please revert this change or eventually remove the waitForMessages method and move its code into the run() method.
7. OverlayWindow.realize - if the owner is null, I don't think is ok to assume the ownerId is -1 - we should just return (or log a message). Hynek/Eugenie: any thoughts?
8. integer(Boolean) c'tor - when setting a BDT instance to unknown, its value field should be also set to null, to be consistent.
9. Utils.intersects - are we sure we want to return true if both the set arguments are null?

Hynek/Eugenie: please take a look at issues 4 and 7 in this note.

#40 - 04/01/2016 06:35 AM - Hynek Cihlar

Constantin Asofiei wrote:

> My review for 2483a rev 10995:
> 4 ConfigSyncManager - this class is not only client-side (as it is used by ConfigManager#syncConfigChanges(Runnable) which is called on server-side widgets), so the ThreadLocal should be moved to ContextLocal. Hynek: please confirm this

The intention was really to put the state into the thread local variable. The configuration scope is designed to work only in the scope of the current thread. If you move the state to ContextLocal you may get into trouble when ConfigSyncManager is run from multiple threads.

Is there any particular problem with ThreadLocal on server?

> 7 OverlayWindow.realize - if the owner is null, I don't think is ok to assume the ownerId is -1 - we should just return (or log a message). Hynek/Eugenie: any thoughts?

Well, in case the owner is null (which must never happen for overlay windows by the way), the code
int ownerId = owner != null ? owner.getId().asInt() : null
will throw an exception, due to the implicit unboxing of null Integer.

Let's just make the case more explicit and throw a runtime exception if owner is null.

#41 - 04/01/2016 06:41 AM - Constantin Asofiei

Hynek Cihlar wrote:

> Constantin Asofiei wrote:
>
> > My review for 2483a rev 10995:
> > 4 ConfigSyncManager - this class is not only client-side (as it is used by ConfigManager#syncConfigChanges(Runnable) which is called on server-side widgets), so the ThreadLocal should be moved to ContextLocal. Hynek: please confirm this
>
> The intention was really to put the state into the thread local variable. The configuration scope is designed to work only in the scope of the current thread. If you move the state to ContextLocal you may get into trouble when ConfigSyncManager is run from multiple threads.
>
> Is there any particular problem with ThreadLocal on server?

Thanks for clarifying, there is no problem; as you mention, the thread-local state is not needed by other threads, and all work with this data is done in only one thread. I always double-check when I see thread-local state on server-side, and I missed the Synchronization scopes don't cross thread boundaries. note in the class javadoc.

#42 - 04/01/2016 08:50 AM - Igor Skornyakov

Constantin Asofiei wrote:

> My review for 2483a rev 10995:
>
> 1. ExpressionConversionWorker.expressionType:1056 - adding the stype = expressionType(source.getChildAt(1), infer, fuzzy); for the MINUS branch activates a code which was not active until now. The problem here is this: the code might have been working correctly previously, and with this, we might broke some case. Greg: any thoughts?

Actally I was confused by the Finsbugs message. Please note however that w/o this line the

```
                if ("date".equals(stype)     ||
                    "datetime".equals(stype) ||
                    "datetimetz".equals(stype))
                {
                   jcls = stype;
                }
```

doesn't make sense as stype is always null.
The line removed.

> 1. ExpressionConversionWorker.compareClientWhereNodes - you are changing the code from while (same && (node1 != null || node2 != null)) to while (same && (node1 != null && node2 != null)): at least, I think same should become false if one of node1 or node is null.

You may be true. Please note however that if at leasy one of the nodes is null there will be NPE inside the loop body.

> 1. KeyImport.main - "try with resources" will close the fk stream, so the fk.close() is not needed in the block\

Fixed.

> 1. ConfigSyncManager - this class is not only client-side (as it is used by ConfigManager#syncConfigChanges(Runnable) which is called on server-side widgets), so the ThreadLocal should be moved to ContextLocal. Hynek: please confirm this

I haven't added ThreadLocal to this class

> 1. the LogicalTerminal.messageBox change is incorrect - we need to log the message via UnimplementedFeature.todo, but still execute the code. You can't add the return - fix in another way, i.e. asume the title is empty string, if is null.

Done.

> 1. PushMessagesWorker.waitForMessages change is incorrect - the javadoc states that the caller must place this in a loop and check the running flag.,

which the caller does. So please revert this change or eventually remove the waitForMessages method and move its code into the run() method.
Done.

> 1. OverlayWindow.realize - if the owner is null, I don't think is ok to assume the ownerId is -1 - we should just return (or log a message). Hynek/Eugenie: any thoughts?

Well, I'm not sure what is correct but previously null value of the owner caused NPE.

1. integer(Boolean) c'tor - when setting a BDT instance to unknown, its value field should be also set to null, to be consistent.

Sorry, I do not understand you. Isn't the value if integer primitive?

1. Utils.intersects - are we sure we want to return true if both the set arguments are null?

Fixed.

Committed to the task branch 2483a revision 10996.

#43 - 04/01/2016 08:54 AM - Igor Skornyakov

Hynek Cihlar wrote:

> Well, in case the owner is null (which must never happen for overlay windows by the way), the code
> int ownerId = owner != null ? owner.getId().asInt() : null
> will throw an exception, due to the implicit unboxing of null Integer.
>
> Let's just make the case more explicit and throw a runtime exception if owner is null.

Done in revision 10997.

#44 - 04/01/2016 10:58 AM - Eugenie Lyzenko

Igor Skornyakov wrote:

> Hynek Cihlar wrote:
>
> > Well, in case the owner is null (which must never happen for overlay windows by the way), the code
> > int ownerId = owner != null ? owner.getId().asInt() : null
> > will throw an exception, due to the implicit unboxing of null Integer.
> >
> > Let's just make the case more explicit and throw a runtime exception if owner is null.
>
> Done in revision 10997.

Yes, the overlay owner can not be null. The overlay window is always the feature of something. I think in such cases we must consider the main P2J window(or dialog) is the owner. This window always exists. If not - certainly is seriously wrong and we need to stop execution.

#45 - 04/01/2016 11:58 AM - Igor Skornyakov

Constantin Asofiei wrote:

> Igor Skornyakov wrote:
>
> 1. AclRow.equals - non symmetric e.g. when comparing AclRow instance with other non AclRow TaggedName.
>
> Sub-classes of TaggedName may use different conditions when checking an equality - an AclRow can't be compared to another implementation of TaggedName. So the code is OK.

I do not think so. If AclRow can't be compared to another implementation of the TaggedName why it inherits from it? I do agree with Joshua Bloch (see Effective Java, Item 8) that violation of the contract of equal is not a good idea. In the future this can result in bug which will be very difficult to pinpoint (including memory leaks if TaggedName instances will be used as keys in a HashMap. I think it is better to remove the !(o instanceof AclRow) test in equals (as it is done in compareTo). At least we'll get a ClassCastException if the assumption that the instances of AccRow will should never be compared to instances of other classes will ever become wrong.

#46 - 04/01/2016 02:37 PM - Greg Shah

> My review for 2483a rev 10995:
>
> 1. ExpressionConversionWorker.expressionType:1056 - adding the stype = expressionType(source.getChildAt(1), infer, fuzzy); for the MINUS branch activates a code which was not active until now. The problem here is this: the code might have been working correctly previously, and with this, we might broke some case. Greg: any thoughts?
>
> Actally I was confused by the Finsbugs message. Please note however that w/o this line the
> [...]
> doesn't make sense as stype is always null.
> The line removed.

You both make good points. If I recall, at one time we did have code to lookup stype and it may have caused an issue. I think that is why the same stype lookup only occurs in 3 of the 4 branches.

On the other hand, it is true that the comparison of stype with "date", "datetime" and "datetimetz" will always flow down the else branch. The original code that added this 3-part check was added (by me!) in H041 (bzr rev 10230). The bug was present at that time.

How about this: put the lookup of stype into the if (ftype == null || "BaseDataType".equals(ftype)) section. If this does cause a problem we will see it in our testing. We need to test conversion of the customer application (both server and GUI) to ensure this is OK.

1. ExpressionConversionWorker.compareClientWhereNodes - you are changing the code from while (same && (node1 != null || node2 != null)) to while (same && (node1 != null && node2 != null)): at least, I think same should become false if one of node1 or node is null.

> You may be true. Please note however that if at leasy one of the nodes is null there will be NPE inside the loop body.

I agree. Please force same to false if one of them is null

> Not sure about the use of ThreadLocal in DateValue and TimeValue. If these really are used across user sessions, we must use ContextLocal instead. Greg, Constantin: are these objects used across sessions?

I think this is OK. We do a similar thing in date.java.

Please note: there is a coding standards issue in both DateValue and TimeValue. The open curly brace for the ThreadLocal needs to be on the next line.

#47 - 04/02/2016 04:20 PM - Eric Faulhaber

If anyone has further contributions on any of the current items under discussion, please express those ASAP. This includes any items addressed in 2483a from #3002. We need to get the current changes into regression testing ASAP.

Igor, are you blocked on any of these, or do you have all the information you need to complete this set of fixes? I know this will be an ongoing process, but I'm just referring to the higher priority items you've addressed in task branch 2483a.

#48 - 04/02/2016 04:24 PM - Igor Skornyakov

Eric Faulhaber wrote:

> If anyone has further contributions on any of the current items under discussion, please express those ASAP. This includes any items addressed in 2483a from #3002. We need to get the current changes into regression testing ASAP.
>
> Igor, are you blocked on any of these, or do you have all the information you need to complete this set of fixes? I know this will be an ongoing process, but I'm just referring to the higher priority items you've addressed in task branch 2483a.

Eric,
I do not think that I'm blocked. I plan to commit the final version of changes for those cases which are described and have a mature solution and concentrate on the the customer app environment configuration and #3035 as I understand that it has the top priority now.

#49 - 04/03/2016 05:58 AM - Igor Skornyakov

The Findbugs complains about 3 places in the ScopedDictionary class (statement foo.addAll(dictionary.entrySet())).
Explanation:

```
Bug: Adding elements of an entry set may fail due to reuse of Map.Entry object in com.goldencode.p2j.util.Scop
edDictionary.entrySet(int)
The entrySet() method is allowed to return a view of the underlying Map in which a single Entry object is reus
ed and returned during the iteration. As of Java 1.6, both IdentityHashMap and EnumMap did so. When iterating
 through such a Map, the Entry value is only valid until you advance to the next iteration. If, for example, yo
u try to pass such an entrySet to an addAll method, things will go badly wrong.
```

#50 - 04/03/2016 06:05 AM - Igor Skornyakov

Regarding previous note:
What's the point to use HashSet<Map.Entry<K, V>> instead of HashMap<K,V>?
Thank you.

#51 - 04/03/2016 06:37 AM - Igor Skornyakov

Igor Skornyakov wrote:

> Regarding previous note:
> What's the point to use HashSet<Map.Entry<K, V>> instead of HashMap<K,V>?
> Thank you.

Sorry, of course instead of HashMap<K,V> it should be HashMap<K, Set<V>> (or Multimap<K,V> from Google Guava)

#52 - 04/03/2016 06:52 AM - Igor Skornyakov

Igor Skornyakov wrote:

> The Findbugs complains about 3 places in the ScopedDictionary class (statement foo.addAll(dictionary.entrySet())).
> Explanation:
> [...]

Actually in my simple test the addAll(foo.EntrySet()) works fine both for EnumMap and IdentityHashMap (Java 8).

#53 - 04/03/2016 02:38 PM - Igor Skornyakov

Finished with the next round of issues.

Committed to the task branch 2483a revision 10998.

Started regression testing.

#54 - 04/03/2016 03:08 PM - Igor Skornyakov

Fixed a found.

Committed to the task branch 2483a revision 10999.

Restarted regression testing.

#55 - 04/03/2016 04:30 PM - Eric Faulhaber

Igor Skornyakov wrote:

> What's the point to use HashSet<Map.Entry<K, V>> instead of HashMap<K,V>?

The API in these 3 places calls for a Set<Map.Entry<K, V>> to be returned, hence the calls to Set<Map.Entry<K, V>>.addAll(Map.entrySet()). The point of this API is to allow the idiom:

```
for (Map.Entry<K, V> e : d.entrySet())
{
   K k = e.getKey();
   V v = e.getValue();
   ...
}
```

...which is significantly less expensive than:

```
for (K k : d.keySet())
{
   V v = d.lookup(k);
   ...
```

```
}
```

There is nothing that should change in these locations.

#56 - 04/03/2016 04:37 PM - Igor Skornyakov

Eric Faulhaber wrote:

> Igor Skornyakov wrote:
>
> > What's the point to use HashSet<Map.Entry<K, V>> instead of HashMap<K,V>?
>
> The API in these 3 places calls for a Set<Map.Entry<K, V>> to be returned, hence the calls to Set<Map.Entry<K, V>>.addAll(Map.entrySet()).
> The point of this API is to allow the idiom:
> [...]
> ...which is significantly less expensive than:
> [...]
> There is nothing that should change in these locations.

I understand. However this seems to be a little bit risky (generally speaking). Simply because Map.Entry is not immutable and changing the the source Map can corrupt Set.

#57 - 04/03/2016 04:44 PM - Eric Faulhaber

True, but that's the case for java.util.Map<K,V>.entrySet() as well (after which these APIs were modelled).

#58 - 04/03/2016 04:47 PM - Igor Skornyakov

Eric Faulhaber wrote:

> True, but that's the case for java.util.Map<K,V>.entrySet() as well (after which these APIs were modelled).

That correct. But the documentation explicitly says that

```
 These Map.Entry objects are valid only for the duration of the iteration; more formally, the behavior of a ma
p entry is undefined if the backing map has been modified after the entry was returned by the iterator, except
 through the setValue operation on the map entry.
```

In the situation we discuss Map.Entry are used externally.

#59 - 04/03/2016 04:58 PM - Eric Faulhaber

Code review 2483a/10999:

The changes look OK to me.

Greg, Constantin: please confirm the changes to your files are OK.

Igor, assuming the others are OK with the changes and regression testing is successful, please rebase to trunk rev. 10991, then merge to trunk. There is no need to re-run regression testing after that rebase.

#60 - 04/03/2016 05:00 PM - Igor Skornyakov

Eric Faulhaber wrote:

> Igor, assuming the others are OK with the changes and regression testing is successful, please rebase to trunk rev. 10991, then merge to trunk. There is no need to re-run regression testing after that rebase.

OK, thank you.

#61 - 04/03/2016 06:05 PM - Igor Skornyakov

CTRL-C part passed OK.

#62 - 04/04/2016 04:20 AM - Igor Skornyakov

From the main part the following tests failed in the first run: tc_dc_slot_026 tc_job_002 tc_job_clock_004 tc_ap_vchrs_011.

Test restarted.

#63 - 04/04/2016 06:34 AM - Constantin Asofiei

Review for 2483a 10999:

1. TemporaryAccountPool - you are removing the synchronization for shutDown and poll and tail. The problem here is that the TemporaryAccountWorker is not thread-safe - only one job can be executed at a time. Your changes will allow multiple jobs to be executed concurrently, in the same instance, with unexpected results.
2. AstGenerator - you've backed out the changes in prepareLexer. Is there a specific reason?
3. remove the copyright date/history change for PushMessagesWorker and any other files which no longer have changes
4. UastHints is missing history entry
5. ExpressionConversionWorker.compareClientWhereNodes issue from note 39 needs to be addressed (same becomes false if either node1 or node2 is null).

#64 - 04/04/2016 07:29 AM - Igor Skornyakov

Constantin Asofiei wrote:

> Review for 2483a 10999:
>
> 1. TemporaryAccountPool - you are removing the synchronization for shutDown and poll and tail. The problem here is that the TemporaryAccountWorker is not thread-safe - only one job can be executed at a time. Your changes will allow multiple jobs to be executed concurrently, in the same instance, with unexpected results.

Both poll and tail methods work with TemporaryAccountPool.pool which is a concurrent collection so they do not need an additional synchronization. The shoutDown method() looks thread safe and idempotent. In fact the workerLatch.getCount() 1 check is not needed as CountDownLatch.countDown() is a noop if @CountDownLatch.getCount() 0.

> 1. AstGenerator - you've backed out the changes in prepareLexer. Is there a specific reason?

The problem was that prepareLexer publishes the created Reader so it should not be closed.

> 1. remove the copyright date/history change for PushMessagesWorker and any other files which no longer have changes

Done.

> 1. UastHints is missing history entry

Done.

> 1. ExpressionConversionWorker.compareClientWhereNodes issue from note 39 needs to be addressed (same becomes false if either node1 or node2 is null).

Committed to the task branch 2483a revision 11000.

#65 - 04/04/2016 08:24 AM - Constantin Asofiei

Igor Skornyakov wrote:

> Constantin Asofiei wrote:
>
> > Review for 2483a 10999:
> >
> > 1. TemporaryAccountPool - you are removing the synchronization for shutDown and poll and tail. The problem here is that the TemporaryAccountWorker is not thread-safe - only one job can be executed at a time. Your changes will allow multiple jobs to be executed concurrently, in the same instance, with unexpected results.
>
> Both poll and tail methods work with TemporaryAccountPool.pool which is a concurrent collection so they do not need an additional synchronization. The shoutDown method() looks thread safe and idempotent. In fact the workerLatch.getCount() 1 check is not needed as CountDownLatch.countDown() is a noop if @CountDownLatch.getCount() 0.

I'm not referring to the synchronization of the TemporaryAccountPool.pool field. I'm referring to the TemporaryAccountPool.enableAccount API, which uses the TemporaryAccountPool.poolTask.execute(core); to execute a certain task, which is not thread-safe and needs to be synchronized: the way it is built now, TemporaryAccountWorker can execute only a task at a time, so we need synchronization for it; otherwise, for example, the TemporaryAccountWorker.task field may be changed in execute() by another thread, between lines 79-83.

Also, look how TemporaryAccountWorker.workerLatch is used: this starts initialized to 1, and it gets re-initialized again to 1, after each executed job, so workerLatch.countDown(); is needed, because this is how the caller gets notified the job has finished.

#66 - 04/04/2016 08:34 AM - Igor Skornyakov

After the second run only tc_job_002 and tc_job_clock_004 tests failed.

Test restarted.

#67 - 04/04/2016 08:59 AM - Igor Skornyakov

Constantin Asofiei wrote:

> I'm not referring to the synchronization of the TemporaryAccountPool.pool field. I'm referring to the TemporaryAccountPool.enableAccount API, which uses the TemporaryAccountPool.poolTask.execute(core); to execute a certain task, which is not thread-safe and needs to be synchronized: the way it is built now, TemporaryAccountWorker can execute only a task at a time, so we need synchronization for it; otherwise, for example, the TemporaryAccountWorker.task field may be changed in execute() by another thread, between lines 79-83.

Now I understand, thank you. Fixed. BTW: if enableAccount is a long-running operation may be it makes sense to use an single-threaded Executor service instead of a single thread? Or at least a SynchronousQueue if we want to avoid queue?

> Also, look how TemporaryAccountWorker.workerLatch is used: this starts initialized to 1, and it gets re-initialized again to 1, after each executed job, so workerLatch.countDown(); is needed, because this is how the caller gets notified the job has finished.

I was talking not about workerLatch.countDown() but about workerLatch.getCount()== 1 check.

Committed to the task branch 2483a revision 11001.

#68 - 04/04/2016 09:06 AM - Eric Faulhaber

Igor Skornyakov wrote:

> CTRL-C part passed OK.

This was the first regression testing item you mentioned. However, there were a number of changes in 2483a which impact conversion. Did you run conversion regression testing first?

#69 - 04/04/2016 09:08 AM - Igor Skornyakov

Eric Faulhaber wrote:

> Igor Skornyakov wrote:
>
> > CTRL-C part passed OK.
>
> This was the first regression testing item you mentioned. However, there were a number of changes in 2483a which impact conversion. Did you run conversion regression testing first?

Of course I did. And I've found and fixed a bug which affected conversion.

#70 - 04/04/2016 09:11 AM - Eric Faulhaber

Constantin, AFAIK, TemporaryAccountPool is not exercised in normal regression testing, correct? In other words, is there any reason to restart runtime regression testing with rev. 11001? Is there a manual test we can use to regression test this feature?

#71 - 04/04/2016 10:33 AM - Constantin Asofiei

Eric Faulhaber wrote:

> Constantin, AFAIK, TemporaryAccountPool is not exercised in normal regression testing, correct? In other words, is there any reason to restart runtime regression testing with rev. 11001? > Is there a manual test we can use to regression test this feature?

No, TemporaryAccountPool is not used by normal testing. This is used for appserver agent startup and also for GUI Web clients - ensuring these both work should be enough.

Igor, please add the { on a new line, in TemporaryAccountPool changes you've added, related to this:

```
      synchronized (taskLock) {
```

#72 - 04/04/2016 10:53 AM - Igor Skornyakov

Constantin Asofiei wrote:

> Constantin, AFAIK, TemporaryAccountPool is not exercised in normal regression testing, correct? In other words, is there any reason to restart runtime regression testing with rev. 11001? > Is there a manual test we can use to regression test this feature?

> No, TemporaryAccountPool is not used by normal testing. This is used for appserver agent startup and also for GUI Web clients - ensuring these both work should be enough.

The web client is working.

> Igor, please add the { on a new line, in TemporaryAccountPool changes you've added, related to this:
> [...]

Fixed.

Committed to the task branch 2483a revision 11002

#73 - 04/04/2016 12:22 PM - Igor Skornyakov

Only tc_job_002 test failed after 3 runs of the main-regression.

The task branch 2483a was rebased from the trunk revision 10991. Committed to revision 11003.

Can I merge it to the trunk?
Thank you.

#74 - 04/04/2016 12:57 PM - Eric Faulhaber

Igor Skornyakov wrote:

> Can I merge it to the trunk?

Was the change you made today (rev. 11001) to ExpressionConversionWorker made before or after conversion regression testing passed? If after, you will need to run (only) conversion regression testing against the newest build. If before, then yes, please merge to trunk.

#75 - 04/04/2016 01:01 PM - Igor Skornyakov

Eric Faulhaber wrote:

> Was the change you made today (rev. 11001) to ExpressionConversionWorker made before or after conversion regression testing passed? If after, you will need to run (only) conversion regression testing against the newest build. If before, then yes, please merge to trunk.

This change was made before the regression test conversion.
Regression test conversion restarted

#76 - 04/04/2016 02:36 PM - Igor Skornyakov

Majic code conversion and build with the task branch 2483a finished OK.

#77 - 04/04/2016 02:45 PM - Eric Faulhaber

Great! Please merge to trunk.

#78 - 04/04/2016 02:54 PM - Igor Skornyakov

Merged and committed to the trunk revision 10992.

Task branch 2483a archived

#79 - 04/04/2016 06:53 PM - Eric Faulhaber

- Status changed from New to Closed

- % Done changed from 0 to 100

Since all of the potential defects from this round of static code analysis that were deemed worthwhile to address have been fixed, I am closing this issue. Any future issues will be addressed separately, rather than keeping this issue open indefinitely.

#80 - 11/16/2016 12:06 PM - Greg Shah

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

Files

ias_upd20150109a.zip	322 KB	01/09/2015	Igor Skornyakov