Runtime Infrastructure - Bug #2981

replace SecurityManager.checkCaller() usage with a more performant approach

02/04/2016 10:37 AM - Greg Shah

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

Description

History

#1 - 02/04/2016 10:56 AM - Greg Shah

We have long since implemented SecurityManager.checkCaller()/SecurityManager.checkCallerAbort() as an approach to secure sensitive APIs which have to be called from other packages. The problem is that it is very costly, since it involves examining the call stack to only allow access from pre-determined call sites.

For a while, I've been considering if some kind of token based approach would be a more performance way to handle this. The checkCaller() could still be used, but would only be used once for a kind of initialization call. That call would return a token to the caller. The token would be registered for calling the matching API. Thus, the call stack check could be done once and then the token would be passed as an additional parameter on every call to that matching API. The check for that token's validity (membership in a set of tokens for that API) would be very fast.

However, with the advent of Java 8 and its support for method references, I think these can be used to create a better solution. I'm thinking of taking something like this:

```
public static synchronized void clearInstance()
throws RestrictedUseException
{
   // restrict the use of this method
   String[] callers = new String[]
   {
      "com.goldencode.p2j.main.ClientCore.start"
   };
   checkCallerAbort(callers);

   securityManager = null;
   ContextLocal.clearInstance();
}
```

and refactoring it into this:

```
public static synchronized Runnable clearInstanceSetup()
throws RestrictedUseException
{
   // restrict the use of this method
   String[] callers = new String[]
   {
      "com.goldencode.p2j.main.ClientCore.start"
   };
   checkCallerAbort(callers);

   return SecurityManager::clearInstance();
}

private static synchronized void clearInstance()
{
   securityManager = null;
   ContextLocal.clearInstance();
}
```

Each caller would have to call clearInstanceSetup() once and then would have direct access to the private "worker". For different use cases, we would implement different functional interfaces, but otherwise the approach is the same. It is no more complicated to use than the token idea, but the implementation is simpler and the performance would be better (no set lookup and comparison).