Leaks - Proposed Solutions

Added by GREGORIO CLAUDIO, last edited by RIVA ALBERTO (Consulente) on Mar 26, 2014

Fix Spring context cleaning

to clean the Intrsospector cache used by Spring use the listener

```
<listener>
        <listener-class>org.springframework.web.util.IntrospectorCleanupLister
</listener>
```

Fix for Resource Bundle references

Simply call the

```
ResourceBundle.clearCache(Thread.currentThread().getContextClassLoader());
```

to clean the cache linked to the actual class loader.

It can be solved adding the following ServletContextListener in the web.xml**:

```
<listener>
        <listener-class>it.sella.utilcore.http.ResourceBundleCacheCleanupListe
</listener>
```

Fix for Log4j Leaks

Simply call the

```
LogFactory.release(Thread.currentThread().getContextClassLoader());
```

to clean the cache linked to the actual class loader. This is true for project that are using the log4j library so not for H2O log4j utils (that will be fixed in another way)

In case of usage of it.sella.util.log4Debug you must use new Util version 5.11.0 in your ear.

<dependency>
  <groupId>it.sella.util</groupId>
  <artifactId>util-lib-core</artifactId>
  <version>5.11.0</version>
</dependency>

```
<dependency>
  <groupId>it.sella.util</groupId>
  <artifactId>util-lib-core</artifactId>
  <version>5.11.0</version>
</dependency>
```

As for sun.awt.AppContext we will give a startup class for creating the FileWatchDog Thread at System Classloader level.

It can be solved adding the following ServletContextListener in the web.xml**:

```
<listener>
        <listener-class>it.sella.utilcore.http.CommonsLoggingCacheCleanupListe
</listener>
```

Fix the sun.awt.AppContext leak

There is a bug where the sun.awt.AppContext is attached to the wrong ClassLoader. To fix we propose to use the following code:

```
try {
    final ClassLoader active = Thread.currentThread().getContextClassLoader();
    try {
     //Find the root classloader
     ClassLoader root = active;
     while (root.getParent() != null) {
      root = root.getParent();
     }
     //Temporarily make the root class loader the active class loader
     Thread.currentThread().setContextClassLoader(root);
     //Force the AppContext singleton to be created and initialized
    sun.awt.AppContext.getAppContext();
    } finally {
     //restore the class loader
     Thread.currentThread().setContextClassLoader(active);
    }
   } catch ( Throwable t) {
    //Carry on if we get an error
   }
```

The fix will be deployed using a startup class centrally. No activity is requested to development groups on this issue.

Fix for the leak present in DefaultValidationProviderResolver (validation API).

The issue in DefaultValidationProviderResolver is in this portion of code

```
//cache per classloader for an appropriate discovery
//keep them in a weak hashmap to avoid memory leaks and allow proper hot redep
//TODO use a WeakConcurrentHashMap
//FIXME The List<VP> does keep a strong reference to the key ClassLoader, use
private static final Map<ClassLoader, List<ValidationProvider<?>>> providersPe
```

and is present if you ship a ValidatorProvider in the webapp (like Hibernate does) AND a ValidatorProvider is present in the Application Server too (like Weblogic 12c does)
In this configuration (Web classpath + AS classpath ValidatorProvider) the WebApp leak a strong reference to his

ClassLoader in the above code.

Solution

The proposed solution is to make a search in the defined classes javax.validation.Validation, search if it's presents the inner class DefaultValidationProviderResolver and clean the Strong reference (this is done to avoid unexpected behavior in custom Validation providers)

```
try {
        List<Class<?>> declaredClasses = Arrays.asList(Class.forName("javax.va
        for(Class<?> c : declaredClasses){
                if(c.getName().contains("DefaultValidationProviderResolver")){
                        Field field = c.getDeclaredField("providersPerClassloa
                        field.setAccessible(true);
                        Map<ClassLoader, List<ValidationProvider<?>>> provider
                        providersPerClassloader.remove(Thread.currentThread().
                        break;
                }
        }
} catch (Exception e) {
        log.error(e,e);
}
```

It can be solved adding the following ServletContextListener in the web.xml**:

```
<listener>
        <listener-class>it.sella.utilcore.http.DefaultValidationProviderResolv
</listener>
```

Fix for the leak present in ThreadLocalContainerResolver (JAX-WS API).

It seems that the ThreadLocalContainerResolver class hold a ThreadLocal variable that is never removed from his Thread. This causes that, every thread that is used to make a WS request, attach a ThreadLocal to the active thread that is never removed.

We opened a bug on the library https://java.net/jira/browse/JAX_WS-1145

Solution

Here the proposed filter to do the job

```
@SuppressWarnings({ "rawtypes", "serial" })
public class JAXWSMemoryLeakFixFilter implements Filter, Serializable
{ private transient ThreadLocal    defaultContainer=null;
 public JAXWSMemoryLeakFixFilter() {}
 @Override
 public void destroy()
 {
  checkIfMineNoneContainerIsAttached();
 }
 @Override
 public void doFilter(ServletRequest arg0, ServletResponse arg1,FilterChain ar
 {
  try
  {
   arg2.doFilter(arg0, arg1);
  }
  finally
  {
   checkIfMineNoneContainerIsAttached();
  }
 }
 private void checkIfMineNoneContainerIsAttached()
 {
  try
  {
   if (defaultContainer != null)
   {
    defaultContainer.remove();
   }
  }
  catch (Throwable t){/*we can not make anything*/}
 }
 @Override
 public void init(FilterConfig arg0) throws ServletException
 {
  try
  {
   final Field f1 = ThreadLocalContainerResolver.class.getDeclaredField("conta
   f1.setAccessible(true);
   defaultContainer = (ThreadLocal)f1.get(ContainerResolver.getDefault());
   checkIfMineNoneContainerIsAttached();
  }
  catch ( Throwable t) {/*we can not make anything*/}
 }
}
```

It can be solved adding the following Filter in the web.xml**:

```
<filter>
        <filter-name>JAXWSDefaultContainerResolverCleanupFilter</filter-name>
        <filter-class>it.sella.utilcore.http.JAXWSDefaultContainerResolverClea
</filter>
<filter-mapping>
        <filter-name>JAXWSDefaultContainerResolverCleanupFilter</filter-name>
        <url-pattern>/*</url-pattern>
</filter-mapping>
```

All the part indicated with ** could be used including this library:

```
<dependency>
  <groupId>it.sella.utilcore</groupId>
  <artifactId>utilcore-lib</artifactId>
  <version>1.8.0_1</version>
</dependency>
```

Pay attention to the listener sequence since for example if u need to use all the listener the sequence must be this:

```
<listener>
        <listener-class>org.springframework.web.util.IntrospectorCleanupLister
</listener>
<listener>
        <listener-class>it.sella.utilcore.http.DefaultValidationProviderResolv
</listener>
<listener>
        <listener-class>it.sella.utilcore.http.ResourceBundleCacheCleanupListe
</listener>
<listener>
        <listener-class>it.sella.utilcore.http.CommonsLoggingCacheCleanupListe
</listener>
```

Fix for the leak on apache.commons.pool

Common pool creates an eviction TimerThredad that run even after undeploy of the application. Inside the TimerThread the application classloader is maintained in memory.

There was a bug inside version 1.x that does not release the Thread even if the close() message was called during application undeploy.

IF your application use apache commons.pool you are required to upgrade to version 2.x. Migration need some easily changes in your code, because apache change the package name.

After upgrading the library you have to register:

```
org.apache.commons.pool2.impl.GenericObjectPool.close()
```

inside your shutdown listener. Here is an example of listener, where StringBufferPool is only an example of a singleton class.

Any ear that use pool internnaly has the responsability to create its own listern class that mimics thiwe behaviour.

```
package it.sella.test.common;

import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;

public class Destroyer implements
ServletContextListener {

        public Destroyer() {
                //do nothing
        }
         public void contextInitialized(ServletContextEvent event) {

              /* This method is called prior to the servlet context being
                 initialized (when the Web application is deployed).
```

```
                    You can initialize servlet context related data here.
                  */

              }
              public void contextDestroyed(ServletContextEvent event) {

                StringBufferPool pool=StringBufferPool.getInstance();
                pool.close();
                pool.setTimeBetweenEvictionRunsMillis(-1);

              }
}
```

Fix for the leak on org.springframework.ws.transport.context.TransportContextHolder

The issue, presents in spring-ws, is quite the same of JAX-WS memory leak. The TransportContextHolder doesn't release a ThreadLocal Variable.

```
@SuppressWarnings({ "rawtypes", "serial" })
public class SpringWSTransportMemoryLeakFixFilter implements Filter,
                Serializable {
        private transient ThreadLocal defaultContainer = null;

        public SpringWSTransportMemoryLeakFixFilter() {
        }

        @Override
        public void destroy() {
                checkIfMineNoneContainerIsAttached();
        }

        @Override
        public void doFilter(ServletRequest arg0, ServletResponse arg1,
                        FilterChain arg2) throws IOException, ServletExceptior
                try {
                        arg2.doFilter(arg0, arg1);
                } finally {
                        checkIfMineNoneContainerIsAttached();
                }
        }

        private void checkIfMineNoneContainerIsAttached() {
                try {
                        if (defaultContainer != null) {
                                defaultContainer.remove();
                        }
                } catch (Throwable t) {/* we can not make anything */
                }
        }

        @Override
        public void init(FilterConfig arg0) throws ServletException {
                try {
                        final Field f1 = org.springframework.ws.transport.cont
                                        .getDeclaredField("transportContextHol
                        f1.setAccessible(true);
                        defaultContainer = (ThreadLocal) f1.get(null);
                        checkIfMineNoneContainerIsAttached();
                } catch (Throwable t) {/* we can not make anything */
                }
        }
}
```

It can be solved adding the following Filter in the web.xml**:

```
<filter>
        <filter-name>SpringWSTransportContextHolderCleanupFilter</filter-name>
        <filter-class>it.sella.utilcore.http.SpringWSTransportContextHolderCle
</filter>
<filter-mapping>
        <filter-name>SpringWSTransportContextHolderCleanupFilter</filter-name>
        <url-pattern>/*</url-pattern>
</filter-mapping>
```

Fix leak on TimerThread from OpenAm filter.

For application already miwith OpenAm a leak is generated from a timerThread created by OpenaAm sdA easily solution is to add inside your web.xml the following declaration:

```
<listener>
<listener-class>com.sun.identity.common.ShutdownServletContextListener</lister
</listener>
```

Printed by Atlassian Confluence 3.4.9, the Enterprise Wiki.