Code Inspection

Beretta Carolina 852650
Brizzolari Cecilia 852399

January 5, 2016

Contents

1 Assigned Class

We had to analyze two different methods, belonging to the same class.

Name: loadServletClass( )
Start Line: 1451
Location: appserver/web/web-core/src/main/java/org/apache/catalina/core/
StandardWrapper.java

```
1451     /*
1452      * Loads the servlet class
1453      */
1454     private synchronized void loadServletClass() throws
             ServletException {
1455         if (servletClass != null) {
1456             return;
1457         }
1458 
1459         // If this "servlet" is really a JSP file, get the right
                 class.
1460         String actualClass = servletClassName;
1461         if ((actualClass == null) && (jspFile != null)) {
1462             Wrapper jspWrapper = (Wrapper)
1463                 ((Context) getParent()).findChild(Constants.
                         JSP_SERVLET_NAME);
1464             if (jspWrapper != null) {
1465                 actualClass = jspWrapper.getServletClassName();
1466                 // Merge init parameters
1467                 String paramNames[] = jspWrapper.findInitParameters
                         ();
1468                 for (String paramName : paramNames) {
1469                     if (parameters.get(paramName) == null) {
1470                         parameters.put(paramName,
1471                                        jspWrapper.findInitParameter
                                               (paramName));
1472                     }
1473                 }
1474             }
1475         }
1476 
1477         // Complain if no servlet class has been specified
1478         if (actualClass == null) {
1479             unavailable(null);
1480             String msg = MessageFormat.format(rb.getString(
                     NO_SERVLET_BE_SPECIFIED_EXCEPTION), getName());
1481             throw new ServletException(msg);
1482         }
1483 
1484         // Acquire an instance of the class loader to be used
1485         Loader loader = getLoader();
1486         if (loader == null) {
1487             unavailable(null);
1488             String msg = MessageFormat.format(rb.getString(
                     CANNOT_FIND_LOADER_EXCEPTION), getName());
1489             throw new ServletException(msg);
1490         }
```

```java

        ClassLoader classLoader = loader.getClassLoader();

        // Special case class loader for a container provided servlet
        //
        if (isContainerProvidedServlet(actualClass) &&
                ! ((Context)getParent()).getPrivileged() ) {
            // If it is a priviledged context - using its own
            // class loader will work, since it's a child of the container
            // loader
            classLoader = this.getClass().getClassLoader();
        }

        // Load the specified servlet class from the appropriate class loader
        Class clazz = null;
        try {
            if (SecurityUtil.isPackageProtectionEnabled()){
                final ClassLoader fclassLoader = classLoader;
                final String factualClass = actualClass;
                try{
                    clazz = AccessController.doPrivileged(
                        new PrivilegedExceptionAction<Class>(){
                            public Class run() throws Exception{
                                if (fclassLoader != null) {
                                    return fclassLoader.loadClass(factualClass);
                                } else {
                                    return Class.forName(factualClass);
                                }
                            }
                    });
                } catch(PrivilegedActionException pax){
                    Exception ex = pax.getException();
                    if (ex instanceof ClassNotFoundException){
                        throw (ClassNotFoundException)ex;
                    } else {
                        String msgErrorLoadingInfo = MessageFormat.format(rb.getString(ERROR_LOADING_INFO),
                                                                        new Object[] {fclassLoader, factualClass});
                        getServletContext().log(msgErrorLoadingInfo, ex );
                    }
                }
            } else {
                if (classLoader != null) {
                    clazz = classLoader.loadClass(actualClass);
                } else {
                    clazz = Class.forName(actualClass);
                }
            }
        } catch (ClassNotFoundException e) {
            unavailable(null);
            String msgErrorLoadingInfo = MessageFormat.format(rb.
                getString(ERROR_LOADING_INFO),
                    new Object[] {classLoader, actualClass});
            getServletContext().log(msgErrorLoadingInfo, e );
            String msg = MessageFormat.format(rb.getString(
                CANNOT_FIND_SERVLET_CLASS_EXCEPTION), actualClass);
            throw new ServletException(msg, e);
        }

        if (clazz == null) {
            String msg = MessageFormat.format(rb.getString(
                CANNOT_FIND_SERVLET_CLASS_EXCEPTION), actualClass);
            unavailable(null);
            throw new ServletException(msg);
        }

        servletClass = castToServletClass(clazz);
    }
```

Name: initServlet(Servlet servlet)
Start Line: 1562
Location: appserver/web/web-core/src/main/java/org/apache/catalina/core/StandardWrapper.java

```
    /**
     * Initializes the given servlet instance, by calling its init
         method.
     */
    private void initServlet(Servlet servlet) throws
        ServletException {
        if (instanceInitialized && !singleThreadModel) {
            // Servlet has already been initialized
            return;
        }

        try {
            instanceSupport.fireInstanceEvent(BEFORE_INIT_EVENT,
                servlet);
            // START SJS WS 7.0 6236329
            //if( System.getSecurityManager() != null) {
            if ( SecurityUtil.executeUnderSubjectDoAs() ){
            // END OF SJS WS 7.0 6236329
                Object[] initType = new Object[1];
                initType[0] = facade;
                SecurityUtil.doAsPrivilege("init", servlet,
                    classType,
                                           initType);
                initType = null;
            } else {
                servlet.init(facade);
            }

            instanceInitialized = true;

            // Invoke jspInit on JSP pages
            if ((loadOnStartup >= 0) && (jspFile != null)) {
                // Invoking jspInit
                DummyRequest req = new DummyRequest();
                req.setServletPath(jspFile);
                req.setQueryString("jsp_precompile=true");

                // START PWC 4707989
                String allowedMethods = (String) parameters.get("
                    httpMethods");
                if (allowedMethods != null
                        && allowedMethods.length() > 0) {
                    String[] s = allowedMethods.split(",");
                    if (s.length > 0) {
                        req.setMethod(s[0].trim());
                    }
                }
                // END PWC 4707989

                DummyResponse res = new DummyResponse();

                // START SJS WS 7.0 6236329
                //if( System.getSecurityManager() != null) {
                if ( SecurityUtil.executeUnderSubjectDoAs() ){
                // END OF SJS WS 7.0 6236329
                    Object[] serviceType = new Object[2];
                    serviceType[0] = req;
                    serviceType[1] = res;
                    SecurityUtil.doAsPrivilege("service", servlet,
                                                   classTypeUsedInService
                                                       ,
                                                   serviceType);
                } else {
                    servlet.service(req, res);
                }
            }
            instanceSupport.fireInstanceEvent(AFTER_INIT_EVENT,
                servlet);

        } catch (UnavailableException f) {
            instanceSupport.fireInstanceEvent(AFTER_INIT_EVENT,
                servlet, f);
            unavailable(f);
            throw f;

        } catch (ServletException f) {
            instanceSupport.fireInstanceEvent(AFTER_INIT_EVENT,
                servlet, f);
            // If the servlet wanted to be unavailable it would
                have
            // said so, so do not call unavailable(null).
            throw f;

        } catch (Throwable f) {
            getServletContext().log("StandardWrapper.Throwable", f)
                ;
            instanceSupport.fireInstanceEvent(AFTER_INIT_EVENT,
                servlet, f);
            // If the servlet wanted to be unavailable it would
                have
            // said so, so do not call unavailable(null).
            String msg = MessageFormat.format(rb.getString(
                SERVLET_INIT_EXCEPTION), getName());
            throw new ServletException(msg, f);
        }
    }
```

2 Functional Role

The functional role of the class, as stated in the javadoc, is:

Standard implementation of the Wrapper interface that represents an individual servlet definition. No child Containers are allowed, and the parent Container must be a Context.

As far as the two methods we were assigned, the following two subsections illustrate their role.

2.1 loadServletClass

This method has no javadoc documentation, as it is a private method, and the provided comment to the method is a bit generic and almost useless:

Loads the servlet class

Its aim is to assign the correct value to the class variable *servletClass*, that indicates the class from which the servlet will be instantiated, if no value has already been set. Each block of code is well commented, and provides the necessary information to understand what that block is supposed to do. The method handles the different kind of exceptions that can occur, such as not being able to find the class loader or the servlet class, by throwing a *ServletException* with a message explaining the reason of the problem.

2.2 initServlet

This method is called when a servlet needs to be initialized after being loaded, as stated in the javadoc documentation:

Initializes the given servlet instance, by calling its init method

It first checks if the servlet provided has already been initialized, if not it executes two tasks: initializing it and then activating the service the chosen servlet has to offer by calling other methods. For each task there are two blocks of operations, one to be executed when there are security issues and the other when the servlet is not protected. The code deals also with catching two different kind of exceptions: *UnavailableException*, which then proceeds to call the method *unavailable(UnavailableException f)* that marks the servlet as unavailable for a given amount of time, and a generic *ServletException.*

3 Issues

This chapter illustrates the issues found in the assigned code, following the checklist provided by the professor.

3.1 StandardWrapper Class

The line numbers in this section refers to the line of the source code.

1. **Class and Interface Declaration**

   (a) (checklist 25D; line 118): the visibility order is not respected, as a private variable is listed before the public ones

   (b) (checklist 25E; lines 292, 392): the visibility order is not respected, as protected variables are mixed up with private ones

   (c) (checklist 25E, line 279): a static variable is mixed up with instance ones

   (d) (checklist 25F, lines 229-235): the constructor is declared in between class and instance variables and not after them

Other

1. **line 1356**: ambiguous fix me is not handled

```
/**
 * FIXME: Fooling introspection ...
 */
public Wrapper findMappingObject() {
    return (Wrapper) getMappingObject();
}
```

2. **lines 2289-2299**: the three methods define attributes of the class, but instead of returning the attribute itself they directly return the value, as no attribute is defined. In order to improve maintainability, the StandardWrapper class should have three private boolean attributes and the methods should return their value.

```
    public boolean isEventProvider() {
        return false;
    }

    public boolean isStateManageable() {
        return false;
    }

    public boolean isStatisticsProvider() {
        return false;
    }
```

3. **line 2029-2045**: the method does nothing and should be removed. Moreover it is listed under the Private Method section, while the method itself is protected.

```
  // -------------------------------------------------------- Package Methods

  // -------------------------------------------------------- Private Methods
    /**
     * Add a default Mapper implementation if none have been
         configured explicitly.
     *
     * @param mapperClass Java class name of the default
         Mapper
     */
    protected void addDefaultMapper(String mapperClass) {

        // No need for a default Mapper on a Wrapper

    }
```

3.2 loadServletClass

1. **Naming convention**

   (a) (checklist 1; line 1505, 1508, 1509): the variables *clazz*, *fclassLoader*, *factualClass* do not have a meaningful name

   (b) (checklist 6; line 1508): the variable *fclassLoader* does not respect naming conventions as the word "class" is not capitalized

2. **Indentation** (checklist 8; lines 1496-1497, 1526-1527, 1540-1541): when a line break occurs the indentation is no more consistent as an arbitrary number of white spaces are used. Lines 1470-1471 are not listed in this point, as the number of white spaces is not arbitrary, but is selected to align the line with the beginning of the expression of the previous one.

3. **File Organization** (checklist 13; lines 1480, 1488, 1526-1527, 1540, 1543, 1548): line length exceeds 80 characters, but is under 120 characters

4. **Wrapping Lines**

    (a) (checklist 15; lines 1511-1512): line break occurs after an open parenthesis

    (b) (checklist 17, lines 1496-1497, 1511-1512, 1526-1527, 1540-1591): after a break line, the new statement is not aligned with the beginning of the expression of the previous line. However line 1496 is acceptable because it is indented with the eight eight spaces rule

5. **Initialization and Declaration** (checklist 30; line 1462, 1485, 1492): the constructor is not called when a new object is desired, but it is immediately initialized

6. **Arrays**

    (a) (checklist 39; line 1467): the constructor is not called for the array

    (b) (line 1467): the array designators "[]" should be on the type, not the variable

7. **Exception** (checklist 53, line 1525-1528): when the caught exception at line 1521 is not an instance of the ClassNotFoundException, the catch block logs the exception but no further action is taken to resolve the problem

The local variables at lines 1508-1509 may seem useless at first glance, as they are just a copy of other local variables and their value is never modified, since they are declared as final. However they are needed because in order to use variable inside the privileged block, they must be declared as final[1]. The partial duplication of these variables could be avoided if the original variable *actualClass* were declared as final instead, as it does not need to be modified after their first initialization.

3.3 initServlet

1. **File Organization** (checklist 13; line 1640): the line exceeds the limit of 80 characters, but it is a chain call so it couldn't be avoided. Furthermore, it doesn't exceed 120 characters

2. **Wrapping Lines**

    (a) (checklist 15; line 1597): the line break happens before the "&&" operator instead of being after

[1] http://docs.oracle.com/javase/7/docs/technotes/guides/security/doprivileged.html

(b) (checklist 17; line 1598): this line should be aligned with the round parenthesis of the line above, although it is still acceptable since it was done for readability's sake and the line is indented with 8 spaces

3. **Comments**

   (a) (checklist 18; lines 1576, 1611): these lines of comment should be after the block of code they refer to

   (b) (checklist 19; lines 1574, 1609, 1631, 1632, 1638, 1639): these lines of comment are to be deleted afterwards, but there is no date on which the deletion should be done

4. **Initialization and Declaration** (checklist 33; line 1606): the variable "res" is not declared at the start of the *if* braces, but instead after another nested *if* has been executed

5. **Arrays** (checklist 39; line 1599): the String[] array is being created without calling the constructor, instead the variable gets filled with the return value from the *split(",")* method

6. **Computations, comparisons and assignments** (checklist 44; lines 1613, 1614): these lines should have been written as one line, *serviceType = { req, res };* , to avoid brutish programming

7. **Exceptions** (checklist 53; lines 1625, 1630, 1637): these three lines of code are exactly the same, but the issue comes up because the third one is inside a *catch* of a generic *Throwable f*; this means that this particular instruction will be reached every time any of the other two exceptions fire, thus repeating the same action twice

A Appendix

A.1 Software and tools

1. *Lyx* to redact and format the document

A.2 Hours of work

- Carolina Beretta ~ 10h
- Cecilia Brizzolari ~ 10h