Inspection Document

Version 1.0

Giorgio Pea(Mat. 853872), Andrea Sessa(Mat. 850082)

5/1/2016

Contents

1 Introduction

The inspection document that follows contains the results of an inspection process on the classes and the methods listed below.
These results inspection process is based on a given checklist(see reference section) that enumerates a series of code errors and software aspects to be considered. Additional considerations over the quality and the functional role of the software artifacts analyzed are also mentioned.

2 Classes

Included in this section are two java classes under analysis.

File: /appserver/web/web-core/src/main/java/org/apache/catalina/ssi/**SSIServlet.java**

Methods under inspection:

- ***init()***
- ***requestHandler(HttpServletRequest req, HttpServletResponse res)***
- ***processSSI(HttpServletRequest req , HttpServletResponse res , URL resource)***

File: /appserver/web/web-core/src/main/java/org/apache/catalina/ssi/**SSIMediator.java**

Methods under inspection:

- ***substituteVariables(String val)***

3 Functional Role

Included in this section is some information about the functional behavior of the analyzed classes and methods.

3.1 SSIServlet.java

From the Javadoc:

```
/**
 * Servlet to process SSI requests within a webpage. Mapped to a path
    from
 * within web.xml.
 *
```

This class represents a Java EE Servlet used to process HTTP requests that includes some SSI instructions.
SSI(Server Side Include) is a interpreted server-side scripting language, that is usually used to include the contents of one or more files into a web page.
Follows a description of the methods the analyzed for this class.

- ***Init()***
  From the javadoc of the method:

  ```
      /**
       * Initialize this servlet.
       *
       * @exception ServletException
       *                if an error occurs
       */
  ```

  This method takes care of initialize the Java EE Servlet by retrieving and setting the configuration parameters.
  The role of these configuration parameters is explained below:

  - **debug**: A flag used to manage debug messages: if debug = 0 then no debug messages logged
  - **isVirtualWebappRelative**: A flag used to manage how file paths in SSI instructions are processed: if isVirtualWebappRelative is true then all the file path are considered relative to the root of the web-server.
  - **expires**: Specifies the expiration time of the HTTP request managed by Servlet(in seconds)
  - **buffered**: A flag used to manage how the content of HTTP responses processed by this Servlet are written on the output stram: if buffered is true then this writing is buffered(see processSSI() for additional explanations)
  - **inputEncoding**: Specifies the characters encoding of the HTTP requests processed by this Servlet

  - **outputEncoding**: Specifies the characters encoding of the HTTP responses processed by this Servlet

- ***requestHandler()***
  From the Javadoc of the method:

```java
    /**
     * Process our request and locate right SSI command.
     *
     * @param req
     *            a value of type 'HttpServletRequest'
     * @param res
     *            a value of type 'HttpServletResponse'
     */
```

  This method takes care of process HTTP get or post requests managed by this Servelet. In particular requestHandler() retrieves the file paths present in eventual SSI commands, checks if they points to an existing resource on the web server, sets some configuration parameters for the processed HTTP request, and invokes processSSI().

  In this block of code the method checks the debug flag of the Servelet; If debug is greater than zero then logs a message into the logger for debug purposes.

```java
        ServletContext servletContext = getServletContext();
        String path = SSIServletRequestUtil.getRelativePath(req);
        if (debug > 0)
            log("SSIServlet.requestHandler()\n" + "Serving "
                    + (buffered?"buffered ":"unbuffered ") + "resource '"
                    + path + "'");
```

  This block of code checks if the file paths present in the SSI commands does exists or points to a resource in the ‘WEB-INF‘ or ‘META-INF‘ subdirectories, if so the method terminates and logs an error message.

```java
        // Exclude any resource in the /WEB-INF and /META-INF subdirectories
        // (the "toUpperCase()" avoids problems on Windows systems)
        if (path == null || path.toUpperCase(Locale.ENGLISH).startsWith("/WEB-INF")
                || path.toUpperCase(Locale.ENGLISH).startsWith("/META-INF")) {
            res.sendError(HttpServletResponse.SC_NOT_FOUND, path);
            log("Can't serve file: " + path);
            return;
        }
```

  In this block of code the method tries to retrieve the resource associated with the file path present in the SSI commands. If this resource doesn’t exist the method terminates and logs an error message.

```
        URL resource = servletContext.getResource(path);
        if (resource == null) {
            res.sendError(HttpServletResponse.SC_NOT_FOUND, path);
            log("Can't find file: " + path);
            return;
        }
```

In this block of code the method sets some header fields of the HTTP response to be processed: it sets the mime type, the character encoding of the output text and the expiration time for the response(in seconds, see init()). At the end of the method the processSSI() method is invoked.

```
        String resourceMimeType = servletContext.getMimeType(path);
        if (resourceMimeType == null) {
            resourceMimeType = "text/html";
        }
        res.setContentType(resourceMimeType + ";charset=" +
            outputEncoding);
        if (expires != null) {
            res.setDateHeader("Expires", (new java.util.Date()).
                getTime()
                    + expires.longValue() * 1000);
        }
        req.setAttribute(Globals.SSI_FLAG_ATTR, "true");
        processSSI(req, res, resource);
```

- ***processSSI()*** *No Javadoc is available for this method* This method parses the SSI commands contained in a given HTTP request via the SSIProcessor class and writes the result of this processing in the output stream of the Servlet.

  In this block of code the method takes care of the initialization of the stream used to read the data contained in the resource and checks the characters encoding of this data according to the character encoding configuration of the Servelet

```java
        URLConnection resourceInfo = resource.openConnection();
        InputStream resourceInputStream = resourceInfo.
            getInputStream();
        String encoding = resourceInfo.getContentEncoding();
        if (encoding == null) {
            encoding = inputEncoding;
        }
        InputStreamReader isr;
        if (encoding == null) {
            isr = new InputStreamReader(resourceInputStream);
        } else {
            isr = new InputStreamReader(resourceInputStream,
                encoding);
        }
        BufferedReader bufferedReader = new BufferedReader(isr);
```

  In this block of code the method parses the SSI commands contained in the web page by using the process() method of the SSIProcessor class. The results are written on the Servlet output stream.

```java
        SSIProcessor ssiProcessor = new SSIProcessor(
            ssiExternalResolver)
```

```java
        long lastModified = ssiProcessor.process(bufferedReader,
                resourceInfo.getLastModified(), printWriter);
```

  In this block of code the method sets different types of output stream for the given HTTP response accordingly to buffered flag of the Servelet, and flushes them.

```java
        if (buffered) {
            stringWriter = new StringWriter();
            printWriter = new PrintWriter(stringWriter);
        } else {
            printWriter = res.getWriter();
        }
```

```java
        if (buffered) {
            printWriter.flush();
            String text = stringWriter.toString();
            res.getWriter().write(text);
        }
```

3.2 SSIMediator.java

From the Javadoc:

```
/**
 * Allows the different SSICommand implementations to share data/talk to each
 * other
 *
```

This class take care of how many different implementations of the SSI instructions can communicate and exchange data with each other.
Follows a description of the methods the analyzed for this class.

- ***substituteVariables()*** From the Javadoc:

```
  /**
   * Applies variable substitution to the specified String and returns the
   * new resolved string.
   */
```

This method processes a given string in a way that HTML special characters are normalized and SSI variable are replaced with their actual value.

In this block of code the method checks the string contains '$' or '&, if not so the method terminates returning the original string(no processing is needed)

```
    // If it has no references or HTML entities then no work
    // need to be done
   if (val.indexOf('$') < 0 && val.indexOf('&') < 0) return val;
```

In this block of code the method normalizes HTML special characters.

```
    // HTML decoding
    val = val.replace("&lt;", "<");
    val = val.replace("&gt;", ">");
    val = val.replace("&quot;", "\"");
    val = val.replace("&amp;", "&");
```

In this block of code the method takes care of substuting '&#n' with 'n' where 'n' is an integer number. See the javadoc of StringBuilder(Java SE 7 class) for a detailed explanation of the methods.
The remaining code replaces SSI variables with their current value.
Variables are always in the form '$ varName' and could possibly be wrapped, ie. '${varName}'. This information has been collected by an direct analysis of the code and by means of the few comments inserted. The actual value of the variables found in the string is retrieved using the 'getVariablesValue()' method(also defined in SSIMediator.java).

```
    StringBuilder sb = new StringBuilder(val);
    int charStart = sb.indexOf("&#");
```

```
        while (charStart > -1) {
        int charEnd = sb.indexOf(";", charStart);
            if (charEnd > -1) {
                char c = (char) Integer.parseInt(
                        sb.substring(charStart + 2, charEnd));
                sb.delete(charStart, charEnd + 1);
                sb.insert(charStart, c);
        charStart = sb.indexOf("&#");
            } else {
                break;
            }
        }
```

In this block of code the methods finds the position of the first ‘$, eventually escaped, in the string.

```
            // Find the next $
            for (; i < sb.length(); i++) {
              if (sb.charAt(i) == '$') {
                    i++;
                    break;
                }
            }
            if (i == sb.length()) break;
            // Check to see if the $ is escaped
        if (i > 1 && sb.charAt(i - 2) == '\\') {
        sb.deleteCharAt(i - 2);
                i--;
                continue;
            }
```

In this block of code the method identifies the SSI variable name considering also the possibility that the variable name could be wrapped in curly braces.

```
            int nameStart = i;
            int start = i - 1;
            int end = -1;
            int nameEnd = -1;
            char endChar = ' ';
            // Check for {} wrapped var
        if (sb.charAt(i) == '{') {
                nameStart++;
                endChar = '}';
            }
            // Find the end of the var reference
            for (; i < sb.length(); i++) {
                if (sb.charAt(i) == endChar) break;
            }
            end = i;
            nameEnd = end;
            if (endChar == '}') end++;
```

In this block of code the method replaces the variable name with its actual value using the substituteVariables() method.

```java
                // We should now have enough to extract the var name
                String varName = sb.substring(nameStart, nameEnd);
                String value = getVariableValue(varName);
                if (value == null) value = "";
                // Replace the var name with its value
                sb.replace(start, end, value);
                // Start searching for the next $ after the value
                // that was just substituted.
                i = start + value.length();
            }
            return sb.toString();
```

4 Issues

In this section is included a list of problems found during the inspection of the assigned code.

4.1 SSIServlet.java

General Considerations
In general the class lacks of documentation: comments and javadoc are not complete and where inserted are sometimes meaningless and very short.
All the member(internal) variables are declared as 'protected', it is always preferable to use the 'private' access modifier(see checklist 28).

- *init()*
  - Checklist[11]: All the if statements present in the body of this method do not use curly braces
  - Checklist[23]: The javadoc written for this method is not sufficient to explain its role and its behavior in the context of the SSIServlet class
  - Checklist[40]: All the comparisons present in the body of this method use improper operators, in fact the elements in comparison are always objects(strings in particular)
  - Checklist[18]: None of the instructions present in the body of this method is commented. This may be correct if all these instructions are self explicative, but at least the last if statement needs comments to explain what it tries to achieve
  - Checklist[14]: The following lines:

```java
            debug = Integer.parseInt(getServletConfig().
                getInitParameter("debug"));
```

```
116                Boolean.parseBoolean(getServletConfig().
                       getInitParameter("isVirtualWebappRelative"));
```

```
119                expires = Long.valueOf(getServletConfig().
                       getInitParameter("expires"));
```

```
121            buffered = Boolean.parseBoolean(getServletConfig().
                   getInitParameter("buffered"));
```

```
126                outputEncoding = getServletConfig().getInitParameter
                       ("outputEncoding");
```

exceed the length of 80 characters

- Checklist[8,9]: The indentation of lines is made using tabs and not spaces
- Checklist[52,53]: In the following lines

```
119                expires = Long.valueOf(getServletConfig().
                       getInitParameter("expires"));
```

  the Long.valueOf method throws a NumberFormatException which is not managed and must be imported

```
113                debug = Integer.parseInt(getServletConfig().
                       getInitParameter("debug"));
```

  The Integer.parseInt method throws a NumberFormatException which is not managed and must be imported.

- ***requestHandler()***
  - Checklist[8,9]: All indentations in the class are made using tabs and not spaces
  - Checklist[11]: The following conditional block

```
185            if (debug > 0)
186                log("SSIServlet.requestHandler()\n" + "Serving "
187                        + (buffered?"buffered ":"unbuffered ") + "
                               resource '"
188                        + path + "'");
```

  uses no enclosing braces

  - Checklist[18]: No comments from line 210 to the end of the function
  - Checklist[29,33]: The following declarations of variables in lines

```
197            URL resource = servletContext.getResource(path);
```

```
203            String resourceMimeType = servletContext.getMimeType(
                   path);
```

  should be placed in top of the body of the method.

  - Checklist[40]: The comparisons presen following lines

```
191          if (path == null || path.toUpperCase(Locale.ENGLISH).
                 startsWith("/WEB-INF")
```

```
198          if (resource == null) {
```

```
204          if (resourceMimeType == null) {
```

```
208          if (expires != null) {
```

uses improper operator in fact the elements in comparison are object(‘==‘ instead of ‘equals()‘)

- Checklist[52,53]: The following line

```
197          URL resource = servletContext.getResource(path);
```

may throw a ‘MalformedURLException‘, neither managed nor re-thrown

- Checklist[15]: Wrong line breaking in the following lines

```
191          if (path == null || path.toUpperCase(Locale.ENGLISH).
                 startsWith("/WEB-INF")
192                      || path.toUpperCase(Locale.ENGLISH).startsWith("
                             /META-INF")) {
```

- ***processSSI()***
  - Checklist[23]: No javadoc has been written for this method
  - Checklist[28]: The method has the proctected access modifier but it is never invoked from its subclasses.
    It should be declared as ‘private‘
  - Checklist[18]: None of the instructions present in the body of this method is commented. This may be correct if all these instructions are self explicative, but most of the instructions present in this method are not self explicative
  - Checklist[40]: All the comparisons present in the body of this method use improper operators, in fact the elements in comparison are always objects(strings in particular)
  - Checklist[29,33]: In these following lines local variables are defined and assigned to a value. Since these assigments and definitions do not depend from the result of previous instructions, they must be put in the top of the body of the method

```
233          URLConnection resourceInfo = resource.openConnection();
234          InputStream resourceInputStream = resourceInfo.
                 getInputStream();
235          String encoding = resourceInfo.getContentEncoding();
```

```
239          InputStreamReader isr;
```

- Checklist[1]: In this line a local variable of the type ”InputStreamReader” is defined. The name of this variable is ”isr” which does not convey any immediate meaning about the role and the use of this variable

```
239         InputStreamReader isr;
```

- Checklist[52,53]: The method openConnection throws an IOException that is not managed
  The method getInputStream throws an IOException that is not managed

```
233         URLConnection resourceInfo = resource.openConnection();
234         InputStream resourceInputStream = resourceInfo.
                getInputStream();
```

  The getWriter method on the HttpServletResponse object throws a IOException, a IllegalStateException, UnsupportedEncodingException which are neither managed nor re-thrown (IllegalStateException)

```
230             printWriter = res.getWriter();
```

```
255             res.getWriter().write(text);
```

- Checklist[58]: The method close should be invoked on the bufferedReader variable and on the isr variable, since these variables are not used anymore in the rest of the method and they represent streams of bytes readers

```
247         long lastModified = ssiProcessor.process(bufferedReader,
248                 resourceInfo.getLastModified(), printWriter);
```

4.2 SSIMediator.java

General Considerations

In general the class lacks of documentation: the javadoc is not complete and many instructions blocks are left with no comments at all.
All the member(internal) variables are declared as ‘protected‘, it is always preferable to use the ‘private‘ access modifier(see checklist 28).

- ***substituteVariables()***
  - Checklist[8,9]: Tabs are used for identation for all the function
  - Checklist[23]: The Javadoc provided for the function is not complete
  - Checklist[11]: No enclosing braces in the following if statements:

```
253       if (val.indexOf('$') < 0 && val.indexOf('&') < 0) return
              val;
```

```
303                 if (sb.charAt(i) == endChar) break;
```

```java
            if (endChar == '}') end++;
```

```java
            if (value == null) value = "";
```

- Checklist[1]: The parameter of the method is named ‘val‘ which which does not convey any immediate meaning about its role and use
- Checklist[15]: Wrong line breaking in the following line

```java
                char c = (char) Integer.parseInt(
                        sb.substring(charStart + 2, charEnd));
```

- Checklist[52,53]: No action are taken in case one of the following lines throws a NullPointerException, which are neither managed nor re-thrown:

```java
        int charStart = sb.indexOf("&#");
```

```java
        int charEnd = sb.indexOf(";", charStart);
```

```java
        charStart = sb.indexOf("&#");
```

- Checklist[40]: The following line

```java
            if (value == null) value = "";
```

  uses improper operator in fact the elements in comparison are object(‘==‘ instead of ‘equals()‘)
- Checklist[29,33]: The variables in following lines

```java
            String varName = sb.substring(nameStart, nameEnd);
            String value = getVariableValue(varName);
```

  should be placed at the top of the function

5 Additional Issues

In this section are included additional problems and issues not present in the checklist

5.1 SSIServlet.java

- ***init()***
  - In these lines we have the assignment of properties of the class, and this assignment does not depend from the result of previous instructions. Given that, these instructions should be put in the top of the body of the method

```
115         isVirtualWebappRelative =
116             Boolean.parseBoolean(getServletConfig().
                    getInitParameter("isVirtualWebappRelative"));
```

```
121         buffered = Boolean.parseBoolean(getServletConfig().
                getInitParameter("buffered"));
```

```
123         inputEncoding = getServletConfig().getInitParameter("
                inputEncoding");
```

  - In the body of this method continuous calls to the methods of the object returned by the getServletConfig() method are performed. This is inefficient since the above mentioned object can be stored in a local variable and so made accessible without method calls
  - Methods and properties of the superclass of a class must be referenced by that class using the "super." prefix. This should be done for reasons of clarity and readability, so that the developer can immediately distinguish the manipulation of methods and properties of the superclass of the current class from the manipulation of those which belong to the current class.
    This behavior is not followed in this method(all lines)

- ***requestHandler()***
  - Methods and properties of the superclass of a class must be referenced by that class using the "super" prefix. This should be done for reasons of clarity and readability, so that the developer can immediately distinguish the manipulation of methods and properties of the superclass of the current class from the manipulation of those which belong to the current class.
    This behavior is not followed in lines:

```
183         ServletContext servletContext = getServletContext();
```

    And in all lines that present the invocation of ‘log()‘
  - Methods and properties of the current class must be referenced within the class using the "this." prefix. This should be done for reasons of clarity and readability, so that the developer can immediately distinguish the manipulation of methods and properties of the current class from the manipulation of

those which belong to the superclass of the current class.
This is also useful for distinguish the manipulation of properties of the current class and local variables. This behavior is not followed in lines:

```
185         if (debug > 0)
186             log("SSIServlet.requestHandler()\n" + "Serving "
187                     + (buffered?"buffered ":"unbuffered ") + "
                            resource '"
188                     + path + "'");
```

```
207         res.setContentType(resourceMimeType + ";charset=" +
                outputEncoding);
208         if (expires != null) {
```

- In these lines the ternary operator ? is used. The expression is syntactically valid but the use of ? makes it counter intuitive and less readable.
  It is preferable to use a classic if-else block.

```
186             log("SSIServlet.requestHandler()\n" + "Serving "
187                     + (buffered?"buffered ":"unbuffered ") + "
                            resource '"
188                     + path + "'");
```

- ***processSSI()***
  - Methods and properties of the superclass of a class must be referenced by that class using the "super" prefix. This should be done for reasons of clarity and readability, so that the developer can immediately distinguish the manipulation of methods and properties of the superclass of the current class from the manipulation of those which belong to the current class.
    This behavior is not followed in this method(all lines).
  - Methods and properties of the current class must be referenced within the class using the "this." prefix. This should be done for reasons of clarity and readability, so that the developer can immediately distinguish the manipulation of methods and properties of the current class from the manipulation of those which belong to the superclass of the current class.
    This is also useful for distinguish the manipulation of properties of the current class and local variables. This behavior is not followed in lines:

```
220             new SSIServletExternalResolver(getServletContext(),
                    req, res,
221                     isVirtualWebappRelative, debug,
                            inputEncoding);
222         SSIProcessor ssiProcessor = new SSIProcessor(
                ssiExternalResolver)
223                 debug);
```

```
226         if (buffered) {
```

```
252         if (buffered) {
```

- In the following two lines of code each statement declares a variable and then assigns to it a ‘null‘ value. In general assign a ‘null‘ value to a fresh declared variable is an useless operation, since this is the default behavior of Java.

```java
        PrintWriter printWriter = null;
        StringWriter stringWriter = null;
```

- In the following block of code the first if statement is redundant, it should be deleted and its contents copied into the body of the second if statement.

```java
        if (encoding == null) {
            encoding = inputEncoding;
        }
        InputStreamReader isr;
        if (encoding == null) {
            isr = new InputStreamReader(resourceInputStream);
        } else {
            isr = new InputStreamReader(resourceInputStream,
                encoding);
        }
```

- [**CRITICAL**]In these following lines:

```java
        URLConnection resourceInfo = resource.openConnection();
        InputStream resourceInputStream = resourceInfo.
            getInputStream();
```

  The getInputStream and getContentEncoding methods cannot be called on a ”URLConnection” object before opening an actual connection to the resource referred by object itself (to solve this problem the ”connect” method must be called on the ”URLConnection” object before calling getInputStream, see the javadoc of the URL class)

5.2 SSIMediator.java

No particular additional issues has been found in the ‘substituteVariables()‘ method.

6 Appendix

6.1 References

- javaCheckList.pdf: Contains the check list used to inspect the code present in this document.

6.2 Tools Used

- Atom/ LaTeX: To redact this document
- Eclipse: To simulate the behavior of the assigned code

6.3 Hours of Work

- Andrea Sessa: 14.5 hours
- Giorgio Pea: 13 hours