Inspection Document

Version 1.0

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

5/1/2016

Contents

1 Introduction

2 Classes

Included in this section the two java classes subjected to the 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

In this section are included some information about the functioning of tha analyzed classes and methods.

3.1   SSIServlet.java

From the Javadoc:

```
78  /**
79   * Servlet to process SSI requests within a webpage. Mapped to a path
         from
80   * within web.xml.
81   *
```

This class represents a Java EE servlet used to process requests that include some SSI instruction.
SSI(Server Side Include) that is a simple interpreted server-side scripting language. The most frequent use of SSI is to include the contents of one or more files into a web page on a web server.

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

  ```
  104      /**
  105       * Initialize this servlet.
  106       *
  107       * @exception ServletException
  108       *                if an error occurs
  109       */
  ```

  It is clear that the method take care of initalize the Java EE servlet by retrieving the configuration parameters(through the methods provided by the superclass: 'GenericServlet')
  Follows an explanation of the initialization parameters(obtained by analizing the comments provided with the method):

  - **debug**: Specifies the debug level of the servlet, if 0 no debug message are logged
  - **isVirtualWebappRelative**: Specifies if the paths can be webapp relative
  - **expires**: Specifies the expiration time of this servlet(in seconds)
  - **buffered**: Specifies if the output response of the serverlet should be buffered first or not(see processSSI() for more details)
  - **inputEncoding**: Specifies the encoding of the HttpServletRequest
  - **outputEncoding**: Specifies the encoding of the HttpServletResponse

- **requestHandler()**
  From the inspection of the code this function is only called when the serverlet receives a HTTP Get or Post request. The javadoc for the method, included in the code, states:

```
173    /**
174     * Process our request and locate right SSI command.
175     *
176     * @param req
177     *          a value of type 'HttpServletRequest'
178     * @param res
179     *          a value of type 'HttpServletResponse'
180     */
```

  Hence the method accepts as parameters a HttpServletRequest,the incoming request, and a HttpServletResponse that is a reference to the response.

  Now the objective of the method is to retrieve the correct resource from the serverlet context. If the debug level is greater than zero then log a message into the logger for debug purposes.

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

  The comment is very clear: it checks if the resource is either in the 'WEB-INF' or 'META-INF' subdirectories; if so the function return with an error code.

```
189        // Exclude any resource in the /WEB-INF and /META-INF
               subdirectories
190        // (the "toUpperCase()" avoids problems on Windows systems)
191        if (path == null || path.toUpperCase(Locale.ENGLISH).
               startsWith("/WEB-INF")
192                || path.toUpperCase(Locale.ENGLISH).startsWith("/
                    META-INF")) {
193            res.sendError(HttpServletResponse.SC_NOT_FOUND, path);
194            log("Can't serve file: " + path);
195            return;
196        }
```

  Here the function tries to retrieve the URL to the resource; it also performs an existence check on the resource, if the resource doesn't exist the function return an error.

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

4

```
201                    return;
202             }
```

In the final part, the function starts to initialize the header of the HttpServletResponse by setting: the mime type, the encoding of the output text and the expiration time for the response(in seconds, see init()).

Finally the processSSI() function is invoked passing as parameters the original request, the reference to the response and the resource.

```
203         String resourceMimeType = servletContext.getMimeType(path);
204         if (resourceMimeType == null) {
205             resourceMimeType = "text/html";
206         }
207         res.setContentType(resourceMimeType + ";charset=" +
                outputEncoding);
208         if (expires != null) {
209             res.setDateHeader("Expires", (new java.util.Date()).
                    getTime()
210                     + expires.longValue() * 1000);
211         }
212         req.setAttribute(Globals.SSI_FLAG_ATTR, "true");
213         processSSI(req, res, resource);
```

- *processSSI()*

3.2   SSIMediator.java

From the Javadoc of the class:

```
75  /**
76   * Allows the different SSICommand implementations to share data/talk to
         each
77   * other
78   *
```

The class is inserted into the context of SSI processing, in particular this class take care of how many different implementations of the SSI instructions can communicate and exchange data with each other.
Follows a detailed description of the assigned methods:

- *substituteVariables()*

```
246      /**
247       * Applies variable substitution to the specified String and
             returns the
248       * new resolved string.
249       */
```

The method accepts as parameter a string and returns a new string to which a variables substitution process has been applied.

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

It checks if the string contains '$' or '&', if not there is nothing to substitute so the original string is simply returned. Otherwise:

```
253        if (val.indexOf('$') < 0 && val.indexOf('&') < 0) return val;
254
255        // HTML decoding
256        val = val.replace("&lt;", "<");
257        val = val.replace("&gt;", ">");
258        val = val.replace("&quot;", "\"");
259        val = val.replace("&amp;", "&");
```

It's easy to understand(from the comments and javadoc) that above snippet of code substitute each occurrence of HTML special codes with the real character.

```
261        StringBuilder sb = new StringBuilder(val);
262        int charStart = sb.indexOf("&#");
263        while (charStart > -1) {
264        int charEnd = sb.indexOf(";", charStart);
265            if (charEnd > -1) {
266                char c = (char) Integer.parseInt(
```

```
267                        sb.substring(charStart + 2, charEnd));
268                sb.delete(charStart, charEnd + 1);
269                sb.insert(charStart, c);
270         charStart = sb.indexOf("&#");
271            } else {
272                break;
273            }
274        }
```

This part of the code 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 processes variables and substitutes 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 are retrieved by means of the 'getVariablesValue()' function(also defined in SSIMediator.java).

Find the first '$', eventually escaped.

```
277            // Find the next $
278            for (; i < sb.length(); i++) {
279        if (sb.charAt(i) == '$') {
280                i++;
281                break;
282            }
283        }
284        if (i == sb.length()) break;
285        // Check to see if the $ is escaped
286        if (i > 1 && sb.charAt(i - 2) == '\\') {
287        sb.deleteCharAt(i - 2);
288            i--;
289            continue;
290        }
```

The following code identifies the portion string to substitute [nameStart, nameEnd] and the name of the variable [start, end]. Also the functions consider the possibility that the variable could be wrapped so it processes the presence of '{' and '}' that are wrapping the variable name.

```
291            int nameStart = i;
292            int start = i - 1;
293            int end = -1;
294            int nameEnd = -1;
295            char endChar = ' ';
296            // Check for {} wrapped var
297        if (sb.charAt(i) == '{') {
298                nameStart++;
299                endChar = '}';
300            }
301            // Find the end of the var reference
```

```
302              for (; i < sb.length(); i++) {
303                  if (sb.charAt(i) == endChar) break;
304              }
305              end = i;
306              nameEnd = end;
307              if (endChar == '}') end++;
```

Finally the variable name has been identified in the original string [start, end] and the 'getVariablesValue()' method is called to retrieve the value of the variable. The value is then substituted and the function seeks for the presence of other variables. If no more variables are found, the function returns the processed string.

```
308              // We should now have enough to extract the var name
309              String varName = sb.substring(nameStart, nameEnd);
310              String value = getVariableValue(varName);
311              if (value == null) value = "";
312              // Replace the var name with its value
313              sb.replace(start, end, value);
314              // Start searching for the next $ after the value
315              // that was just substituted.
316              i = start + value.length();
317          }
318          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.

- *init()*

    1. Checklist[11]: All the if statements present in the body of this method do not use curly braces
    2. 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
    3. 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)
    4. 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

8

5. Checklist[14]: Lines

```
113             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

6. Checklist[8,9]: The indentation of lines is made using tabs and not spaces

7. Checklist[52,53]: In the line

```
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()**

  1. Checklist[8,9]: All indentations in the class are made by means of tabs

  2. Checklist[11]: The conditional block

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

uses no enclosing braces

  3. Checklist[18]: No comments from line 210 to the end of the function

  4. Checklist[29,33]: The declarations of variables in lines

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

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

9

should be placed at the start of the function block

5. Checklist[40]: The 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 for comparation '==' instead of 'equals()'

6. Checklist[52,53]: The line

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

may throws a 'MalformedURLException', neither actions are taken to manage the exception nor the exception is explicitly re-thrown

7. Checklist[15]: Wrong line breaking in

```
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[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(== or !== instead of equals or !..equals), in fact the elements in comparison are always objects(strings in particular)
  - Checklist[29,33]: In these lines

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

239          InputStreamReader isr;
```

    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
  - Checklist[1]:

```
239          InputStreamReader isr;
```

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

– Checklist[52,53]:

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

The method openConnection throws an IOException that is not managed
The method getInputStream throws an IOException that is not managed

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

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

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

– Checklist[58]:

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

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

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.

- *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;
```

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

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

11

- Checklist[1]: The parameter of the function is named 'val' which is not meaningfull to understand its role in the function execution

- Checklist[15]: Wrong line breking in the line

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

- Checklist[52,53]: No action are taken in case one of the following lines throws a NullPointerException:

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

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

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

- Checklist[40]: The line

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

  uses for comparation '==' instead of 'equals()'

- Checklist[33]: The variables in lines

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

  should be placed at the top of the function

5 Additional Issues

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

5.1 SSIServlet.java

- *init()*

  - In these lines:

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

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

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

    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

  - In the body of this method continuos 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 lines

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

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.

- **_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:

```
224          PrintWriter printWriter = null;
225          StringWriter stringWriter = null;
```

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, indeed this is the default behavior of Java.

– In the following block of code:

```
236          if (encoding == null) {
237              encoding = inputEncoding;
238          }
239          InputStreamReader isr;
240          if (encoding == null) {
241              isr = new InputStreamReader(resourceInputStream);
242          } else {
243              isr = new InputStreamReader(resourceInputStream,
                    encoding);
244          }
```

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

– In the following line:

```
233          URLConnection resourceInfo = resource.openConnection();
234          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: xxx hours

- Giorgio Pea: xxx hours