Inspection

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 analisys.

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:

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

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

  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 servelet context. If the debug level is greater than zero then log a message into the logger for debug purposes.

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

```java
                    + 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.

```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;
        }
```

Here the fucntion 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.

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

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.

```java
        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()*

3.2 SSIMediator.java

From the Javadoc of the class:

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

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()*

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

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

```
        // If it has no references or HTML entities then no work
        // need to be done
        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:

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

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

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

```
        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;
            }
        }
```

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 proccesses variables and substitutes their current value.
Variables are always in the form '$ varName' and could possibly be wrapped, ie. '${varName}'. This information has been collected by an direct analisys of the code and by means of the few comments inserted. The actual value of the variables found in the string are retrieved by means of the 'getVariablesValue()' function(also defined in SSIMediator.java).
Find the first '$', eventually escaped.

```
            // 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;
            }
```

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 proccesses the presence of '{' and '}' that are wrapping the bariable name.

```
            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
```

```
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 then 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 ispection 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()*
- *requestHandler()*

  1. Checklist[8,9]: All indentations in the class are made by means of tabs
  2. Checklist[18]: The function is not fully commented, some instructions(lines 197 to 213) are not commented at all
  3. 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

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

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

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

   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 rethrown

- *processSSI()*

4.2 SSIMediator.java

- *substituteVariables()*

5 Additional Considerations

6 Appendix

6.1 Java Checklist

6.2 Statistics