

TABLE OF CONTENTS (SHOW)

Java Server-side Programming

Getting started with JSP by Examples

1. Introduction

JavaServer Page (JSP) is Java's answer to the popular Microsoft's *Active Server Pages (ASP)*. JSP, like ASP, provides a simplified and fast mean to generate *dynamic* web contents. It allows you to mix *static* HTML with *dynamically generated* HTML - in the way that the *business logic* and the *presentation* are well separated.

The advantages of JSP are:

1. **Separation of static and dynamic contents:** JSP enables the separation of *static* contents from *dynamic* contents. The *dynamic* contents are generated via programming logic and inserted into the *static template*. This greatly simplifies the creation and maintenance of web contents.
2. **Reuse of components and tag libraries:** The *dynamic* contents can be provided by reusable components such as JavaBean, Enterprise JavaBean (EJB) and tag libraries - you do not have to re-inventing the wheels.
3. **Java's power and portability**

JSPs are Internally Compiled into Java Servlets

That is to say, anything that can be done using JSPs can also be accomplished using Java servlets. However, it is important to note that servlets and JSPs are *complementary* technologies, NOT replacement of each other. Servlet can be viewed as "**HTML Inside Java**", which is better for implementing business logic - as it is Java dominant. JSP, on the other hand, is "**Java Inside HTML**", which is superior for creating presentation - as it is HTML dominant. In a typical *Model-View-Control (MVC)* application, servlets are often used for the Controller (C), which involves complex programming logic. JSPs are often used for the View (V), which mainly deals with presentation. The Model (M) is usually implemented using JavaBean or EJB.

Apache Tomcat Server

JSPs, like servlets, are server-side programs run inside a HTTP server. To support JSP/servlet, a Java-capable HTTP server is required. Tomcat Server (@ <http://tomcat.apache.org>) is the official reference implementation (RI) for Java servlet and JSP, provided free by Apache (@ <http://www.apache.org>) - an open-source software foundation.

2. First JSP Example - "**Java inside HTML**"

Let's begin with a simple JSP example. We shall use the webapp called "hello" that we have created in our earlier exercise. Use a programming text editor to enter the following HTML/JSP codes and save as "first.jsp" (the file type of ".jsp" is mandatory) in your webapp (web context) home directory (i.e., "webapps\hello".

```

1 <html>
2 <head><title>First JSP</title></head>
3 <body>
4   <%
5     double num = Math.random();
6     if (num > 0.95) {
7       %>
8       <h2>You'll have a luck day!</h2><p>(<%= num %>)</p>
9     <%
10    } else {
11      %>
12      <h2>Well, life goes on ... </h2><p>(<%= num %>)</p>
13    <%
14    }
15    %>
16    <a href="<%= request.getRequestURI() %>"><h3>Try Again</h3></a>
17  </body>
18 </html>
```

To execute the JSP script: Simply start your Tomcat server and use a browser to issue an URL to browse the JSP page (i.e., <http://localhost:8080/hello/first.jsp>).

From your browser, choose the "View Source" option to check the response message. It should be either of the followings depending on the random number generated.

```
<html>
<h2>You'll have a luck day!</h2>
<p>(0.987)</p>
<a href="first.jsp"><h3>Try Again</h3></a>
</html>
```

```
<html>
<h2> Well, life goes on ... </h2>
<p>(0.501)</p>
<a href="first.jsp"><h3>Try Again</h3></a>
</html>
```

It is important to note that the client is not able to "view" the original JSP script (otherwise, you may have security exposure), but merely the result generated by the script.

Explanations

1. A JSP script is a regular HTML page containing Java programs. Recall that JSP is "**Java inside HTML**" (whereas servlet is "**HTML inside Java**"). The Java statements are enclosed by <% ... %> (called *JSP scriptlet*) or <%= ... %> (called *JSP expression*).
2. *JSP Scriptlet* <% ... %> is used to include Java statements.

3. JSP Expression <%= ... %> is used to evaluate a single Java expression and display its result.

4. The method `request.getRequestURI()` is used to retrieve the URL of the current page. This is used in the anchor tag <a> for refreshing the page to obtain another random number.

Behind the Scene

When a JSP is first accessed, Tomcat converts the JSP into a servlet; compile the servlet, and execute the servlet. Check out the generated servlet for "first.jsp", and study the JSP-to-servlet conversion. Look under Tomcat's "work\Catalina\localhost\hello" for "first_jsp.java".

The relevant part of the generated servlet is extracted as follows (with some simplifications):

```
1 out.write("<html>\r\n  ");
2 double num = Math.random();
3 if (num > 0.95) {
4     out.write("<h2>You will have a luck day!");
5     out.write("</h2><p>(");
6     out.print( num );
7     out.write("</p>\r\n");
8 } else {
9     out.write("\r\n    ");
10    out.write("<h2>Well, life goes on ... ");
11    out.write("</h2><p>(");
12    out.print( num );
13    out.write("</p>\r\n  ");
14 }
15 out.write("<a href=\"\"");
16 out.print( request.getRequestURI() );
17 out.write("\">");
18 out.write("<h3>Try Again</h3></a>\r\n");
19 out.write("</html>\r\n");
```

Explanation

1. The HTML statements are written out as part of the response via `out.write()`, as "it is".
2. The JSP scriptlets <% ... %> are kept, as "it is", in the converted servlet as the program logic.
3. The JSP expressions <%= ... %> are placed inside a `out.print()`. Hence, the expression will be evaluated, and the result of the evaluation written out as part of the response message.

Compare the JSP script and the internally generated servlet, you shall understand that servlet is "*HTML inside Java*", whereas JSP is "*Java inside HTML*".

Subsequent accesses to the same JSP will be much faster, because they will be re-directed to the converted and compiled servlet directly (no JSP-to-servlet conversion and servlet compilation needed again), unless the JSP has been modified.

3. Revisit Java Servlets

A typical Java servlet (as shown below) contains three kinds of methods: `init()`, `destroy()`, and one or more `service()` methods such as `doGet()` and `doPost()`. `init()` runs when the servlet is loaded. `destroy()` runs when the servlet is unloaded. `service()` runs once per HTTP request. The `service()` methods takes two arguments: `request` and `response`, corresponding to the HTTP request and response messages respectively. A `PrintWriter` called `out` is created for writing out the response to the network.

```
1 import java.io.*;
2 import javax.servlet.*;
3 import javax.servlet.http.*;
4
5 public class ...Servlet extends HttpServlet {
6
7     // Runs when the servlet is loaded onto the server.
8     public void init() {
9         .....
10    }
11
12     // Runs on a thread whenever there is HTTP GET request
13     // Take 2 arguments, corresponding to HTTP request and response
14     public void doGet(HttpServletRequest request, HttpServletResponse response)
15         throws IOException, ServletException {
16
17         // Set the MIME type for the response message
18         response.setContentType("text/html");
19         // Write to network
20         PrintWriter out = response.getWriter();
21
22         // Your servlet's logic here
23         out.println("<html>");
24         out.println( ..... );
25         out.println("</html>");
26    }
27
28     // Runs as a thread whenever there is HTTP POST request
29     public void doPost(HttpServletRequest request, HttpServletResponse response)
30         throws IOException, ServletException {
31         // do the same thing as HTTP GET request
32         doGet(request, response);
33    }
34
35     // Runs when the servlet is unloaded from the server.
36     public void destroy() {
37         .....
38    }
39
40     // Other instance variables and methods
41 }
```

Java servlet produces HTML codes by calling `out.print()` methods. You have to hardcode all the HTML tags (and cannot use any WYSIWYG web authoring tools). Any change to the web page's

4. Second JSP example - Echoing HTML Request Parameters

Enter the following JSP script and save as "echo.jsp" in your webapp's root directory.

```
1 <html>
2 <head>
3   <title>Echoing HTML Request Parameters</title>
4 </head>
5 <body>
6   <h3>Choose an author:</h3>
7   <form method="get">
8     <input type="checkbox" name="author" value="Tan Ah Teck">Tan
9     <input type="checkbox" name="author" value="Mohd Ali">Ali
10    <input type="checkbox" name="author" value="Kumar">Kumar
11    <input type="submit" value="Query">
12  </form>
13
14  <%
15  String[] authors = request.getParameterValues("author");
16  if (authors != null) {
17    %>
18    <h3>You have selected author(s):</h3>
19    <ul>
20      <%
21        for (int i = 0; i < authors.length; ++i) {
22          %>
23          <li><%= authors[i] %></li>
24        <%
25      }
26    <%
27    </ul>
28    <a href="<%= request.getRequestURI() %>">BACK</a>
29  <%
30  }
31 <%
32 </body>
33 </html>
```

Browse the JSP page created and study the generated servlet.

Explanations

1. This HTML page has a form with 3 checkboxes. The "name=value" pair of the checkboxes is "author=so_and_so". No "action" attribute is specified, the default "action" is the current page (i.e. the query will be sent to the same page).
2. The JSP scriptlet checks if the query parameter "author" exists to decide whether to dynamically generate the enclosed codes. "author" parameter is absent when the page is first requested. Once the client fills in the form (by checking the boxes) and submits the form, "author" will be present in the HTTP request, and submitted to the *same* page for processing (with the default <form>'s "action" attribute).
3. The `request.getParameterValues()` is used to retrieve all the values of the query parameter. The values are echoed back using an unordered list.

5. JSP Scripting Elements

JSP provides the following scripting elements:

- JSP Comment <%-- comments -->
- JSP Expression <%= Java Expression %>
- JSP Scriptlet <% Java Statement(s) %>
- JSP Directive <%@ page|include ... %>

To simplify the access of the HTTP *request* and *response* messages, JSP has *pre-defined* the following variables:

- `request`: corresponds to the HTTP request message.
- `response`: corresponds to the HTTP response message.
- `out`: corresponds to the HTTP response message's output stream.
- others such as session, page, application, pageContext, which are outside the scope of this tutorial.

5.1 JSP comment <%-- comments -->

JSP comments <%-- comments --%> are ignored by the JSP engine. For example,

```
<%-- anything but a closing tag here will be ignored -->
```

Note that HTML comment is <!-- comments -->. JSP expression within the HTML comment will be evaluated. For example,

```
<!-- HTML comments here <%= Math.random() %> more comments -->
```

5.2 JSP Expression <%= Java Expression %>

JSP Expression can be used to insert a *single* Java expression directly into the response message. This expression will be placed inside a `out.print()` method. Hence, the expression will be evaluated and printed out as part of the response message. Any valid Java expression can be used. There is no semi-colon at the end of the expression. For examples:

```
<p>The square root of 5 is <%= Math.sqrt(5) %></p>
<h5><%= item[10] %></h5>
<p>Current time is: <%= new java.util.Date() %></p>
```

The above JSP expressions will be converted to:

```

out.write("<p>The square root of 5 is ");
out.print( Math.sqrt(5) );
out.write("</p>");
out.write("<h5>");
out.print( item[10] );
out.write("</h5>");
out.write("<p>Current time is: ");
out.print( new java.util.Date() );
out.write("</p>");

```

You can use the pre-defined variables, such as `request`, in the expressions. For examples:

```

<p>You have choose author <%= request.getParameter("author") %></p>
<%= request.getRequestURI() %>
<%= request.getHeader("Host") %>

```

5.3 JSP Scriptlet <% Java statement(s) %>

JSP scriptlets allow you to do more *complex operations* than inserting a single Java expression (with the JSP expression). JSP scriptlets let you insert an arbitrary sequence of valid Java statement(s) into the `service()` method of the converted servlet. All the Java statements in a scriptlet are to be terminated with a semi-colon. For example:

```

<%
String author = request.getParameter("author");
if (author != null && !author.equals("")) {
%>
<p>You have choose author <%= author %></p>
<%
}
%>

```

In the converted servlet, the above will be inserted into the `service()` method as follows:

```

String author = request.getParameter("author");
if (author != null && !author.equals("")) {
    out.write("<p>You have choose author ");
    out.print( author );
    out.write("</p>");
}

```

Notice that the Java codes inside a scriptlet are inserted exactly as they are written, and used as the programming logic. The HTML codes are passed to an `out.write()` method and written out as part of the response message.

5.4 JSP Directive <%@ page|include ... %>

JSP directives provide instructions to the JSP engine. The syntax of the JSP directive is:

```

<%@ directive_name
attribute1="value1"
attribute2="value2"
...
attributeN="valueN" %>

```

JSP page Directive

The "page" directive lets you import classes and customize the page properties. For examples,

```

<!-- import package java.sql.* -->
<%@ page import="java.sql.*" %>

<!-- Set the output MIME type -->
<%@ page contentType="image/gif" %>

<!-- Set an information message for getServletInfo() method -->
<%@ page info="Hello-world example" %>

```

JSP include Directive

The "include" directive lets you include another file(s) at the time when the JSP page is compiled into a servlet. You can include any JSP files, or static HTML files. You can use `include` directive to include navigation bar, copyright statement, logo, etc. on every JSP pages. The syntax is:

```
<%@ include file="url" %>
```

For example:

```

<%@ include file="header.html" %>
.....
<%@ include file="footer.html" %>

```

6. JSP Database Example

Let's revisit our e-shop, which was created using Java Servlet.

Database

Database: ebookshop				
Table: books				
id	title	author	price	qty
(INT)	(VARCHAR(50))	(VARCHAR(50))	(FLOAT)	(INT)
1001	Java for dummies	Tan Ah Teck	11.11	11
1002	More Java for dummies	Tan Ah Teck	22.22	22

1003 More Java for more dummies Mohammad Ali 33.33 33
1004 A Cup of Java Kumar 44.44 44
1005 A Teaspoon of Java Kevin Jones 55.55 55

Querying - "query.jsp"

Let's create the query page called "query.jsp".

```

1  <html>
2  <head>
3      <title>Book Query</title>
4  </head>
5  <body>
6      <h1>Another E-Bookstore</h1>
7      <h3>Choose Author(s):</h3>
8      <form method="get">
9          <input type="checkbox" name="author" value="Tan Ah Teck">Tan
10         <input type="checkbox" name="author" value="Mohd Ali">Ali
11         <input type="checkbox" name="author" value="Kumar">Kumar
12         <input type="submit" value="Query">
13     </form>
14
15    <%
16        String[] authors = request.getParameterValues("author");
17        if (authors != null) {
18    %>
19    <%@ page import = "java.sql.*" %>
20    <%
21        Connection conn = DriverManager.getConnection(
22            "jdbc:mysql://localhost:8888/ebookshop", "myuser", "xxxx"); // <== Check!
23        // Connection conn =
24        //     DriverManager.getConnection("jdbc:odbc:eshopODBC"); // Access
25        Statement stmt = conn.createStatement();
26
27        String sqlStr = "SELECT * FROM books WHERE author IN (";
28        sqlStr += "" + authors[0] + ""; // First author
29        for (int i = 1; i < authors.length; ++i) {
30            sqlStr += ", " + authors[i] + ""; // Subsequent authors need a leading comma
31        }
32        sqlStr += ") AND qty > 0 ORDER BY author ASC, title ASC";
33
34        // for debugging
35        System.out.println("Query statement is " + sqlStr);
36        ResultSet rset = stmt.executeQuery(sqlStr);
37    %>
38    <hr>
39    <form method="get" action="order.jsp">
40        <table border=1 cellpadding=5>
41            <tr>
42                <th>Order</th>
43                <th>Author</th>
44                <th>Title</th>
45                <th>Price</th>
46                <th>Qty</th>
47            </tr>
48        <%
49            while (rset.next()) {
50                int id = rset.getInt("id");
51        %>
52            <tr>
53                <td><input type="checkbox" name="id" value="<%= id %>"></td>
54                <td><%= rset.getString("author") %></td>
55                <td><%= rset.getString("title") %></td>
56                <td>$<%= rset.getInt("price") %></td>
57                <td><%= rset.getInt("qty") %></td>
58            </tr>
59        <%
60    }
61    <%
62        </table>
63        <br>
64        <input type="submit" value="Order">
65        <input type="reset" value="Clear">
66    </form>
67    <a href="<%= request.getRequestURI() %>"><h3>Back</h3></a>
68    <%
69        rset.close();
70        stmt.close();
71        conn.close();
72    %}
73    <%
74    </body>
75    </html>
```

Explanations

1. This HTML page has a form with 3 checkboxes. The "name=value" pair of the checkboxes is "author=so_and_so". No "action" attribute is specified, hence, it defaulted to current page. The processing script is contained in the *same* page.
2. The method `request.getParameter("author")` is used to check if the query parameter "author" exists. "author" is absent during the first reference of the page.

3. The <%@ page ... %> contains a JSP "page" directive to import the java.sql package.
4. The scriptlet performs the database query operation. The steps are:
 - a. Establish a database connection via a java.sql.Connection object;
 - b. Allocate a java.sql.Statement object under the Connection;
 - c. Prepare a SQL SELECT string;
 - d. Execute the SQL SELECT using executeQuery() method. The result of query is returned in an object of java.sql.ResultSet;
 - e. Process the ResultSet row by row via ResultSet.next();
 - f. Free resources and close the Connection.
5. The query result is tabulated in a HTML table. Note the mixing of HTML and Java in producing the table.

Notice that JSP carries out the presentation much better and neater than servlet. The presentation can be changed easily with JSP. The JSP pages can be created and modified using a WYSIWYG web authoring tool and reload to see the effect on the presentation. Whereas, in the case of servlet, you have to *explicitly* code all the HTML tags inside the servlet program and re-compile the program.

Ordering - "order.jsp"

Let's write the "order.jsp" for processing the order, by updating the appropriate records in the database.

```

1  <html>
2  <head>
3      <title>Order Book</title>
4  </head>
5
6  <body>
7      <h1>Another E-Bookstore</h1>
8      <h2>Thank you for ordering...</h2>
9
10     <%
11         String[] ids = request.getParameterValues("id");
12         if (ids != null) {
13             %>
14             <%@ page import = "java.sql.*" %>
15             <%
16                 Connection conn = DriverManager.getConnection(
17                     "jdbc:mysql://localhost:8888/ebookshop", "myuser", "xxxx"); // <== Check!
18                 // Connection conn =
19                 //     DriverManager.getConnection("jdbc:odbc:eshopODBC"); // Access
20                 Statement stmt = conn.createStatement();
21                 String sqlStr;
22                 int recordUpdated;
23                 ResultSet rset;
24             %>
25                 <table border=1 cellpadding=3 cellspacing=0>
26                     <tr>
27                         <th>Author</th>
28                         <th>Title</th>
29                         <th>Price</th>
30                         <th>Qty In Stock</th>
31                     </tr>
32             <%
33                 for (int i = 0; i < ids.length; ++i) {
34                     // Subtract the QtyAvailable by one
35                     sqlStr = "UPDATE books SET qty = qty - 1 WHERE id = " + ids[i];
36                     recordUpdated = stmt.executeUpdate(sqlStr);
37                     // carry out a query to confirm
38                     sqlStr = "SELECT * FROM books WHERE id = " + ids[i];
39                     rset = stmt.executeQuery(sqlStr);
40                     while (rset.next()) {
41                         %>
42                             <tr>
43                                 <td><%= rset.getString("author") %></td>
44                                 <td><%= rset.getString("title") %></td>
45                                 <td>$<%= rset.getInt("price") %></td>
46                                 <td><%= rset.getInt("qty") %></td>
47                             </tr>
48             <%
49                 rset.close();
50             }
51             stmt.close();
52             conn.close();
53         }
54     %>
55     </table>
56     <a href="query.jsp"><h3>BACK</h3></a>
57 </body>
58 </html>

```

7. JSP Exercises

[TODO]

REFERENCES & RESOURCES

1. JavaServer Pages Technology @ <http://java.sun.com/products/jsp>.

2. Java Servlet Technology @ <http://java.sun.com/products/servlet>.
3. Apache Tomcat @ <http://tomcat.apache.org>, Apache Software Foundation.
4. The Java EE 5 Tutorial @ <http://java.sun.com/javaee/5/docs/tutorial/doc/>.
5. Marty Hall, et al., "Core Servlets and JavaServer Pages", vol.1 (2nd eds, 2003) and vol. 2 (2nd eds, 2006), Prentice Hall.
6. Java Database Connectivity (JDBC) @ <http://java.sun.com/products/jdbc>.
7. RFC2616 "Hypertext Transfer Protocol HTTP 1.1", World-Wide-Web Consortium (W3C), June 1999.
8. "HTML 4.01 Specification", W3C Recommendation, 24 Dec 1999.

Latest version tested: Tomcat 7.0.32, JDK 1.7.0_07

Last modified: October, 2012

Feedback, comments, corrections, and errata can be sent to Chua Hock-Chuan (ehchua@ntu.edu.sg) | [HOME](#)