Java Server Pages

JSP Basics
Advantages of JSP
JSP Design Goals
Sample JSP Page
Sample JSP Code
Components of a JSP Page
JSP Elements
Directives
page Directive Attributes
Include Directive
Comments in JSP
Expressions
Declaratives
Example using Directives
JSP Syntax- Scriptlets
Example Using Scriptlets
Predefined Variables
JSP Predefined Variables

JSP Basics

- Template for a Web Page that uses java code to generate an html document dynamically.
- Java Server Pages (JSP) technology is the Java platform technology for delivering dynamic content to web clients in a portable, secure and well-defined way
- Parallel technology to ASP
- JSPs are run in a server side component known as a JSP container.
- The JSP container translates the JSP into equivalent Java Servlets.
- What is possible in Servlets is largely possible in JSPs too.

Benefits of JSP

- JSP pages have all the advantages of Servlets.
- Its own Advantages
  - Automatically recompiled when necessary
  - Are Html like so they have greater compatibility with Web development tools.
  - It is easier to write and maintain the HTML.
  - You can use standard Web-site development tools.
  - You can divide up your development team.

Sample JSP page

```
<html>
<body>

<h2>Writing simple Jsp</h2> <hr>

Today's date is  <%= new java.util.Date() %>

</body>
</html>
```

How JSP works

- Firstly when a web browser seeks a JSP file through an URL from the web server, the web server recognizes the .jsp file extension in the URL requested by the browser
- Then the web server passes the request to the JSP engine. The JSP page is then translated into a Java class, which is then compiled into a servlet.

The Life Cycle of a JSP Page

- A JSP page services requests as a servlet.
- When a request is mapped to a JSP page, the web container first checks whether the JSP page's servlet is older than the JSP page.
- During development, one of the advantages of JSP pages over servlets is that the build process is performed automatically.

Translation and Compilation

- Directives are used to control how the web container translates and executes the JSP page.
- Scripting elements are inserted into the JSP page's servlet class.
- Expression language expressions are passed as parameters to calls to the JSP expression evaluator.
- jsp:[set|get]Property elements are converted into method calls to JavaBeans components.
- jsp:[include|forward] elements are converted into invocations of the Java Servlet API.
- The jsp:plugin element is converted into browser-specific markup for activating an applet.
- Custom tags are converted into calls to the tag handler that implements the custom tag.

Note :This translation and compilation phase occurs only when the JSP file is requested for the first time, or if it undergoes any changes to the extent of getting retranslated and recompiled.

Sample JSP Expression/Scriptlet

- <H2>foo</H2>
- <%= bar() %>
- <% baz(); %>

Representative Resulting Servlet Code: Expression/Scriptlet

- public void _jspService(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException

```
  {
      response.setContentType("text/html");
    HttpSession session =  request.getSession();
      JspWriter out = response.getWriter(); out.println("<H2>foo</H2>");
    out.println(bar()); baz();
... }
```

Sample JSP Declaration

- <H1>Some Heading</H1>

```
<%! private String randomHeading()
      {
            return("<H2>" + Math.random() + "</H2>");
       }
 %>

<%= randomHeading() %>
```

Representative Resulting Servlet Code: Declaration

```
public class xxxx implements HttpJspPage
 {
    private String randomHeading()
   {
    return("<H2>" + Math.random() + "</H2>");
    }
    public void _jspService(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
    response.setContentType("text/html");
     HttpSession session = request.getSession();
    JspWriter out = response.getWriter();
     out.println("<H1>Some Heading</H1>");
     out.println(randomHeading());
     ... }
... }
```

Sample Jsp Code

```
<html>
<Head>
<title>   JSP PAGE     </title>
</Head>
<body>
<%
       for(int i=0;i<5;i++)
       {
        %>
         The value of i is
       <% out.println(i);
       %>
       <hr>
       <%
       }          %>
</body>
</html>
```

Components of a JSP page

- Combination of
  - JSP elements
    - ➔Instructions to the JSP container about what code to generate and how it should operate
    - ➔They have specific start and end tags that identify them to the JSP compiler
  - Fixed Template Data
    - ➔Everything else that is not recognized by the JSP container.
    - ➔Passed through unmodified

JSP Elements

- Directives
  - These provide global information to the page
  - Eg: import ,error handling page ,setting the script language
- Declaratives
  - These are for page wide variable and method declarations
- Scriptlets
  - Java code embedded in the page
- Expressions
  - Formats the expression as a string for inclusion in the output of the page

Directives

- Statement that gives the JSP engine information for the page that follows
- Placed at the beginning of a JSP page.
- General syntax
  - <%@ directive { attribute =“value” } %>
- Each directive may have a number of (optional) attributes
- 3 types of directives
  - page
  - include
  - taglib

page Directive Attributes

- All are optional .The mandatory ones have default values
- language="java"
  - Only supported language in the current JSP specification
- extends="package.class"
- import="package.* , package.class"
- session="true|false"
  - By default true and is available to the page
- errorPage="ErrPage"
  - Page that will handle unhandled exceptions .The ErrPage must have isErrorPage set to true
- isErrorPage="true|false"

include Directive

- Merges the contents of another file at translation time into the .jsp source input stream
- <%@include file="filename"%>

Comments in JSP

- 2 methods
  - Hidden Comments (visible only in JSP page)
  - Comments included in the generated HTML

- <%-- This is a hidden comment --%>

- <!-- This is included in the html -->

Expressions

- Accessing java variable value and merging that with the HTML in the page.
- Syntax
  - <%= exp %>
- Example
  - The current time is < %= new java.util.Date() %>

Declarations

- *Declarations* are used to define page-wide variables and methods.
- Basic Syntax

```
<%!
… some java code
%>
```

Example using Directives and Declarations

Exp.jsp

```
<%@ page import="java.util.*" %>

<html><body>
<h2>Welcome to the Weather Site</h2> <hr>

Today's date is  <%= new Date() %>

<%!
        String clr=“blue”;
        String getColor() {
                return clr;
        }
%>

<p>The color value is <b> <%= get Color() %> </b>

</body>
</html>
```

JSP Syntax - Scriptlets

- *Scriptlets* are used to execute Java code-blocks.
- Basic Syntax

```
<% … java code … %>
```

- Can place *any* valid Java code in the *scriptlet!*
- Also have access to predefined variables.

Example using Scriptlets

Scr.jsp

```
<html><body>

<%
        String[ ] cities = {“Houston”, “Boston”, “Atlanta”, “Chicago”};

        for (int i=0; i < cities.length; i++) {
          out.println(“<li>” + cities[i]);
        }
%>

</body>
</html>
```

Predefined Variables

- JSP has predefined variables
  - `request`
  - `response`
  - `out`
  - `Session`
  - `application`
  - `page`

JSP Predefined Variables (cont)

- **`request`**
  - contains HTTP information sent from client
  - useful for reading form data
    - `request.getParameter("form field");`
- **`response`**
  - used to send HTTP information back to client
  - useful for setting content-type, adding cookies
- **`out`**
  - output writer, a `JspWriter` object
  - useful for sending HTML text back to client
    - `out.println("The forecast is ...");`

JSP Predefined Variables (cont)

- **`Session`**

  the session object created for the client

  an object of javax.servlet.http.HttpSession

  session.getValue("key");

  session.putValue("key", value);

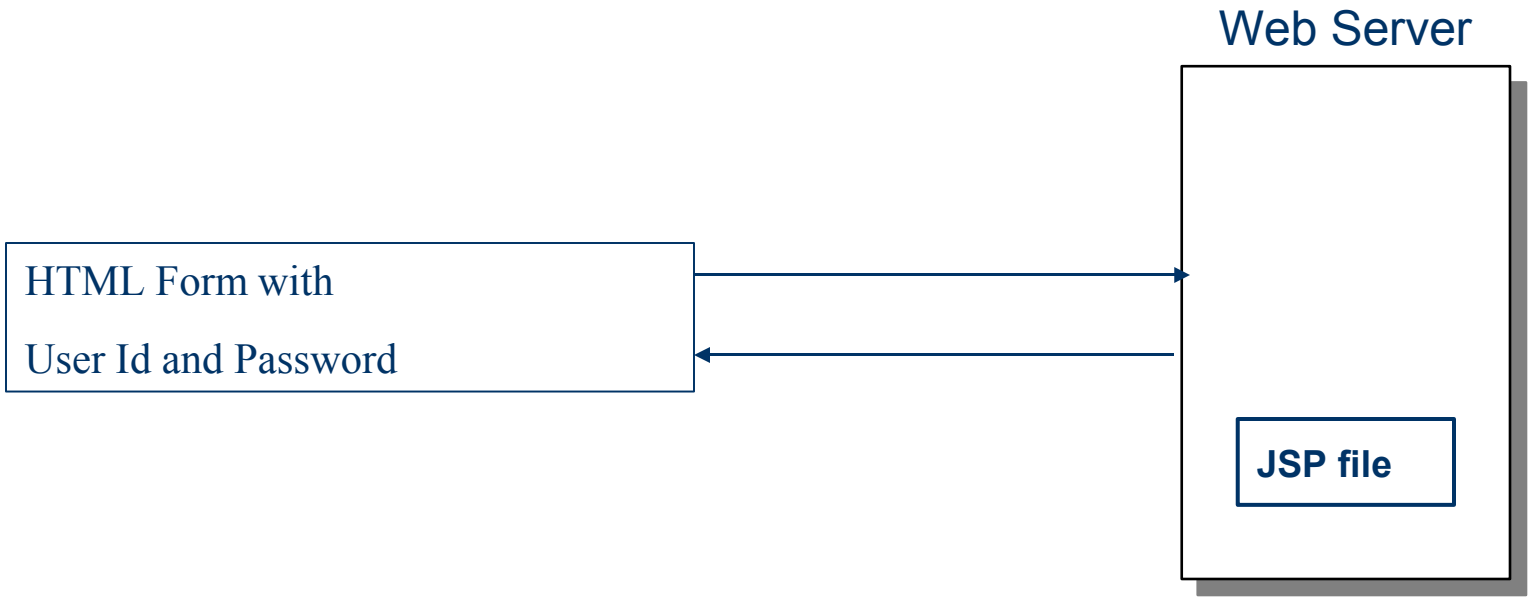
- **Application**

  the ServletContext.

  This is a data structure shared by all servlets and JSP pages in the Web application and is good for storing shared data.

Example Accessing Form Data

- Allow a user to enter a User id and password from an HTML form.
- Requested data will be catches by Jsp file.



Assignment

- Check user entered values are available in database or not , and send appropriate message to the user

The errorPage and isErrorPage Attributes

- The errorPage attribute specifies a JSP page that should process any exceptions (i.e., something of type Throwable) thrown but not caught in the current page.

  It is used as follows:

- **<%@ page errorPage="Relative URL" %>**

  The exception thrown will automatically be available to the designated error page by means of the exception variable.

- The isErrorPage attribute indicates whether or not the current page can act as the error page for another JSP page. Use of isErrorPage takes one of the following two forms:

- **<%@ page isErrorPage="true" %> <%@ page isErrorPage="false" %> <%-- Default --%>**

SpeedErrors.jsp

```
<HTML>
<HEAD> <TITLE>Error Computing Speed</TITLE> </HEAD>
<BODY>
<%@ page isErrorPage="true" %>
 Error Computing Speed
<P> ComputeSpeed.jsp reported the following error:
<I><%= exception %></I>.
 This problem occurred in the following place:
<PRE> <%@ page import="java.io.*" %>
 <% exception.printStackTrace(new PrintWriter(out)); %>
 </PRE> </BODY></HTML>
```

The isThreadSafe Attribute

- The isThreadSafe attribute controls whether the servlet that results from the JSP page will allow concurrent access (the default) or
- will guarantee that no servlet instance processes more than one request at a time (isThreadSafe="false"). Use of the isThreadSafe attribute takes one of the following two forms.
- **<%@ page isThreadSafe="true" %> <%-- Default --%>**
- **<%@ page isThreadSafe="false" %>**

Note :: Is there another alternative for this?

Request Dispatching

- Including Other Resources:
- <%@ include %>
  - Page Directive
  - Used to copy static text into the JSP source code before it is transformed into servlet source code.
  - Typically HTML code
  - Performed once at compile time
- <jsp:include> action
  - Causes the servlet engine to invoke another URL and merge its output with the original page.
  - Performed with every request

include Directive

- <%@ include file="filename" %>
- Filename must be only the URI specification (only path info ,not protocol or server info

Example

- Web1.jsp

```
<h1> The List of Jsp Elements are :</h1>

<%@ include file="elements.html" %>
Learn them all !!!!
<br>
All The Best !!!!!!
```

Example

- elements.html

```
<ol>
<li>Scriplets
<li>Expression
<li>Declerative
</ol>
```

Note: can I have only static data? (.html file)

Using include Directive to copy Java Source code

IncludeDirective.JSP

- <BODY>
- <P>Power of Include directive</P>
- <%@ include file="snippet.jsp" %>
- <%= accesscount++%>
- </BODY>
- </HTML>

Example

snippet.jsp

- <html>
- <BODY>
- <%!int accesscount =0; %>
- </BODY>
- </html>

jsp:include Action

- <jsp: include page="pagename" flush="true" />
- Only parsed by the JSP compiler
- The jsp page invoked by the include action has access to all the implicit objects available to the calling JSP

Form.html

- <BODY>
- <FORM action="IncludeAction.jsp">
- Enter your Name <INPUT type="text" name="nm" size="20">
- <INPUT type="submit" name="click" value="click">
- </FORM>
- </BODY>

IncludeAction.jsp

```
<html>
<BODY>
<P>Main Page</P>
<%!
String s;
%>
<%
s= request.getParameter("nm");
%>
<jsp:include page="first.jsp" flush="false">
<jsp:param name="nm" value="<%=s%>"/>
</jsp:include>
</BODY>
</HTML>
```

first.jsp

- <HTML>
- <BODY>
- Hello, <%= request.getParameter("nm")%>
- <P>Welcome at AmDocs</P>
- </BODY>
- </HTML>

Forwarding Requests

- The named page is loaded
- The current page is terminated

JSP and Beans

Why Use Beans?

- **No Java syntax**. By using beans, page authors can manipulate Java objects using only XML-compatible syntax: no parentheses, semicolons, or curly braces. This promotes a stronger separation between the content and the presentation and is especially useful in large development teams that have separate Web and Java developers.
- **Simpler object sharing**. When you use the JSP bean constructs, you can much more easily share objects among multiple pages or between requests than if you use the equivalent explicit Java code.
- **Convenient correspondence between request parameters and object properties.** The JSP bean constructs greatly simplify the process of reading request parameters, converting from strings, and putting the results inside objects.

What Are Beans?

- A bean class must have a zero-argument (default) constructor.
- A bean class should have no public instance variables (fields).
- Persistent values should be accessed through methods calledget Xxx and setXxx

Using Beans:

- JavaBeans components in JSP pages:
- **jsp:useBean**. In the simplest case, this element builds a new bean. It is normally used as follows:

  <jsp:useBean id="beanName" class="package.Class" />

- **jsp:getProperty**. This element reads and outputs the value of a bean property. Reading a property is a shorthand notation for calling a method of the form getXxx.

  <jsp:getProperty name="beanName" property="propertyName" />

- **jsp:setProperty**. This element modifies a bean property (i.e., calls a method of the form setXxx).

  <jsp:setProperty name="beanName" property="propertyName" value="propertyValue" />

Beans and Form processing

GetName.html

```
<HTML>
<BODY>
<FORM METHOD=POST ACTION="SaveName.jsp">
What's your name? <INPUT TYPE=TEXT NAME=username SIZE=20><BR>
What's your e-mail address? <INPUT TYPE=TEXT NAME=email SIZE=20><BR>
What's your age? <INPUT TYPE=TEXT NAME=age SIZE=4> <P><INPUT TYPE=SUBMIT>
</FORM>
</BODY>
</HTML>
```

UserData.java

```
public class UserData
 {
  String username;
String email;     int age;
   public void setUsername( String value )
{         username = value;     }
   public void setEmail( String value )
 {         email = value;     }
 public void setAge( int value )
  {         age = value;     }
 public String getUsername()
{ return username; }
 public String getEmail()
 { return email; }
public int getAge()
{ return age; }
}
```

SaveName.jsp"

- <HTML> <BODY>

  <jsp:useBean id="user" class="UserData" scope="session"/>
  <jsp:setProperty name="user" property="*"/>

  <A HREF="NextPage.jsp">Continue</A>

  </BODY> </HTML>

NextPage.jsp

```
<jsp:useBean id="user" class="UserData" scope="session"/>
 <HTML>
<BODY> You entered<BR>
    Name: <%= user.getUsername() %><BR>
    Email: <%= user.getEmail() %><BR>
    Age: <%= user.getAge() %><BR>
</BODY>
</HTML>
```