Java Server Pages

Unit II

Introduction

- JavaServer Pages (JSP) technology enables you to mix regular, static HTML with dynamically generated content from servlets.
- You simply write the regular HTML in the normal manner, using familiar Web-page-building tools. You then enclose the code for the dynamic parts in special tags, most of which start with ***<% and end with %>.***
- For Example:
- Thanks for ordering <I><%= request.getParameter("title") %></I>
- Separating the static HTML from the dynamic content provides a number of benefits over servlets alone, and the approach used in JavaServer Pages offers several advantages over competing technologies such as ASP, PHP, or ColdFusion.

- JSP is widely supported and thus doesn’t lock you into a particular operating system or Web server and that JSP gives you full access to servlet and Java technology for the dynamic part, rather than requiring you to use an unfamiliar and weaker special-purpose language.
- The process of making JavaServer Pages accessible on the Web is much simpler than that for servlets.
- Assuming you have a Web server that supports JSP, you give your file a .jsp extension and simply install it in any place you could put a normal Web page: no compiling, no packages, and no user CLASSPATH settings.

- Although what you write often looks more like a regular HTML file than a servlet, behind the scenes, the JSP page is automatically converted to a normal servlet, with the static HTML simply being printed to the output stream associated with the servlet's service method.
- If you make an error in the dynamic portion of your JSP page, the system may not be able to properly translate it into a servlet. If your page has such a fatal translation-time error, the server will present an HTML error page describing the problem to the client.

- Aside from the regular HTML, there are three main types of JSP constructs that you embed in a page:
  - *scripting elements,*
  - *directives, and*
  - *actions.*
- Scripting elements let you specify Java code that will become part of the resultant servlet,
- directives let you control the overall structure of the servlet,and
- actions let you specify existing components that should be used and otherwise control the behavior of the JSP engine.

Scripting Elements

- JSP scripting elements let you insert code into the servlet that will be generated from the JSP page. There are three forms:

1. ***Expressions*** *of the form <%= expression %>, which are evaluated* and inserted into the servlet's output
2. ***Scriptlets*** *of the form <% code %>, which are inserted into the* servlet's _jspService method (called by service)
3. ***Declarations*** *of the form <%! code %>, which are inserted into* the body of the servlet class, outside of any existing methods

Template Text

- In many cases, a large percentage of your JSP page just consists of static HTML, known as *template text.*
- *In almost all respects, this HTML looks just* like normal HTML, follows all the same syntax rules, and is simply “passed through” to the client by the servlet created to handle the page.
- There are two minor exceptions to the “template text is passed straight through” rule.
- First, if you want to have <% in the output, you need to put <\% in the template text.
- Second, if you want a comment to appear in the JSP page but not in the resultant document, use

`<%-- *JSP Comment* --%>

- HTML comments of the form

  <!-- *HTML Comment* -->

are passed through to the resultant HTML normally.

JSP Expressions

- A JSP expression is used to insert values directly into the output. It has the following form:

  <%= *Java Expression %>*
- The expression is evaluated, converted to a string, and inserted in the page.
- This evaluation is performed at run time (when the page is requested) and thus has full access to information about the request.
- For example, the following shows the date/time that the page was requested:

  Current time: <%= new java.util.Date() %>

Cont..

- ***Predefined Variables:*** To simplify these expressions, you can use a number of predefined variables.
- **request**, the HttpServletRequest
- **response**, the HttpServletResponse
- **session**, the HttpSession associated with the request
- **out**, the PrintWriter (a buffered version called JspWriter) used to send output to the client
- Here is an example:
- Your hostname: <%= request.getRemoteHost() %>

- ***Example:***

```
<HTML>
    <HEAD>
        <TITLE>JSP Expressions</TITLE>
    </HEAD>
    <BODY>
        <H2>JSP Expressions</H2>
        <UL>
            <LI>Current time: <%= new java.util.Date() %>
            <LI>Your hostname: <%= request.getRemoteHost()
    %>
            <LI>Your session ID: <%= session.getId() %>
            <LI>The <CODE>testParam</CODE> form parameter:
            <%= request.getParameter("testParam") %>
        </UL>
    </BODY>
</HTML>
```

JSP Scriptlets

- If you want to do something more complex than insert a simple expression, JSP scriptlets let you insert arbitrary code into the servlet's _jspService method (which is called by service).
- Scriptlets have the following form:
- <% *Java Code* %>
- Scriptlets have access to the same automatically defined variables as Expressions So, for example, if you want output to appear in the resultant page, you would use the out variable, as in the following example.

```
<%
        String queryData = request.getQueryString();
        out.println("Attached GET data: " + queryData);
%>
```

- Setting response headers and status codes, invoking side effects such as writing to the server log or updating a database, or executing code that contains loops, conditionals, or other complex constructs.
- <% response.setContentType("text/plain"); %>
- It is important to note that you can set response headers or status codes at various places within a JSP page, even though this capability appears to violate the rule that this type of response data needs to be specified before any document content is sent to the client.

```
<HTML>
  <HEAD>
   <TITLE>JSP: Scriptlets</TITLE>
  </HEAD>
   <%
      String bgColor = request.getParameter("COLOR");

      if (bgColor == CYAN)
         bgColor = "WHITE";

   %>
  <BODY BGCOLOR="<%= bgColor %>" >
   <H1>Example Scriptlet: Sets background color</H1>

  </BODY>
 </HTML>
```

JSP Declarations

- A JSP declaration lets you define methods or fields that get inserted into the main body of the servlet class.
- (*outside of the _jspService method that is* called by service to process the request).
- A declaration has the following form:
- <%! *Java Code %>*
- Since declarations do not generate any output, they are normally used in conjunction with JSP expressions or scriptlets.

```
<%! int data=50; %>
<%= "Value of the variable is:"+data %>
<%!
int cube(int n){
return n*n*n*;
}
%>
<%= "Cube of 3 is:"+cube(3) %>
```

Difference between JSP Scriptlet tag and Declaration tag

Jsp Scriptlet Tag	Jsp Declaration Tag
The jsp scriptlet tag can only declare variables not methods.	The jsp declaration tag can declare variables as well as methods.
The declaration of scriptlet tag is placed inside the _jspService() method.	The declaration of jsp declaration tag is placed outside the _jspService() method.

Predefined Variables

- To simplify code in JSP expressions and scriptlets, you are supplied with eight automatically defined variables, sometimes called *implicit objects.*
- *Since JSP* declarations result in code that appears outside of the _jspService method, these variables are not accessible in declarations.
- The available variables are request, response, out, session, application, config, pageContext, and page.

Object	Type
out	JspWriter
request	HttpServletRequest
response	HttpServletResponse
config	ServletConfig
application	ServletContext
session	HttpSession
pageContext	PageContext
page	Object
exception	Throwable

JSP Directives

- The **jsp directives** are messages that tells the web container how to translate a JSP page into the corresponding servlet.
- There are three types of directives:
  - page directive
  - include directive
  - taglib directive
- Syntax is:

<%@ directive attribute="value" %>

<%@ directive attribute1="value1"

attribute2="value2"

...

attribute*N="valueN" %*>

- The page directive lets you control the structure of the servlet by importing classes, customizing the servlet superclass, setting the content type, and the like. A page directive can be placed anywhere within the document.
- The second directive, include, lets you insert a file into the servlet class at the time the JSP file is translated into a servlet. An include directive should be placed in the document at the point at which you want the file to be inserted.
- JSP 1.1 introduces a third directive, taglib, which can be used to define custom markup tags.

JSP page directive

- The page directive defines attributes that apply to an entire JSP page.
- Syntax of JSP page directive
- <%@ page attribute="value" %>
- The page directive lets you define one or more of the following case-sensitive attributes:

- Attributes of JSP page directive
  - import
  - contentType
  - extends
  - info
  - buffer
  - language
  - isELIgnored
  - isThreadSafe
  - autoFlush
  - session
  - pageEncoding
  - errorPage
  - isErrorPage

- ***import***: The import attribute is used to import class, interface or all the members of a package. It is similar to import keyword in java class or interface.
- <%@ page **import**="java.util.Date" %>
- <%@ page import="package.class1,...,package.classN" %>
- If you don’t explicitly specify any classes to import, the servlet imports java.lang.*, javax.servlet.*, javax.servlet.jsp.*, javax.servlet.http.*, and possibly some number of server-specific entries.
- The import attribute is the only page attribute that is allowed to appear multiple times within the same document.
- Although page directives can appear anywhere within the document, it is traditional to place import statements either near the top of the document or just before the first place that the referenced package is used.

- ***contentType:*** The contentType attribute defines the MIME(Multipurpose Internet Mail Extension) type of the HTTP response. The default value is "text/html;charset=ISO-8859-1".
- The contentType attribute sets the Content-Type response header, indicating the MIME type of the document being sent to the client.
- Use of the contentType attribute takes one of the following two forms:
- <%@ page contentType="MIME-Type" %>
- <%@ page contentType="MIME-Type; charset=Character-Set" %>
- Unlike regular servlets, where the default MIME type is text/plain, the default for JSP pages is text/html

- ***extends***: The extends attribute defines the parent class that will be inherited by the generated servlet.It is rarely used.
- ***info:*** This attribute simply sets the information of the JSP page which is retrieved later by using getServletInfo() method of Servlet interface.
- <%@ page info="composed by abc" %>
- Today is: <%= **new** java.util.Date() %>
- The web container will create a method getServletInfo() in the resulting servlet.For example:
- **public** String getServletInfo() {
- **return** "composed by Sonoo Jaiswal";
- }

- ***buffer:*** The buffer attribute sets the buffer size in kilobytes to handle output generated by the JSP page. The default size of the buffer is 8Kb.
- <%@ page buffer="16kb" %>
- ***language:*** The language attribute specifies the scripting language used in the JSP page. The default value is "java".
- ***isELIgnored:*** We can ignore the Expression Language (EL) in jsp by the isELIgnored attribute. By default its value is false i.e. Expression Language is enabled by default.
- <%@ page isELIgnored="true" %>//Now EL will be ignored
- ***errorPage:***
- The errorPage attribute is used to define the error page, if exception occurs in the current page, it will be redirected to the error page

- <%@ page errorPage="myerrorpage.jsp" %>
- ***isErrorPage***: The isErrorPage attribute is used to declare that the current page is the error page.
- <%@ page isErrorPage="true" %>
- Sorry an exception occured!
- The exception is: <%= exception %>
- **The exception object can only be used in the error page.**

- ***The isThreadSafe Attribute*:** The isThreadSafe attribute controls whether or not the servlet that results from the JSP page will implement the SingleThreadModel interface.
- Use of the isThreadSafe attribute takes one of the following two forms:
- <%@ page isThreadSafe="true" %> <%!-- Default --%>
- <%@ page isThreadSafe="false" %>
- With normal servlets, simultaneous user requests result in multiple threads concurrently accessing the service method of the same servlet instance.

- ***The session Attribute*:** The session attribute controls whether or not the page participates in HTTP sessions.
- Use of this attribute takes one of the following two forms:
  - <%@ page session="true" %> <%-- Default --%>
  - <%@ page session="false" %>
- A value of true indicates that the predefined variable session (of type HttpSession) should be bound to the existing session if one exists; otherwise, a new session should be created and bound to session.
- A value of false means that no sessions will be used automatically and attempts to access the variable session will result in errors at the time the JSP page is translated into a servlet.

- ***The autoflush Attribute*:** controls whether the output buffer should be automatically flushed when it is full or whether an exception should be raised when the buffer overflows.
- Use of this attribute takes one of the following two forms:
- <%@ page autoflush="true" %> <%-- Default --%>
- <%@ page autoflush="false" %>

Jsp Include Directive

- The include directive is used to include the contents of any resource it may be jsp file, html file or text file.
- This directive tells the container to merge the content of other external files with the current JSP during the translation phase. You may code *include* directives anywhere in your JSP page.
- A good example of **include** directive is including a common header and footer with multiple pages of content.
- It is used for Code Reusability.
- Syntax: <%@ include file="resourceName" %>

Taglib Directives

- The JavaServer Pages API allows you to define custom JSP tags that look like HTML or XML tags and a tag library is a set of user-defined tags that implement custom behavior.
- The **taglib** directive declares that your JSP page uses a set of custom tags, identifies the location of the library, and provides a means for identifying the custom tags in your JSP page.
- <%@ taglib uri="uri" prefix="prefixOfTag" >
- Where the **uri** attribute value resolves to a location the container understands and the **prefix** attribute informs a container what bits of markup are custom actions.

JSP Action Tags

- JSP actions use constructs in XML syntax to control the behavior of the servlet engine. You can dynamically insert a file, reuse JavaBeans components, forward the user to another page, or generate HTML for the Java plugin.
- Each JSP action tag is used to perform some specific tasks.
- <jsp:action_name attribute="value" />
- Common Attributes:
  - There are two attributes that are common to all Action elements: the **id** attribute and the **scope** attribute.
  - **Id attribute:** The id attribute uniquely identifies the Action element, and allows the action to be referenced inside the JSP page
  - **Scope attribute:** The id attribute and the scope attribute are directly related, as the scope attribute determines the lifespan of the object associated with the id. The scope attribute has four possible values: (a) page, (b)request, (c)session, and (d) application.

JSP Action Tags	Description
jsp:forward	forwards the request and response to another resource.
jsp:include	includes another resource.
jsp:useBean	creates or locates bean object.
jsp:setProperty	sets the value of property in bean object.
jsp:getProperty	prints the value of property of the bean.
jsp:plugin	embeds another components such as applet.
jsp:param	sets the parameter value. It is used in forward and include mostly.
jsp:fallback	can be used to print the message if plugin is working. It is used in jsp:plugin.

jsp:forward Action

- The jsp:forward action tag is used to forward the request to another resource it may be jsp, html or another resource.
- Syntax:

  <jsp:forward page="relativeURL | <%= expression %>" />
- Syntax of jsp:forward action tag with parameter

  <jsp:forward page="relativeURL | <%= expression %>">

  <jsp:param name="parametername" value="parametervalue | <%=expression%>" />

</jsp:forward>
- df

The <jsp:include> Action

- The **jsp:include action tag** is used to include the content of another resource it may be jsp, html or servlet.
- The jsp include action tag includes the resource at request time so it is **better for dynamic pages** because there might be changes in future.
- The jsp:include tag can be used to include static as well as dynamic pages.
- Syntax:
- <jsp:include page="relativeURL | <%= expression %>" />
- <jsp:include page="relativeURL | <%= expression %>">

  <jsp:param name="parametername" value="parametervalue | <%=expression%>" />

  </jsp:include>

- Difference between jsp include directive and include action:

JSP include directive	JSP include action
includes resource at translation time.	includes resource at request time.
better for static pages.	better for dynamic pages.
includes the original content in the generated servlet.	calls the include method.

jsp:plugin

- The jsp:plugin action tag is used to embed applet in the jsp file. The jsp:plugin action tag downloads plugin at client side to execute an applet or bean.
- Syntax:
- **<jsp:plugin** type= "applet | bean" code= "nameOfClassFile"
  codebase= "directoryNameOfClassFile"
- **</jsp:plugin>**

JSTL (JSP Standard Tag Library)

- The JSP Standard Tag Library (JSTL) represents a set of tags to simplify the JSP development.
- The JavaServer Pages Standard Tag Library (JSTL) is a collection of useful JSP tags which encapsulates core functionality common to many JSP applications.
- JSTL has support for common, structural tasks such as iteration and conditionals, tags for manipulating XML documents, internationalization tags, and SQL tags. It also provides a framework for integrating existing custom tags with JSTL tags.
- The JSTL tags can be classified, according to their functions, into following JSTL tag library groups that can be used when creating a JSP page:

- Advantage of JSTL
  1. **Fast Developement** JSTL provides many tags that simplifies the JSP.
  2. **Code Reusability** We can use the JSTL tags in various pages.
  3. **No need to use scriptlet tag** It avoids the use of scriptlet tag.
- For creating JSTL application, you need to load jstl.jar file.
- There JSTL mainly provides 5 types of tags:

Tag Name	Description
Core tags	The JSTL core tag provide variable support, URL management, flow control etc. The url for the core tag is**http://java.sun.com/jsp/jstl/core** . The prefix of core tag is **c**.
Function tags	The functions tags provide support for string manipulation and string length. The url for the functions tags is **http://java.sun.com/jsp/jstl/functions** and prefix is **fn**.
Formatting tags	The Formatting tags provide support for message formatting, number and date formatting etc. The url for the Formatting tags is **http://java.sun.com/jsp/jstl/fmt** and prefix is **fmt**.
XML tags	The xml sql tags provide flow control, transformation etc. The url for the xml tags is**http://java.sun.com/jsp/jstl/xml** and prefix is **x**.
SQL tags	The JSTL sql tags provide SQL support. The url for the sql tags is **http://java.sun.com/jsp/jstl/sql**and prefix is **sql**.

Core Tags:

- The JSTL core tag provides variable support, URL management, flow control etc. The syntax used for including JSTL core library in your JSP is:
- **<**%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %**>**
- JSTL Core Tags List

Tags	Description
c:out	It display the result of an expression, similar to the way <%=...%> tag work.
c:import	It Retrives relative or an absolute URL and display the contents to either a String in 'var',a Reader in 'varReader' or the page.
c:set	It sets the result of an expression under evaluation in a 'scope' variable.
c:remove	It is used for removing the specified scoped variable from a particular scope.
c:catch	It is used for Catches any Throwable exceptions that occurs in the body.
c:if	It is conditional tag used for testing the condition and display the body content only if the expression evaluates is true.
c:choose, c:when, c:otherwise	It is the simple conditional tag that includes its body content if the evaluated condition is true.
c:forEach	It is the basic iteration tag. It repeats the nested body content for fixed number of times or over collection.

c:forTokens	It iterates over tokens which is separated by the supplied delimeters.
c:param	It adds a parameter in a containing 'import' tag's URL.
c:redirect	It redirects the browser to a new URL and supports the context-relative URLs.
c:url	It creates a URL with optional query parameters.

Expression Language

- The **Expression Language** (EL) simplifies the accessibility of data stored in the Java Bean component, and other objects like request, session, application etc.
- There are many implicit objects, operators and reserve words in EL.
- Syntax for Expression Language (EL)
- ${ expression }

Implicit Objects	Usage
pageScope	it maps the given attribute name with the value set in the page scope
requestScope	it maps the given attribute name with the value set in the request scope
sessionScope	it maps the given attribute name with the value set in the session scope
applicationScope	it maps the given attribute name with the value set in the application scope
param	it maps the request parameter to the single value
paramValues	it maps the request parameter to an array of values
header	it maps the request header name to the single value
headerValues	it maps the request header name to an array of values
cookie	it maps the given cookie name to the cookie value
initParam	it maps the initialization parameter
pageContext	it provides access to many objects request, session etc.