Java Server Pages

JSP Basics - Objectives

- JSP Basics
  - Syntax elements
  - JSP life cycle
  - JSP Page directives

JSP Basics – Syntax Elements

Tag Type	Brief Description	Tag Syntax
Directive	Specifies translation time instructions to JSP engine	<%@ directive %>
Declaration	Declares and defines methods and variables	<%! Java declarations %>
Scriptlet	Allows the developer to write free-form java code in a JSP page	<% some java code %>

JSP Basics – Syntax Elements

Tag Type	Brief Description	Tag Syntax
Expression	Used as a shortcut to print values in the output HTML of the JSP page	<%= an expression %>
Action	Provides request time instructions to the JSP engine	<jsp:actionName />
Comment	Used for documentation and for commenting parts of JSP code	<%-- any text --%>

Syntax Elements - Example

- <html>

```
<body>
<%@ page language="java" %>
<%! int count=0; %>
<% count++; %>
Welcome!! You are visitor number
<%= count %>
</body>
</html>
```


Directive

Declaration

Scriptlet

Expression

Syntax elements - Directives

- Provide general information about the JSP page to the JSP engine
- Three types
  - page
  - taglib
  - include
- Points to remember about the syntax of the directives:
  - The tag names, their attributes, and their values are all case sensitive
  - The value must be enclosed within a pair of single or double quotes
  - A pair of single quotes is equivalent to a pair of double quotes
  - There must be no space between the equal sign(=) and the value

Syntax elements - Directives

- A *page* directive informs the engine about the overall properties of a JSP page.
  - Ex: `<%@ page language="java" %>` Scripting language is java
- An *include* directive tells the JSP engine to include the contents of another file in the current page
  - Ex: `<%@ include file="copyright.html" %>`
- A *taglib* directive is used for associating a prefix with a tag library
  - Ex: `<%@ taglib prefix="test" uri="taglib.tld" %>`

Syntax elements - Declarations

- Declarations declare and define variables and methods that can be used in the JSP page
  - Ex:

```
<%! int count=0; %>
<%!
    String color[] = {“red”, “green”, “blue”};
    String getColor(int i)
    {
        return color[i%3];
    }
%>
```

Syntax elements - Scriptlets

- Scriptlets are java code fragments that are embedded in the JSP page.
- A scriptlet always start with <% and %>
  - Ex :

```
<%@ page language="java" %>
<%! int count=0; %>
<% out.print("<html><body>");
count++;
 out.print("Welcome ! You are visitor number"+count);
 out.print("</body></html>");
%>
```

Syntax elements - Expressions

- Expressions acts as place holders for java language expressions.
- A JSP expression always starts with <%= and ends with %>

```
<html><body>
   <%@ page language="java" %>
   <%! int count=0; %>
Welcome ! You are visitor number <%= ++count %>
</body></html>
```

Syntax elements -Actions

- Actions are commands given to the JSP engine.
- They direct the engine to perform certain tasks during the execution of a page.
- General syntax of a JSP action

  <jsp:actionName attribute-list />

Syntax elements –Actions-Conti..

- There are six standard JSP actions
  - include
  - forward
  - useBean
  - setProperty
  - getProperty
  - plugin

Syntax elements - Comments

- Comments do not affect the out of a JSP page in any way but are useful for documentation purposes.
- JSP comment always starts with <%-- and ends with --%>
- The syntax of a jsp comment is

  <%-- anything you want to be commented --%>

```
<html><body>
<%-- JSP Comment --%>
<% //java comment %>
<!-- html comment -->
</body></html>
```

JSP page life cycle

Phase name	Description
Page translation	The page is parsed and a java file containing the corresponding servlet is created
Page compilation	The java file is compiled
Load class	the compiled class is loaded
Create instance	An instance of the servlet is created
Call jspInit()	This method is called before any other method to allow initialization
Call_jspService()	This method is called for each request
Call_jspDestroy	This method is called when the servlet container decides to take the servlet out of service

Understanding JSP page directive attributes

- import
- session
- errorPage
- isErrorPage
- language
- extends
- buffer
- autoFlush
- isThread Safe
- info
- contentType
- pageEncoding

JSP Advanced - Objectives

- The translation process
- JSP implicit variables and JSP implicit objects
- JSP page scopes

Understanding the translation process

- Some directives are used by the JSP engine during the translation phase to generate java code
  - Ex: import attribute of *page* directive aids in generating import statements
  - Ex: info attribute aids in implementing the getServletInfo() method of the generated servlet class
- All JSP declarations become a part of the generated servlet class
  - variables ____ instance variables
  - methods ____ instance methods

Understanding the translation process

- All JSP scriptlets become a part of the generated jspService() method
- All JSP expressions become a part of the _jspService() method. They are wrapped inside out.print()
- All JSP actions are replaced by calls to vendor-specific classes.
- All JSP comments are ignored
- Any other text becomes part of the _jspService() method. It is wrapped inside out.write()

JSP implicit variables and implicit objects

Identifier Name	Class or Interface	Description
application	interface javax.servlet.ServletContext	Refers to the web application's environment
session	interface javax.servlet.http.HttpSession	Refers to the current request to the page
response	interface javax.servlet.http.HttpServletResponse	Used for sending a response to the client
out	class javax.servlet.jsp.JspWriter	Refers to the output stream for the page

JSP implicit variables and implicit objects

Identifier Name	Class or Interface	Description
page	class java.lang.Object	Refers to the page's servlet instance
pageContext	class javax.servlet.jsp.PageContext	Refers to the page's environment
config	interface javax.servlet.ServletConfig	Refers to the servlet's configuration
exception	class java.lang.Throwable	Used for error handling

Implicit variables - application

- The application variable is of type javax.servlet.ServletContext, and it refers to the environment of the web application to which the JSP page belongs.
- Thus, the following two scriptlets are equivalent:

```
<%
String path =
application.getRealPath("/WEB-INF/counter.db");
application.log("Using: "+path);
%>
<%
String path =
getServletContext().getRealPath("/WEB-INF/counter.db
");
getServletContext().log("Using: "+path);
%>
```

Implicit variables - session

- Session, as in an HTTP session, is a concept that logically groups multiple requests from the same client as part of one conversation.
- session, as used in a *page* directive, refers to the attribute named session.

<%@ page session="true" %>

(By default, the value of session attribute is true)

Its value, which is true or false, determines whether or not the JSP page participates in an HTTP session.

Implicit variables - session

- Session, as in implicit object, refers to the variable session of type javax.servlet.http.HttpSession
- Session, as a scope of an object, refers to the lifetime and availability of the object. A session-scoped object persists throughout the life of an HTTP session.

Implicit variables - session

- <html>
  <body>
    <%@ page session="false" %>
  session is not used
    Session ID = <%= session.getId() %>

  Error: undefined symbol session
  </body>
  </html>

Implicit variables – request and response

- The request and response implicit variables are of type

  javax.servlet.http.HttpServletRequest

  and

  javax.servlet.http.HttpServletResponse, respectively.
- They are passed in as parameters to the _jspService() method when the page's servlet is executed upon a client request.

Implicit variables – request and response

- ```
  <html><body>
  <%
  String remoteAddr = request.getRemoteAddr();
  response.setContentType("text/html;charset=ISO
  -8859-1");
  %>
  Hi! Your IP address is <%= remoteAddr %>
  </body></html>
  ```
  ```

Implicit variables – page

- The implicit variable page is of class java.lang.Object, and refers to the instance of the generated servlet.

  Object page = this; ← this refers to instance of this servlet

  <%= page.getServletInfo() %> ← Error

  <%= ((Servlet)page.getServletInfo() %> ← OK;typecast

  <%= this.getServletInfo() %> ← OK

Implicit variables – pageContext

- The pageContext variable is of type javax.servlet.jsp.PageContext
- It does three things:
  - Stores references to the implicit objects.
  - It acts as a one-stop place for managing all the other objects, both user-defined and implicit, used by the JSP page, and it provides the getter methods to retrieve them

Implicit variables – pageContext

- Provides convenience methods to get and set attributes in different scopes.(explained in JSP page scope)
- Provides convenience methods for transferring requests to other resources in the web application

Implicit variables – pageContext

- For example, to write a request to another resource from a servlet, we have to write the following two lines:

```
RequestDispatcher rd =
  request.getRequestDispatcher("other.jsp");
rd.forward(request,response);
```

  - In a JSP page, we can do that in just one line by using the pageContext variable:

```
pageContext.forward("other.jsp");
```

Implicit variables – pageContext

Method	Description
void include(String relativeURL)	Includes the output of another resource in the output of the current page. Same as ServletRequest.getRequestDispatcher().include();
void forward(String relativeURL)	Forwards the request to another resource. Same as ServletRequest.getRequestDispatcher().forward();

Implicit variables – out

- The implicit variable out is of type javax.servlet.jsp.Jspwriter
- We use it directly in scriptlets and indirectly in expressions to generate HTML code:

```
<% out.println("Hello 1"); %>
<%=  "Hello 2" %>
For both of the above lines, the generated servlet
  code will use the out variable to print the
  values:
Public void _jspService(….)
{
    // other code
    out.print("Hello 1");
    out.print("Hello 2");
}
```

Implicit variables – config

- The implicit variable config is of type javax.servlet.ServletConfig
- We can pass configuration parameters that are specific to a JSP page, which the page can retrieve using the implicit variable config.

```
<html><body>
Servlet Name = <%= config.getServletName() %> <br>
Parameter region = <%=
config.getInitParameter("region") %>
</body></html>
```

Implicit variables – exception

- The implicit variable is of type java.lang.Throwable
- The exception variable is available to the pages that acts as error handlers for other pages

```
<html><body>
<%@ page isErrorPage='true' %>
Msg: <%= exception.toString() %>
</body></html>
```

Understanding JSP page scopes

- The JSP technology uses the following four scopes
  - application
  - session
  - request
  - page

Understanding JSP page scopes

Scope name	Existence and Accessibility
Application	Limited to a single web application
Session	Limited to a single user session
Request	Limited to a single request
Page	Limited to a single page (translation unit) and a single request

Static inclusion & Dynamic Inclusion

- Static inclusion:
  - In static inclusion, the contents of another file are included with the current JSP file at translation time to produce a single servlet.
  - <%@ include file="relativeURL" %>
  - <jsp:directive.include file="relativeURL" %>
- Implications of static inclusion:
  - No processing can be done at translation time, which means the file attribute value cannot be an expression
  - <%@ include file="other,jsp?abc=pqr" %> ← invalid
  - The included page may or may not be able to compile independently.

Static inclusion & Dynamic Inclusion

- Dynamic inclusion:
  - In this , when the JSP page is requested, it sends a request to another object, and the output from that object is included in the requested JSP page.

    <jsp:include page="relativeURL" flush="true"/>
  - <jsp:forward page="relativeURL"/> ← delegates the request processing to the forwarded component. The forwarded component then send the reply to the client.

Java Beans - Objectives

- Why use Beans?
- What are Beans?
- How to use Beans?
- How to share Beans?

Why use Beans?

- Java Beans are independent software components that we can use to assemble other components and applications
- No Java syntax
- Simpler object sharing
- Convenient correspondence between request parameters and object properties

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 called getXxx and setXxx

Using Beans: Basic Tasks

- Three main constructs to build and manipulate Java Beans components in JSP pages
  - jsp:useBean – Builds a new bean

    <jsp:useBean id="beanName" class="package.class"/>
  - jsp:getProperty – This element reads and outputs the value of a bean property

    <jsp:getProperty name="beanName" property="propertyName" />
  - jsp:setProperty – This element modifies a bean property

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

Sharing Beans – page scope

- Using scope="page" – No sharing
  - Create the bean: Use jsp:useBean with scope="page" (or no scope at all, since page is the default)
  - Modify the bean: use jsp:setProperty with property="*".Then supply request parameters that match the bean property names.
  - Access the bean: use jsp:getProperty

Sharing Beans – request scope

- Using request based sharing
  - Create the bean: use jsp:useBean with scope=“request”
  - Modify the bean: use jsp:setProperty with property=“*”.Then, supply request parameters that match the bean property names.
  - Access the bean in the second page: use jsp:useBean with the same id as on the first page, again with scope=“request”. Then, use jsp:getProperty

Sharing Beans – session scope

- Using session based sharing
  - Create the bean: use jsp:useBean with scope=“session”
  - Modify the bean: use jsp:setProperty with property=“*”.Then, supply request parameters that match the bean property names.
  - Access the bean in the initial request: use jsp:getProperty in the request in which jsp:setProperty is invoked
  - Access the bean later: use jsp:getProperty in a request that does not include request parameters and thus invoke jsp:setProperty.

Sharing Beans – application scope

- Using ServletContext based sharing
  - Create the bean: use jsp:useBean with scope="application"
  - Modify the bean: use jsp:setProperty with property="*".Then, supply request parameters that match the bean property names.
  - Access the bean in the initial request: use jsp:getProperty in the request in which jsp:setProperty is invoked.
  - Access the bean later: use jsp:getProperty in a request that does not include request parameters and thus does not invoke jsp:setProperty.

Bibliography

- SCWCD Exam study kit – Hanumant Deshmukh and Jignesh Malavia
- Core Servlets and JavaServer Pages, volume1:core Technologies, 2nd edition – Marty Hall and Larry Brown
- ServletBasics_speakernoted from Sun Microsystems
- ServletAdvanced_speakernoted from Sun Microsystems