JSP BASICS

JSP files are HTML files with special tags that contain Java source code that provide dynamic content.

LIFE Cycle of JSP:

- https://beginnersbook.com/2013/05/jsp-tutorial-life-cycle/

The steps in the life cycle of jsp page are:

- Translation
- Compilation
- Loading
- Instantiation
- Initialization
- Request Processing
- Destruction

JSP Life Cycle Methods

jspInit() - The web container calls the jspInit() to initialize the servlet instance generated. It is invoked before servicing the client request and invoke only once for a servlet instance.

jspservice() - The container calls the jspservice() for each user request, passing it the request and the response objects.

jspDestroy() - The container calls this when it decides take the instance out of service. It is the last method called in the servlet instance.

Difference between JSP and Servlet:

- https://javarevisited.blogspot.com/2017/05/difference-between-servlet-and-jsp.html

What are JSP implicit objects?

Implicit objects in JSP are the objects that are created by the web container automatically and the container makes them available to the JSP to access it. Implicit objects are available only inside the jspService() method hence cannot be accessed anywhere outside.

- https://www.tutorialspoint.com/jsp/jsp_implicit_objects.htm

Handling Request and Response Object

JSP provides both request and response as implicit objects . All the HttpServletRequest and HttpServletResponse methods can be called on these objects

Example :
request.getParameter(“name”) ;
response.sendRedirect(“home.jsp”);

Form Processing JSP

Rendering Forms:

Forms can be rendered as normal HTML forms inside JSP pages.

Forms can also be created dynamically by printing using the out.print() method if required.

Example: <%

```
 out.print(“<form>”);
out.print(“<input type='text' name='userName'/>”);
out.print (“</form>”);
%>
```

How to handle form using JSP?

- https://www.tutorialspoint.com/jsp/jsp_form_processing.htm

Handling Forms Actions:

Form actions can be handled by the same JSP or another servlet/JSP.

Example: When user clicks the login button in Login.jsp it can be handled by Login.jsp or HomePage.jsp or LoginValidator.java (Servlet)

If the same JSP is handling the form action then the form action attribute need not be specified.

Example: <form name=”LoginForm” method=”get”>

// No action attribute specified.

Session Handling in JSP:

The session management techniques used in servlets are also applicable to handle session in JSP also.

The following are the techniques used to manage session in JSP

URL rewriting

Hidden Field

Cookie

Session Object

Handling Session with implicit Session Object

1. In JSP the HttpSession object named session is automatically created by the container and passed to the service method for use.
2. By setting attributes to the implicit session object , session can be managed across pages in JSP.

session.setAttribute(attributeName,value) :- Sets attribute to session

session.getAttribute(attributeName) :-Reads the attribute value

session.removeAttribute(attributeName) : Removes attribute from the session.

Types of JSP elements:

JSP elements:

Directives
Actions
Scripting
Comments

Scripting Elements:

Scripting element are used to embed java code in JSP files. There are three types of scripting elements

- Scriptlet
- Declaration
- Expression
- https://javabeginnerstutorial.com/jsp-tutorial/jsp-scripting-elements/

Scriptlet Element:

- Used to embed java code in JSP pages.
- Contents of JSP scriptlet goes into jspService() method during the translation phase.

- Code should comply with syntactical and semantic construct of java.
- Embedded between <% and %> delimiters.

Creating a Scriplet

Scriplets are embedded between <% and %> delimiters

Syntax: <% Java code goes in here %>

Example: To print a variable value,

```
<%
String username = "visualbuilder" ;
 out.println ( username ) ;
%>
```

Declaration Element

- Declarations are used to declare, define methods & instance variables.
- Declaration tag does not produce any output that is sent to client.
- The methods and classes declared will be translated as class level variables and methods during translation.

How to declare?

Methods or variables are declared using <%! and %> delimiters.

Syntax:<%! Variable=0; %>

Example: This declares a variable count as int and set value 10.

```
<%! int count=10; %>
```

Expression Element:

- Used to write dynamic content back to the client browser.
- Used in place of out.print() method.
- Only expressions are supported inside the tag. Declarations of methods and variables is not possible inside this tag.
- During translation the return type of expression goes as argument into out.print()

method. Expression should not be ended with a semicolon (;) since semicolon are automatically added during translation.

How to use Expressions?

Expressions are Embedded in <%= and %> delimiters

Syntax: <%= expresion 1 %>

Example: To print the date dynamically for each client request.

```
<HTML>
<BODY>Hello! The time is now <%=new java.util.Date() %>
</BODY>
</HTML>
```

The date expression will be evaluated and the current date will be printed in the HTML rendered.

Comments:

There are two type of comments supported by JSP

- HTML comment

<!-- This is a comment -–!>

- JSP comment

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

- HTML comments are passed during the translation phase and hence can be viewed in the page source in the browser.
- JSP comments are converted to normal java comments during the translation process and will not appear in the output page source.

What is JSP directive

A directive provides meta data information about the JSP file to the web container. Web container uses this during the translation/compilation phase of the JSP life cycle.

- https://www.guru99.com/jsp-actions.html

The JSP specification defines three directives,

- Page:

  Provides information about page, such as scripting language that is used, content type, or buffer size etc.

- Include:

  Used to include the content of external files.

- Taglib:

  Used to import custom tags defined in tag libraries. Custom tags are typically developed by developers.

<u>JSP Directive Syntax:</u>

<%@ directive attribute="value" %>
where
directive – The type of directive (page,taglib or include)
attribute – Represents the behavior to be set for the directive to act upon.

<u>Page Directive:</u>

- The page directive is used to provide the metadata about the JSP page to the container. Page directives may be coded anywhere in JSP page.
- By standards, page directives are coded at the top of the JSP page.
- A page can have any number of page directives.
- Any attribute except the import attribute can be used only once in a JSP page.
- Single can contain more than one attribute specified.

Syntax:
<%@page attribute1="value" attribute2="value" %>

<u>Attributes for Page directive:</u>
buffer

autoFlush

contentType

errorPage

isErrorPage

extend

import

info

isThreadSafe

language

session

isELIgnored

isScriptingEnabled

- isErrorPage- Works in tandem with the page errorPage directive and specifies that this JSP is an error page. Syntax: <% @page isErrorPage="true|false"%>

<u>Include Directive:</u>

- This directive inserts a HTML file or a JSP file into another JSP file at translation time.
- The include process is static, it means that the text of the included file is added to the JSP file. (Similar to copy pasting the contents).
- The included file can be a JSP file, HTML file, or text file.
- If the included page is a JSP page it will be translated along with the main JSP page.

Note: Be careful that the included file should not contain <html> ,</html> ,<body> , or </body> tags. Because the entire content of the included file is added to the main JSP file, these tags would conflict with the same tags in the main JSP file, causing an error.

<u>Syntax:</u>

<%@include attribute =“value” %>

Include can have only file attribute

<%@include file="value" %>

where file determines the relative path of the file to be included.

Example:

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

Includes a file named header.

JSP BeansAndAction

What is a java bean?

A serverside java bean is a class used to store the details of realworld entities.

A Java Bean is a java class that should follow following conventions:

- It should have a no-arg constructor.
- It should be Serializable.
- It should provide methods to set and get the values of the properties, known as getter and setter methods.

Java Bean Design Convention:

A JavaBean component property can be read/write, read-only, or write-only.

• The bean property needs to be accessible using public methods.

For each readable property, the bean must have a method as illustrated below to retrieve the property value

```
Syntax:  Public Datatype get<PropertyName> {
 return value; // Returns the property value
}
```

For each writable property, the bean must have a method as illustrated below,

```
Syntax: Public  set<PropertyName>(DatatypenewValue) {
property= newValue; // sets the new value into the property
}
```

Need for Beans in JSP

- Beans are used to JSP for collectively storing some information.

- Beans makes transfer of data between JSP’s easier.
- For example

  if you are handling with a registration form all the registration details can be loaded into a RegistrationBean and can be transported across other components as a single object.

How to set value to a Bean ?

- Values can be set to the bean using the setter method.

userBean.setName(request.getParameter(“name”));

- Reads the parameter name from the request and sets it to the property name in userBean.

How to read values from a bean ?

- Values can be retrieved from a bean using the getter method .

String userName=userBean.getName();

Reads the property value name from the bean and assigns it to a variable

What is JSP action tag?

JSP action tags are set of predefined tags provided by the JSP container to perform some common tasks thus reducing the java code in JSP.

The action tags are used to control the flow between pages and to use Java Bean. The Jsp action tags are given below.

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.

- https://www.javatpoint.com/jsp-action-tags-forward-action

Syntax:

- <jsp:include page=”PageName” />
- <jsp:forward page=”URL” />
- <jsp:include page=”URL” />

  <jsp:param name=”paramname” value=”parameterValue”/>

  </jsp:include>
- <jsp:useBean id=”name” class=”package.class”

  scope=”request/session/page/application”/>
- <jsp:setPropertyname=”myname”property=”someProperty” value=”somevalue” />
- <jsp:getProperty name=”myname” property=”someProperty” />

Attributes for UseBean Action tag:

- id

  Gives a name to the variable that will reference the bean. A previous bean object is used instead of instantiating a new one if one can be found with the same id and scope.
- Class

  Instantiates a Bean from a class, using the new keyword and the class constructor. The class must not be abstract and must have a public, no-argument constructor. The package and class name are case sensitive.
- Scope

scope="page|request|session|application" Defines a scope in which the bean exists .The default value is page.

- Type

  If the Bean already exists in the scope, gives the Bean a data type other than the class from which it was instantiated. If you use type without class or beanName, no Bean is instantiated. The package and class name are case sensitive

- beanName

  Gives the name of the bean, as you would supply it to the instantiate method of Beans.

How jsp:useBean works?

Example : <jsp:useBean id="user" class="com.catp.beans.UserBean" scope="request " />

How it works?

Sequence of steps,

1. Attempts to locate a Bean with the name "UserBean" in the request scope.

2. If it finds the Bean, stores a reference in the variable user.

3. If it does not find the Bean, instantiates a bean using the class UserBean, and stores the reference to the variable user.

<u>Bean Object Scopes</u>:

- page :is available only within the JSP page and is destroyed when the page has finished generating its output for the request.
- request : valid for the current request and is destroyed when the response is sent
- session : valid for a user session and is destroyed when the session is destroyed
- application : valid throughout the application and is destroyed when the web application is destroyed/uninstalled.

JSP CUSTOM TAGS

What is a custom tag?

A custom tag is a user-defined JSP language element. When a JSP page containing a custom tag is translated into a servlet, the tag is converted to operations on an object called a tag handler. The Web container then invokes those operations when the JSP page's servlet is executed.

https://www.tutorialspoint.com/jsp/jsp_custom_tags.htm

Types of Custom Tags:

Simple Tags: A simple tag contains no body and no attributes.

Example: <tt:CustomDate/>

Tags With Attributes: A custom tag can have attributes. Attributes are listed in the start tag and have the syntax: attribute name ="value". Attribute are like configuration details for the custom tag.

```
<tt:CustomDate attribute=”value”/>
```

Tags with Bodies: A custom tag can contain custom, core tags, scripting elements and HTML text content between the start and end tag.

```
<tt:mytag/>
<h1>This is body inside the tag </h1>
</tt:mytag/>
```

Steps to create Custom Tags:

Step 1: Create a tag handler class with the custom logic.

Step 2: Create a tag library descriptor (TLD) and configure the custom handler. It should be placed in the web-inf directory.

Step 3: Import the custom tag library in JSP using taglib directive.

Step 4: Start using the custom tag in a JSP page.

Tag Handler Class:

- Tag Handler is a java class that holds the logic of the custom tag. This is triggered by the web container whenever it encounters the custom tag in a JSP file.
- We will be using the SimpleTagSupport class to create the custom tags.
- The SimpleTagSupport class is the base class intended to be used for developing tag handlers.
- The SimpleTagSupport class implements the SimpleTag interface and defines API's which can be overridden by the custom tag handlers.

Methods in SimpleTagSupport

- doTag()
- getParent()
- setParent(JspTag tag)
- setJspContext(JspCont ext context)
- getJspContext

JSP EXPRESSION LANGUAGE

What is an Expression Language(EL)?

Expression Language (EL) is a simple language for accessing data stored in java beans.

This was introduced with the JSP 2.0 specification.

It can also be used to access the values from implicit objects like page context , header , cookie etc.

Expression languages are always specified within curly braces and prefixed with a dollar sign.

Example: ${person.name} Period Operator(.) [ ] operator can be used instead of the period operator. ${person.["name"]}

- https://www.tutorialspoint.com/jsp/jsp_expression_language.htm

Implicit Objects in EL:

EL operators: Arithmetic, Logical, Relational

- https://docs.oracle.com/cd/E19316-01/819-3669/bnaij/index.html

- https://www.journaldev.com/2064/jsp-expression-language-el-example-tutorial
- If your application requires EL to be ignored it can be done by setting the isELIgnoredAttribute to true in the page directive.

```
<%@ page isELIgnored ="true"%>
```

JSTL

JSTL: Java Server Pages Standard Tag Library

- The Java Server Pages Standard Tag Library (JSTL) is a collection of useful JSP tags which encapsulates some useful functionality widely used in many JSP applications.
- JSTL has some common functionalities implemented such as iteration, conditionals statements, tags for manipulating XML documents, internationalization tags, and SQL tags.

Types of JSTL Tags:

Based on the functionality there are five categories of JSTL tags

- Core Tags
- Formatting tags
- SQL tags
- XML tags
- JSTL Functions.

The absolute URIs for the JSTL library are as follows:

- Core: http://java.sun.com/jsp/jstl/core
- XML: http://java.sun.com/jsp/jstl/xml
- Internationalization: http://java.sun.com/jsp/jstl/fmt
- SQL: http://java.sun.com/jsp/jstl/sql

- Functions: http://java.sun.com/jsp/jstl/functions

Core Tags:

- Core tags specify several actions such as displaying content based on a condition, manipulating collections and managing URL’s such as redirecting to a different page.
- Used to do the more common actions in easier and effective way whereby minimizing the use of scriptlets.
- Uses the prefix c.

<c:out>

- The tag displays the result of an expression, similar to the JSP expression tag

<c:if>

- The tag evaluates an expression and displays its body content only if the expression evaluates to true.

<c:forEach>

- The tag is used for looping. It lets us iterate through a collection of objects

<c:forTokens>

- The tag is used for tokenizing strings using a specific delimiter.

<c:set>

- The tag is used to sets the property of a bean or map similar to the action.
- It can also set a parameter with a value.
- Value can be set either in the body or using the “value” attribute.

<c:remove>

- The tag is used to remove an attribute set in a specified scope.
- <c:import>
- Used to include resources to a page can be a JSP page, servlet , XML file etc

- <c:param>
- Used to pass parameters to an imported resource
- <c:redirect>
- Used to redirect to a URL based on some conditions
- <c:catch>
- The tag catches any exceptions that occurs in its body

- https://www.tutorialspoint.com/jsp/jsp_standard_tag_library.htm