SESSION-12

JSP -IMPLICIT OBJECTS

Implicit objects are those which will be available to each and every JSP page by default. In JSP we have the following implicit objects.

Implicit object	Creator or Instantiated by
1. out	1. JSPWriter extends PrintWriter
2. request	2. HttpServletRequest
3. response	3. HttpServletResponse
4. application	4. ServletContext
5. config	5. ServletConfig
6. pageContext	6. PageContext
7. page	7. Object (core java example is this)
8. session	8. HttpSession
9. exception	9. Throwable

DIRECTIVES:

Directives are basically used to configure the code that is generated by container in a servlet. As a part of JSP we have three types of directives; they are page directives, include directives and taglib directives.

1. Page directives:

Page directives are basically used for supplying compile time information to the container for generating a servlet. The page directive will take the data in the form of (key, value) pair.

```
<%@
      page attribute_name1=attribute_value1,
            attribute_name2=attribute_value2,
            ……..,
            Attribute_nameN=attribute_valueN
%>
```

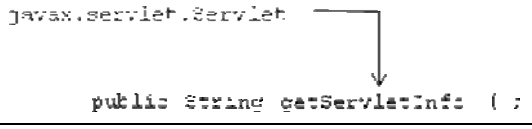
- Whenever we use page directive as a part of JSP program that statement must be the first statement.
- The scope of page directive is applicable to current JSP page only.

The following table gives the page directive attribute name and page directive attribute value:

Attribute name	Attribute value
1. import	1. This attribute is used for importing either pre-defined or user-defined packages. The default value is java.lang.* For example: `<%@ page import="java.sql.*, java.io.*" %>`
2. contentType	2. This attribute is used for setting the MIME type (plain/text, img/jpeg, img/gif, audio/wave, etc.). The default value is text/html. For example: `<%@ page contentType="img/jpeg" %>`
3. language	3.6 This attribute represents by default java i.e., in order to represent any business logic a JSP program is making use of java language. The attribute language can support any of the other programming languages for developing a business logic at server side. For example: `<%@ page language="java" %>`
4.6 isThreadSafe	4. This attribute represents by default true which represents one server side resource can be accessed by many number of clients (each client is treated as one thread). At a time if we make the server side resource to be accessed by a single client then the value of isThreadSafe is false. For example: `<%@ page isThreadSafe="false" %>`

5. isErrorPage
6. errorPage
	5.6 When we write 'n' number of JSP pages, there is a possibility of occurring exceptions in each and every JSP page. It is not recommended for the JSP programmer to write try and catch blocks in each and every JSP page. It is always recommended to handle all the exceptions in a single JSP page.
	isErrorPage is an attribute whose default value is true which indicates exceptions to be processed in the same JSP page which is not recommended. If isErrorPage is false then exceptions are not processed as a part of current JSP page and the exceptions are processed in some other JSP page which will be specified through an attribute called errorPage.
	For example: <%@ page isErrorPage="false" errorPage="err.jsp" %>
	err.jsp:
	`<%= exception %>`
	[or]
	`<%= exception.getMessage () %>`

7. autoflush 8. buffer	7. Whenever the server side program want to send large amount of data to a client, it is recommended to make autoflush value as false and we must specify the size of the buffer in terms of kb. The default value of autoflush is true which represents the server side program gives the response back to the client each

	and every time. Since, the buffer size is zero. For example: <%@ page autoflush=”false” buffer=”12kb” %> <%@ page autoflush=”true” %> [by default]
9. session	9. When we want to make ‘n’ number of independent requests as consecutive requests one must use the concept of session. In order to maintain the session we must give the value of session attribute has true in each and every JSP page (recommended). The default value of session is true which represents the session is applicable to current JSP page. For example: <%@ page session=”true” %> €session will be created or old session will be continued.
10. info	10. Using this attribute it is recommended for the JSP programmer to specify functionality about a JSP page, on what date it is created and author. In order to get information (info) of a JSP page we must use the following method: 

Write a JSP page which illustrates the concept of isErrorPage and errorPage?

Answer:

web.xml:

```
<web-app>
</web-app>
```

Exception.jsp:

```
<%@ page isErrorPage="false" errorPage="ErrorPage.jsp" %>
<html>
	<body>
		<%= 30/0 %>
	</body>
</html>
```

ErrorPage.jsp:

```
<%@ page isErrorPage="true" %>
<html>
	<body>
		Exception is <%= exception %> generated...<br>
		Exception message is <%= exception.getMessage () %><br>
		<%@ include file="copyright.html" %>
	</body>
</html>
```

Copyright.html:

```
<html>
	<br><br><br><em>All copy rights are reserved for kvr.com<em>
</html>
```

Develop JSP pages which will participate in session?

Answer:

web.xml:

```
<web-app>
</web-app>
```

First.jsp:

```
<%@ page session="true" %>
<html>
	<body>
		<form name="first" action="Second.jsp">
			Enter your name : <input type="text" name="first_name"><br>
			<input type="submit" value="Send">
		</form>
	</body>
</html>
```

Second.jsp:

```
<%@ page session="true" %>
<html>
	<body>
		<% String str=request.getParameter ("first_name"); %>
		Your name is <h4><%= str %></h4>
		<% session.setAttribute ("name",str); %>
		<form action="Third.jsp"><br>
			Send the request to next page<br>
			<input type="submit" value="Send">
		</form>
	</body>
</html>
```

Third.jsp:

```
<%@ page session="true" %>
<html>
	<body>
		Value in third page from session objects is
		<h4><%= session.getAttribute ("name") %></h4>
	</body>
</html>
```

2. Include directives:

Include is the directive to include the server side resource. The server side resource can be either an html file or JSP or a servlet.

If we include html file, it will be executed by browser when the response is rendering to the client. When we include a JSP or a servlet, it will be executed by container.

Syntax:

```
<% include file="file name to be included" %>
```

For example:

```
<% include file="copyright.html" %>
```

3. Taglib directives (Custom Tags):

The tags which are provided in JSP may not be solving the total problems of JSP programmer. In some situations, there is a possibility of repeating the same code many times by various JSP programmers and it leads to poor performance of a JSP application. In order to avoid the repetition of the code, it is highly desirable to develop the code only one time in the form of tags and use these tags by other JSP programmers. These tags are known as custom tags.

Advantages of custom tags:

1. Repetition of code (application logic) is reduced. Hence, we can achieve high performance and less storage cost.
2. We can achieve the slogan of WORA.
3. Custom tags provide simplicity for the JSP programmer in developing the application logic.

Steps for developing custom tags:

1. Decide which tag to be used along with prefix or short name, tag name and attribute names if required.
2. While we are choosing prefix it should not belongs to JSP, javax, javaw and java.
3. After developing a custom tag one must specify the details about tag in a predefined file called tld (Tag Library Descriptor) file.
4. tld file contains declarative details about custom tags.
5. After developing tld file keep it into either WEB-INF folder directly or keep it into a separate folder called tlds folder and it in turns present into WEB-INF.
6. Whenever we make a request to a JSP page where we are using custom tag will give location of tld file.
7. The tld file gives information about tag handler class (JavaBeans class) in which we develop the arithmetic logic or business logic for the custom tag.

Syntax for specifying the location of tld files:

```
<% taglib    uri="location of tld file"
             prefix="prefix or short name of custom tag" %>
```

For example:

```
<% taglib uri="/WEB-INF/tlds/x.tld" prefix="database" %>
```

Here, taglib is a directive used for given information regarding tld file and prefix or short name of custom tag.

Entries in tld file:

Every tld file gives declarative details about custom tags. The following structure gives information regarding prefix name, tag name, tag handler class name, attribute names, etc.

x.tld:

```
<taglib>
      <tlibversion>1.0</tlibversion>
      <jspversion>1.1</jspversion>
      <shortname>database</shortname>
```

```
        <tag>
                <name>display</name>
                <tagclass>MyPack.x</tagclass>
                <bodycontent>empty</bodycontent>
                <attribute>
                        <name>username</name>
                        <required>true</required>
                        <rtexprvalue>false</rtexprvalue>
                </attribute>
        </tag>
</taglib>
```

If a tag contains any attributes we must used <attribute/> in tld file. <attribute/> tag contains the following entries:

Syntax:

```
<attribute>
        <name>…….</name> € represents name of the attribute
        <required>…….</required> € represents either true or false
        <rtexprvalue>…….</rtexprvalue> € represents required text expression value
</attribute>
```

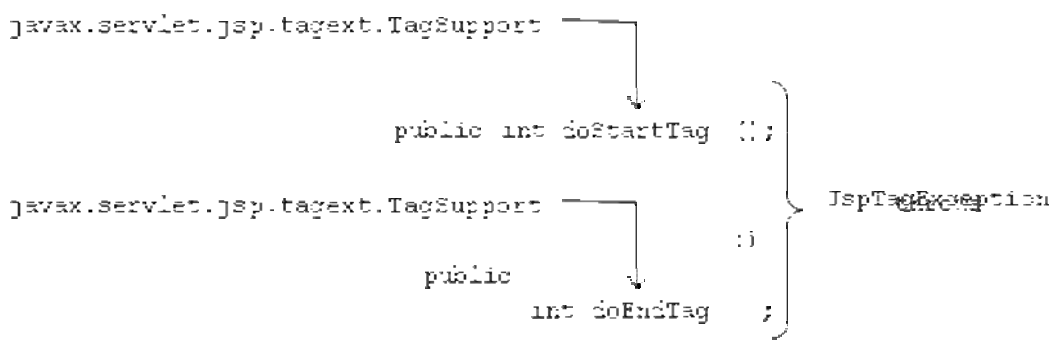
When we use true, the required attribute should name. This attribute should be used compulsorily. When it is false, it is optional to use.

If we pass the data at runtime to the rtexprvalue attribute, this attribute must be true otherwise it is false.

Steps for developing tag handler class:

A tag handler class is basically a JavaBeans class, it should contain set of set methods and set of get methods.

1. All the attributes of custom tag must be used as properties or data members in tag handler class in the same order in which they appear.
2. Every tag handler class must extend a predefined class called javax.servlet.jsp.tagext. TagSupport
3. TagSupport class contains the following two life cycle methods (which are automatically called by the JSP container) and they must be overridden in tag handler class.



The legal values which are returned by doStartTag are:

```
public static final int SKIP_BODY;
public static final int EVAL_BODY_INCLUDE;
```

Present in TagSupport class

doStartTag () method will be called by JSP container when starting of custom tag taken place **<**x:hello/>. SKIP_BODY is the constant which will be returned by doStartTag () method when we don't want to execute body of the custom tag. EVAL_BODY_INCLUDE is the constant which will be returned by doStartTag () method when we want to evaluate body of the custom tag.

The legal values which are returned by doEndTag () are:

```
public static final int EVAL_PAGE;
public static final int SKIP_PAGE;
```

Present in TagSupport class

doEndTag () method will be called by JSP container when the closing tag of custom tag taken place <x:hello**/>**. EVAL_PAGE is the constant to be returned by doEndTag () method when we want to execute rest of the JSP page. SKIP_PAGE is the constant to be returned by doEndTag () method when we don't want to execute rest of the JSP page.

URI → Location of file which is not running

URL → Location of file which is running

HelloTag.jsp:

```
<%@
	taglib uri="/WEB-INF/tlds/hello.tld" prefix="test"
%>
<html>
	<h3>This is example on custom tag</h3>
	<h4><test:hello name="Hyderabad"/></h4>
</html>
```

HelloTag.java (Tag Handler Class):

```
package t1;
import java.io.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
public class HelloTag extends TagSupport
{
	private String name;
	public void setName (String name)
	{
		this.name=name;
	}
	public int doEndTag () throws JspException
	{
		try
		{
			pageContext.getOut ().print ("Hello..! "+name);
		}
```

```
                catch (Exception e)
                {
                       throw new JspException (e.getMessage ());
                }
                return EVAL_PAGE;
        }
  }
```

Hello.tld:

```
  <taglib>
     <shortname>test</shortname>
     <tag>
        <name>hello</name>
        <tagclass>t1.HelloTag</tagclass>
        <bodycontent>empty</bodycontent>
        <attribute>
           <name>name</name>
         <required>true</required>
         <rtexprvalue>false</rtexprvalue>
        </attribute>
 </tag>
</tagli
     b>
```

NOTE:

pageContext is an object of javax.servlet.jsp.PageContext interface and it will be created automatically by JSP container. pageContext object is pre-declared in TagSupport class and this object will be available to each and every sub-class of TagSupport class.