Name of the School: School of Computer Science and Engineering

Course Code: R1UC602C Course Name: Web Technology

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Subject Name: Web Technology

Day: 30

Topics Covered: JSP- Action Tags

Faculty Name: Dr. Avinash Dwivedi Programe Name: B.Tech (CSE,AI &ML)

Prerequisites, Objectives and Outcomes

Prerequisite of topic: Basic concepts related to web programming

Objective: To make students aware about the server side programming using JSP.

Outcome : 1. Students will be able to use JSP as a server side technology.

2. Students will be able to use web server along with deployment of application

3. Students will be able to implement in practical applications.

Java Server Pages

JSP Action Tags

- There are many JSP action tags or elements. Each JSP action tag is used to perform some specific tasks.
- 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.

jsp:forward action tag

- The jsp:forward action tag is used to forward the request to another resource it may be jsp, html or another resource.
- Syntax of jsp:forward action tag without parameter
- <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>

Example of jsp:forward action tag without parameter

- In this example, we are simply forwarding the request to the printdate.jsp file.
- index.jsp
- <html> <body>
- <h2>**this** is index page</h2>
- **<jsp:forward page="printdate.jsp" />**
- </body>
- </html>
- printdate.jsp
- <html>
- <body>
- <% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
- </body>
- </html>

Example of jsp:forward action tag with parameter

- In this example, we are forwarding the request to the printdate.jsp file with parameter and printdate.jsp file prints the parameter value with date and time.
- **index.jsp**
- <html> <body> <h2>**this** is index page</h2>
- <jsp:forward page="printdate.jsp" >
- <jsp:param name="name" value=“google.com" />
- </jsp:forward>
- </body> </html>
- **printdate.jsp**
- <html>
- <body>
- <% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
- <%= request.getParameter("name") %>
- </body> </html>

jsp:include action tag

- 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.

Advantage of jsp:include action tag

- **Code reusability** : We can use a page many times such as including header and footer pages in all pages. So it saves a lot of time.

Difference between jsp include directive and include action

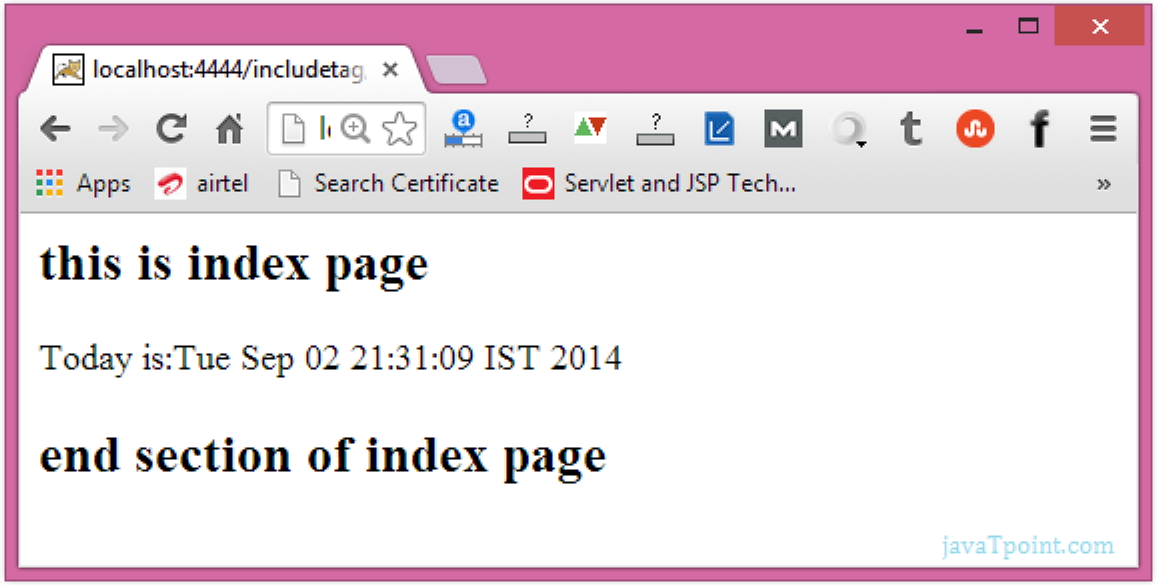
JSP include directive	JSP include action
includes resource at translation time.	includes resource at request time.
better for static pages.	better for dynamic pages.
includes the original content in the generated servlet.	calls the include method.

Syntax

- Syntax of jsp:include action tag without parameter
- <jsp:include page="relativeURL | <%= expression %>" />
- Syntax of jsp:include action tag with parameter
- <jsp:include page="relativeURL | <%= expression %>" >
- <jsp:param name="parametername" value="parametervalue | <%=expression%>" />
- </jsp:include>

Example of jsp:include action tag without parameter

- ***File: index.jsp***
- <h2>**this** is index page</h2>
- <jsp:include page="printdate.jsp" />
- <h2>end section of index page</h2>
- ***File: printdate.jsp***
- <% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>



this is index page

Today is:Tue Sep 02 21:31:09 IST 2014

end section of index page

Java Bean

- A Java Bean is a java class that should follow following conventions:
- It should have a no-arg constructor.
- It should be Serializable (It means that instances of the class can be turned into a byte-stream (for example, to be saved to a file) and then converted back into classes again)).

- It should provide methods to set and get the values of the properties, known as getter and setter methods.
- it is a reusable software component. A bean encapsulates many objects into one object, so we can access this object from multiple places. Moreover, it provides the easy maintenance.

Program of java bean class

- //Employee.java
- **package** mypack;
- **public class** Employee **implements** java.io.Serializable{
- **private int** id;
- **private** String name;
- **public** Employee(){}
- **public void** setId(**int** id){**this**.id=id;}
- **public int** getId(){**return** id;}
- **public void** setName(String name){**this**.name=name;}
- **public** String getName(){**return** name;} }

How to access the java bean class

- To access the java bean class, we should use getter and setter methods.
- **package** mypack;
- **public class** Test{
- **public static void** main(String args[]){
-   Employee e=**new** Employee();//object is created
-   e.setName("Arjun");//setting value to the object
-   System.out.println(e.getName());
-   }}
- Note: There are two ways to provide values to the object, one way is by constructor and second is by setter method.

jsp:useBean action tag

- The jsp:useBean action tag is used to locate or instantiate a bean class. If bean object of the Bean class is already created, it doesn't create the bean depending on the scope. But if object of bean is not created, it instantiates the bean.

- Syntax of jsp:useBean action tag

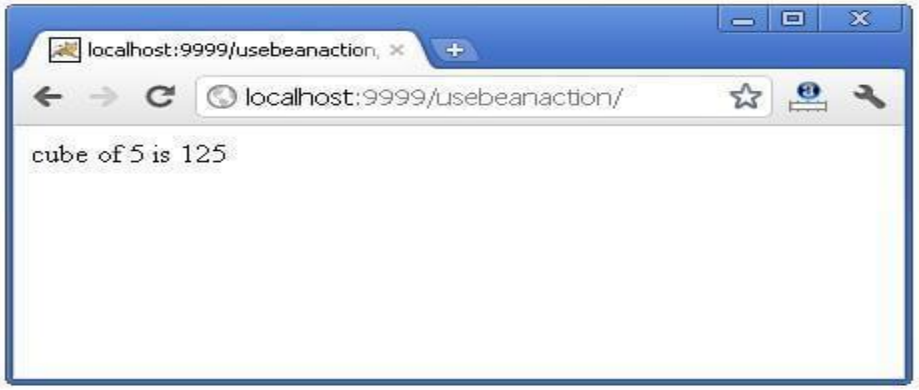
- <jsp:useBean id= "instanceName" scope= "page | request | session | appli cation"
- **class**= "packageName.className" type= "packageName.className"
- beanName="packageName.className | <%= expression >" >
- </jsp:useBean>

Attributes and Usage of jsp:useBean action tag

- **id:** is used to identify the bean in the specified scope.
- **scope:** represents the scope of the bean. It may be page, request, session or application. The default scope is page.
  - **page:** specifies that you can use this bean within the JSP page. The default scope is page.
  - **request:** specifies that you can use this bean from any JSP page that processes the same request. It has wider scope than page.
  - **session:** specifies that you can use this bean from any JSP page in the same session whether processes the same request or not. It has wider scope than request.
  - **application:** specifies that you can use this bean from any JSP page in the same application. It has wider scope than session.
- **class:** instantiates the specified bean class (i.e. creates an object of the bean class) but it must have no-arg or no constructor and must not be abstract.
- **type:** provides the bean a data type if the bean already exists in the scope. It is mainly used with class or beanName attribute. If you use it without class or beanName, no bean is instantiated.
- **beanName:** instantiates the bean using the java.beans.Beans.instantiate() method.

Program of jsp:useBean action tag

- In this example, we are simply invoking the method of the Bean class.
- Calculator.java (a simple Bean class)
- **package** abc;
- **public class** Cubev{
- **public int** cube(**int** n){**return** n*n*n;}
- }
- index.jsp file
- <jsp:useBean id="obj" **class**=“abc.Cubev"/>
- <%
- **int** m=obj.cube(5);
- out.print("cube of 5 is "+m);
- %>



```
cube of 5 is 125
```

jsp:setProperty and jsp:getProperty action tags

- In web devlopment, bean class is mostly used because it is a reusable software component that represents data.
- The jsp:setProperty action tag sets a property value or values in a bean using the setter method.

- Syntax of jsp:setProperty action tag
- <jsp:setProperty name="instanceOfBean" property= "*" |
- property="propertyName" param="parameterName" |
- property="propertyName" value="{ string | <%= expression % >}"
- />

Example of jsp:setProperty action tag

if you have to set all the values of incoming request in the bean

- <jsp:setProperty name="bean" property="*" />
- Example of jsp:setProperty action tag if you have to set value of the incoming **specific property**
- <jsp:setProperty name="bean" property="username" />
- Example of jsp:setProperty action tag if you have to set a **specific value in the property**
- <jsp:setProperty name="bean" property="username" value="Kumar" />

jsp:getProperty action tag

- The jsp:getProperty action tag returns the value of the property.
- Syntax of jsp:getProperty action tag
- <jsp:getProperty name="instanceOfBean" property=" propertyName" />
- Simple example of jsp:getProperty action tag
- <jsp:getProperty name="obj" property="name" />

- Note we do not need to define setter getter methods

Example of bean development in JSP

- In this example there are 3 pages:
- index.html for input of values
- welocme.jsp file that sets the incoming values to the bean object and prints the one value
- User.java bean class that have setter and getter methods
- **index.html**
- <form action="process.jsp" method="post">
- Name:<input type="text" name="name"><br>
- Password:<input type="password" name="password"><br>
- Email:<input type="text" name="email"><br>
- <input type="submit" value="register">
- </form>

process.jsp

- <jsp:useBean id="u" **class**="org.sssit.User"></jsp:useBean>
- <jsp:setProperty property="*" name="u"/>
-   Record:<br>
- <jsp:getProperty property="name" name="u"/><br>
- <jsp:getProperty property="password" name="u"/><br>
- <jsp:getProperty property="email" name="u" /><br>
- User.java
- **package** org.sssit;
-
- **public class** User {
- **private** String name,password,email;
- //setters and getters
- }

Reusing Bean in Multiple Jsp Pages

- **index.html**
- <form action="process.jsp" method="post">
- Name:<input type="text" name="name"><br>
- Password:<input type="password" name="password"><br>
- Email:<input type="text" name="email"><br>
- <input type="submit" value="register">
- </form>

- User.java
- **package** org.sssit;
- **public class** User {
- **private** String name,password,email;
- //setters and getters
- }

process.jsp

<jsp:useBean id="u" **class**="org.sssit.User" scope="session"></jsp:useBean>

- <jsp:setProperty property="*" name="u"/>
- Record:<br>
- <jsp:getProperty property="name" name="u"/><br>
- <jsp:getProperty property="password" name="u"/><br>
- <jsp:getProperty property="email" name="u" /><br>
- <a href="second.jsp">Visit Page</a>
- second.jsp
- <jsp:useBean id="u" **class**="org.sssit.User" scope="session"></jsp:useBean
  >
- Record:<br>
- <jsp:getProperty property="name" name="u"/><br>
- <jsp:getProperty property="password" name="u"/><br>
- <jsp:getProperty property="email" name="u" /><br>

Using variable value in setProperty tag

- In some case, you may get some value from the database, that is to be set in the bean object, in such case, you need to use expression tag. For example:
- **process.jsp**
- <jsp:useBean id="u" **class**="org.sssit.User"></jsp:useBean>
- <%
- String name="akash";
- %>
- <jsp:setProperty property="name" name="u" value="<%=name %>"/>
- Record:<br>
- <jsp:getProperty property="name" name="u"/><br>

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.
- JSP tag extensions lets you create new tags that you can insert directly into a JavaServer Page. The JSP 2.0 specification introduced the Simple Tag Handlers for writing these custom tags.
- To write a custom tag, you can simply extend **SimpleTagSupport** class and override the **doTag()** method, where you can place your code to generate content for the tag.

Create "Hello" Tag

- Consider you want to define a custom tag named <ex:Hello> and you want to use it in the following fashion without a body
- <ex:Hello /> To create a custom JSP tag, you must first create a Java class that acts as a tag handler.

```
  package abc;
  import javax.servlet.jsp.tagext.*;
  import javax.servlet.jsp.*;
  import java.io.*;
  public class HelloTag extends SimpleTagSupport {
public void doTag() throws JspException, IOException {
   JspWriter out = getJspContext().getOut();
   out.println("Hello ABESIT Tag!");
}}
```

The above code has simple coding where the **doTag()** method takes the current JspContext object using the **getJspContext()** method and uses it to send **"Hello Custom Tag!"** to the current **JspWriter** object

Tld file

- compile the above class and copy it in a directory available in the environment variable CLASSPATH. Finally, create the following tag library file:
- <taglib>
- <tlib-version>1.0</tlib-version>
- <jsp-version>2.0</jsp-version>
- <short-name>Example TLD</short-name>
- <tag>
- <name>Hello</name>
- <tag-class>abc.HelloTag</tag-class>
- <body-content>empty</body-content>
- </tag></taglib>

- Let us now use the above defined custom tag Hello in our JSP program as follows −

- <%@ taglib prefix = "ex" uri = "WEB-INF/custom.tld"%>
- <html>
-    <head>   <title>A sample custom tag</title>
-    </head>
-    <body>
-       <ex:Hello/>
-    </body></html>
- Output: Hello ABESIT Tag

Accessing the Tag Body

- We can include a message in the body of the tag as you have seen with standard tags. Consider you want to define a custom tag named **<ex:Hello>**and you want to use it in the following fashion with a body –
- <ex:Hello>

  This is message body
- </ex:Hello>

Tag class

- package abc;
- import javax.servlet.jsp.tagext.*;
- import javax.servlet.jsp.*;
- import java.io.*;
- public class **HelloTag** extends SimpleTagSupport {
-     StringWriter sw = new StringWriter();
-     public void doTag() throws JspException, IOException {
-         getJspBody().invoke(sw);
-         getJspContext().getOut().println(sw.toString());
-     } }

-
  ```
  <taglib>
      <tlib-version>1.0</tlib-version>
      <jsp-version>2.0</jsp-version>
      <short-name>Example TLD with Body</short-name>
      <tag>
          <name>Hello</name>
          <tag-class>abc.HelloTag</tag-class>
          <body-content>scriptless</body-content>
      </tag>
  </taglib>
  ```

Jsp page

```
• <%@ taglib prefix = "ex" uri = "WEB-INF/custom.tld"%>
• <html>   <head>      <title>A sample custom tag</title>
•   </head>
•   <body>
•     <ex:Hello>
•       This is message body
•     </ex:Hello>
•   </body></html>
```

- Output: This is message body

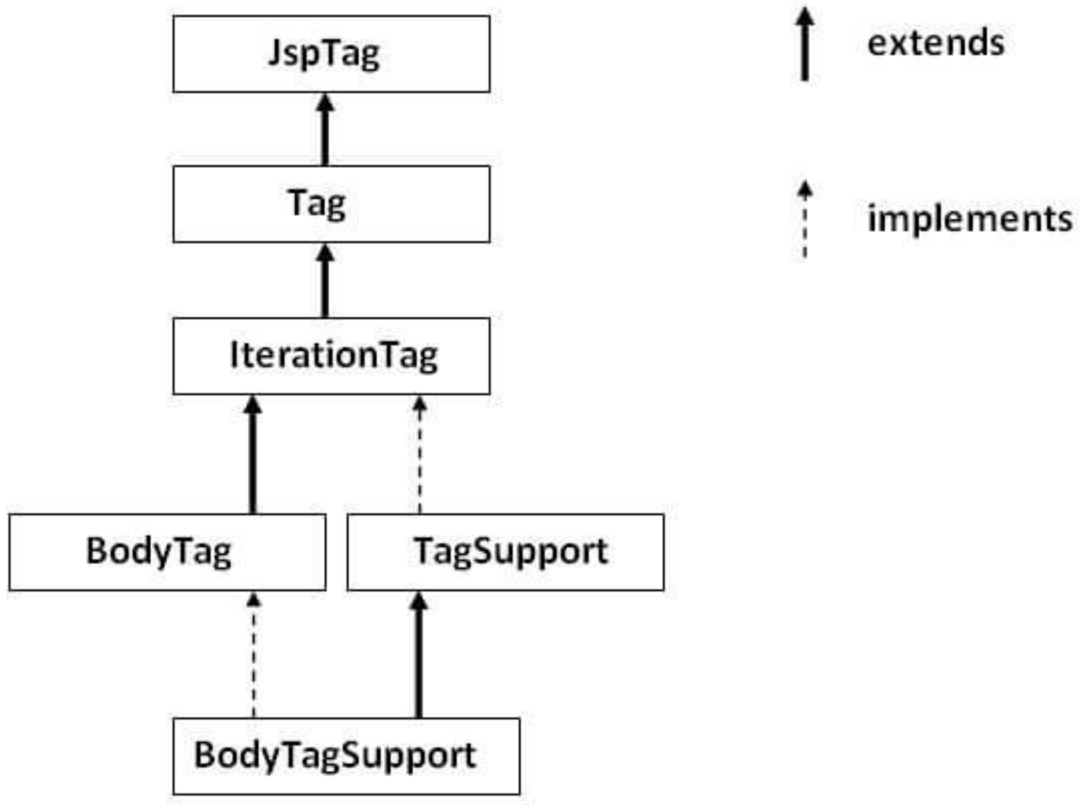
Custom Tags in JSP

- **Custom tags** are user-defined tags. They eliminates the possibility of scriptlet tag and separates the business logic from the JSP page.
- The same business logic can be used many times by the use of custom tag.
- Advantages of Custom Tags
- The key advantages of Custom tags are as follows:
- **Eliminates the need of scriptlet tag** The custom tags eliminates the need of scriptlet tag which is considered bad programming approach in JSP.
- **Separation of business logic from JSP** The custom tags separate the the business logic from the JSP page so that it may be easy to maintain.
- **Re-usability** The custom tags makes the possibility to reuse the same business logic again and again.

Syntax to use custom tag

- There are two ways to use the custom tag. They are given below:
- **<prefix:tagname** attr1=value1....attrn=valuen **/>**
- **<prefix:tagname** attr1=value1....attrn=valuen **>**
- body code
- **</prefix:tagname>**

JSP Custom Tag API



jspTag and Tag Interface

- **JspTag interface**
- The JspTag is the root interface for all the interfaces and classes used in custom tag. It is a marker interface.
- **Tag interface**
- The Tag interface is the sub interface of JspTag interface. It provides methods to perform action at the start and end of the tag.

Fields of Tag interface

- There are four fields defined in the Tag interface. They are:

Field Name	Description
public static int EVAL_BODY_INCLUDE	it evaluates the body content.
public static int EVAL_PAGE	it evaluates the JSP page content after the custom tag.
public static int SKIP_BODY	it skips the body content of the tag.
public static int SKIP_PAGE	it skips the JSP page content after the custom tag.

Methods of Tag interface

Method Name	Description
public void setPageContext(PageContext pc)	it sets the given PageContext object.
public void setParent(Tag t)	it sets the parent of the tag handler.
public Tag getParent()	it returns the parent tag handler object.
public int doStartTag()throws JspException	it is invoked by the JSP page implementation object. The JSP programmer should override this method and define the business logic to be performed at the start of the tag.
public int doEndTag()throws JspException	it is invoked by the JSP page implementation object. The JSP programmer should override this method and define the business logic to be performed at the end of the tag.
public void release()	it is invoked by the JSP page implementation object to release the state.

IterationTag interface

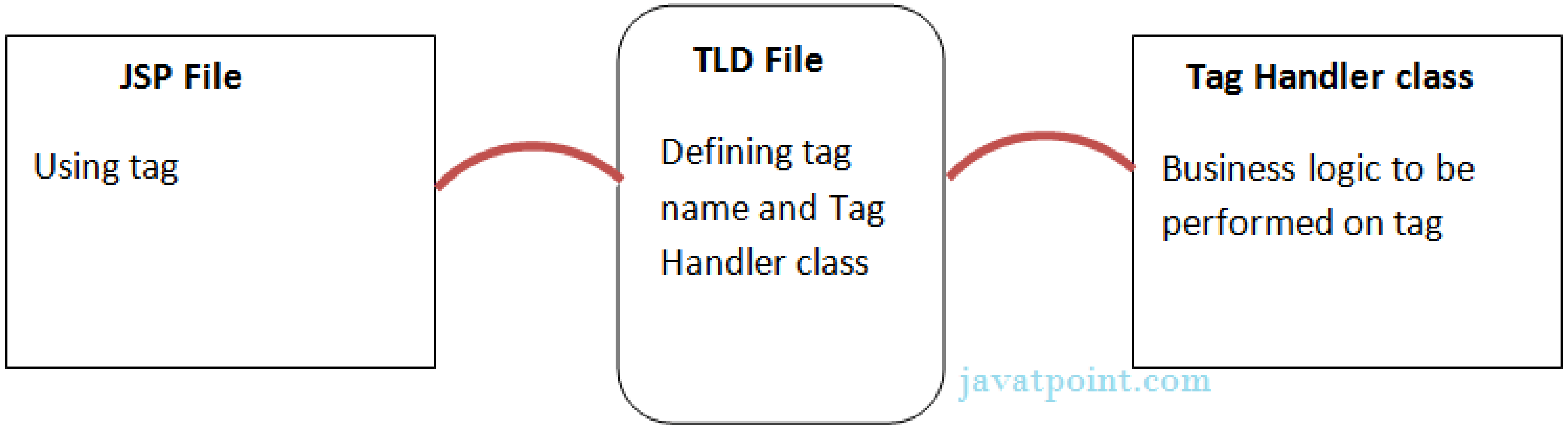
- The IterationTag interface is the sub interface of the Tag interface. It provides an additional method to reevaluate the body.
- Field of IterationTag interface
- There is only one field defined in the IterationTag interface.
- **public static int EVAL_BODY_AGAIN** it reevaluates the body content.
- Method of Tag interface
- There is only one method defined in the IterationTag interface.
- **public int doAfterBody()throws JspException** it is invoked by the JSP page implementation object after the evaluation of the body. If this method returns EVAL_BODY_INCLUDE, body content will be reevaluated, if it returns SKIP_BODY, no more body cotent will be evaluated.

- **TagSupport class**
- The TagSupport class implements the IterationTag interface. It acts as the base class for new Tag Handlers. It provides some additional methods also.

Program of JSP Custom Tag

- a **custom tag that prints the current date and time**. We are performing action at the start of tag.
- For creating any custom tag, we need to follow following steps:
- **Create the Tag handler class** and perform action at the start or at the end of the tag.
- **Create the Tag Library Descriptor (TLD) file** and define tags
- **Create the JSP file that uses the Custom tag defined in the TLD file**

Flow in Custom Tag


JSP File
Using tag
TLD File
Defining tag name and Tag Handler class
Tag Handler class
Business logic to be performed on tag

1) Create the Tag handler class

- To create the Tag Handler, we are inheriting the **TagSupport class** and overriding its method **doStartTag()**.To write data for the jsp, we need to use the **JspWriter class**.
- The **PageContext** class provides **getOut()** method that returns the instance of JspWriter class. TagSupport class provides instance of pageContext bydefault.

File: MyTagHandler.java

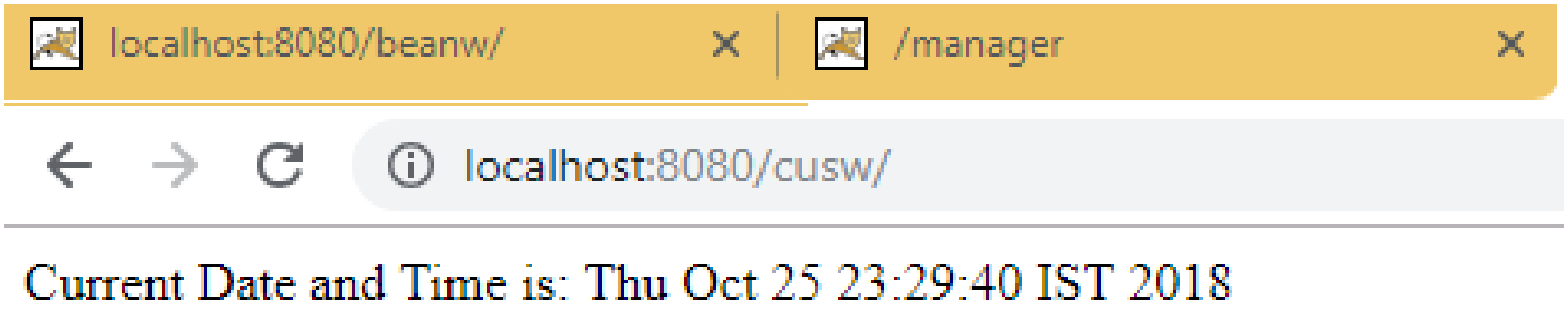
```
•  package abc;
•  import java.util.Calendar;
•  import javax.servlet.jsp.JspException;
•  import javax.servlet.jsp.JspWriter;
•  import javax.servlet.jsp.tagext.TagSupport;
•  public class MyTagHandler extends TagSupport{
•
•  public int doStartTag() throws JspException {
•      JspWriter out=pageContext.getOut();//returns the instance of JspWriter
        try{
•       out.print(Calendar.getInstance().getTime());//printing date and time usi
   ng JspWriter
•      }catch(Exception e){System.out.println(e);}
•      return SKIP_BODY;//will not evaluate the body content of the tag
•  }  }
```

Create the TLD file

- **Tag Library Descriptor** (TLD) file contains information of tag and Tag Hander classes. It must be contained inside the **WEB-INF**directory.
- *File: mytags.tld*
- **<?xml** version="1.0" encoding="ISO-8859-1" **?>**
- <!DOCTYPE taglib
-         PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
-     "http://java.sun.com/j2ee/dtd/web-jsptaglibrary_1_2.dtd"**>**
-
- **<taglib>**
-     **<tlib-version>**1.0**</tlib-version>**
-   **<jsp-version>**1.2**</jsp-version>**
-   **<short-name>**simple**</short-name>**
-   **<uri>**http://tomcat.apache.org/example-taglib**</uri>**
-   **<tag>**
- **<name>**today**</name>**
- **<tag-class>**com.javatpoint.sonoo.MyTagHandler**</tag-class>**
- **</tag> </taglib>**

3) Create the JSP file

- Let's use the tag in our jsp file. Here, we are specifying the path of tld file directly. But it is recommended to use the uri name instead of full path of tld file. We will learn about uri later.
- It uses **taglib** directive to use the tags defined in the tld file.
- *File: index.jsp*
- <%@ taglib uri="WEB-INF/mytags.tld" prefix="m" %>
- Current Date and Time is: <m:today/>



```
Current Date and Time is: Thu Oct 25 23:29:40 IST 2018
```

Attributes in JSP Custom Tag

- There can be defined too many attributes for any custom tag. To define the attribute, you need to perform two tasks:
- Define the property in the TagHandler class with the attribute name and define the setter method
- define the attribute element inside the tag element in the TLD file
- Let's understand the attribute by the tag given below:
- **<m:cube** number="4"**></m:cube>**
- Here **m** is the prefix, **cube** is the tag name and **number** is the attribute.

Simple example of attribute in JSP Custom Tag

- In this example, we are going to use the cube tag which return the cube of any given number. Here, we are defining the number attribute for the cube tag. We are using the three file here:
- index.jsp
- CubeNumber.java
- mytags.tld
- **index.jsp**
- **<**%@ taglib uri="WEB-INF/mytags.tld" prefix="m" %**>**
- Cube of 4 is: **<m:cube** number="4"**></m:cube>**
-

CubeNumber.java

```
package com.javatpoint.taghandler;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.TagSupport;

public class CubeNumber extends TagSupport{
private int number;

public void setNumber(int number) {
    this.number = number;
}

public int doStartTag() throws JspException {
    JspWriter out=pageContext.getOut();
    try{
        out.print(number*number*number);
    }catch(Exception e){e.printStackTrace();}
         return SKIP_BODY;  }  }
```

mytags.tld

- <?xml version="1.0" encoding="ISO-8859-1" ?>
- <!DOCTYPE taglib
-         PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
-         "http://java.sun.com/j2ee/dtd/web-jsptaglibrary_1_2.dtd">
-
- <taglib>
-   <tlib-version>1.0</tlib-version>
-   <jsp-version>1.2</jsp-version>
-   <short-name>simple</short-name>
-   <uri>http://tomcat.apache.org/example-taglib</uri>
-   <description>A simple tab library for the examples</description>
-     <tag>
-     <name>cube</name>
-     <tag-class>com.javatpoint.taghandler.CubeNumber</tag-class>
-     <attribute>
-     <name>number</name>
-     <required>true</required>
-     </attribute>   </tag> </taglib>