Java Server Pages - JSP

LEARNING OBJECTIVES

At the end of this lesson, you will be able to:

- Introduction
- Elements of JSP
- Implicit Objects
- Using Java Beans in JSP
- Custom Tags

JSP – Java Server Pages

- Java technology to create web pages with dynamic content
- Server-side scripting language
- Bits of Java code embedded in HTML
- Suitable for coding the presentation layer of an enterprise application
- Created and maintained by web designers
- Has access to all Java APIs and J2EE APIs
- Has inherent Platform independence

Servlet Code to print 'Hello World':

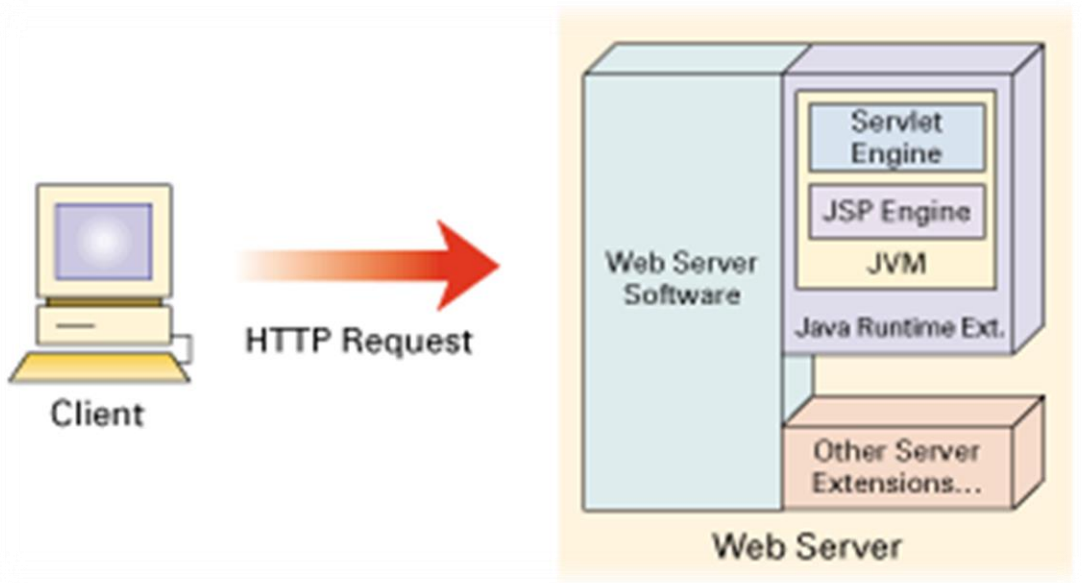
```
import java.io.*;
import javax.servlet.*;;
import javax.servlet.http.*;

public class NewServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse
response)
   throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();
      try {
            out.println("<html>");
         out.println("<head>");
         out.println("<title>Servlet NewServlet</title>");
         out.println("</head>");
         out.println("<body>");
         out.println("Hello world");
         out.println("</body>");
         out.println("</html>");

      } finally {
         out.close();
      }
   } }
```

JSP Equivalent Code:

```
<%@ page language="java"%>
 <HTML>
   <HEAD>
      <TITLE>Servlet NewServlet</TITLE>
   </HEAD>
   <BODY>
      <%out.print("Hello World");%>
   </BODY>
 </HTML>
```

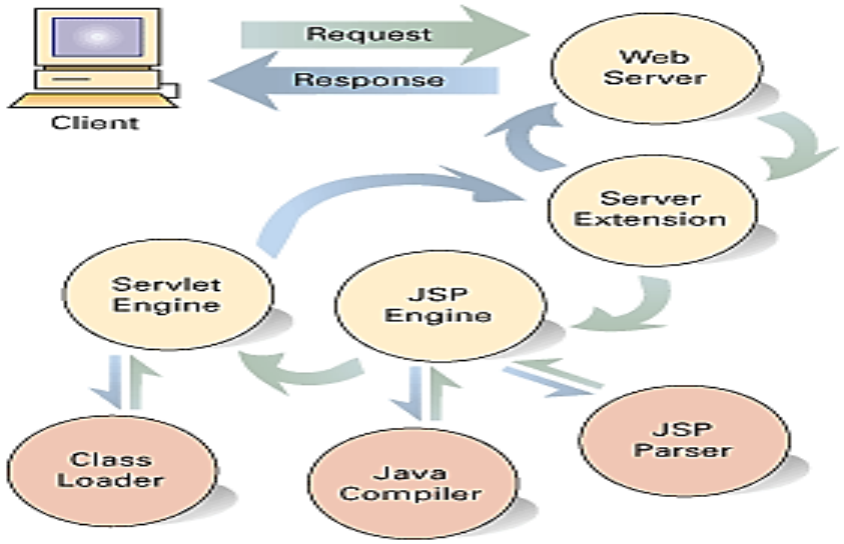
JSP Engine

- HTTP requests are sent to the web server from a browser client
- if the request is for a JSP, JSP engine responds



Life Cycle of a JSP Page

- Request comes to a web server
- Server passes it on to the JSP engine
- JSP engine translates the JSP in to equivalent Servlet Java code
- JSP engine invokes Java compiler to compile the JSP into an servlet class.
- Control is passed on to a servlet engine.
- The servlet engine loads the servlet into memory
- Appropriate methods are invoked
- Response is routed back to the browser



Methods of Generated Servlet

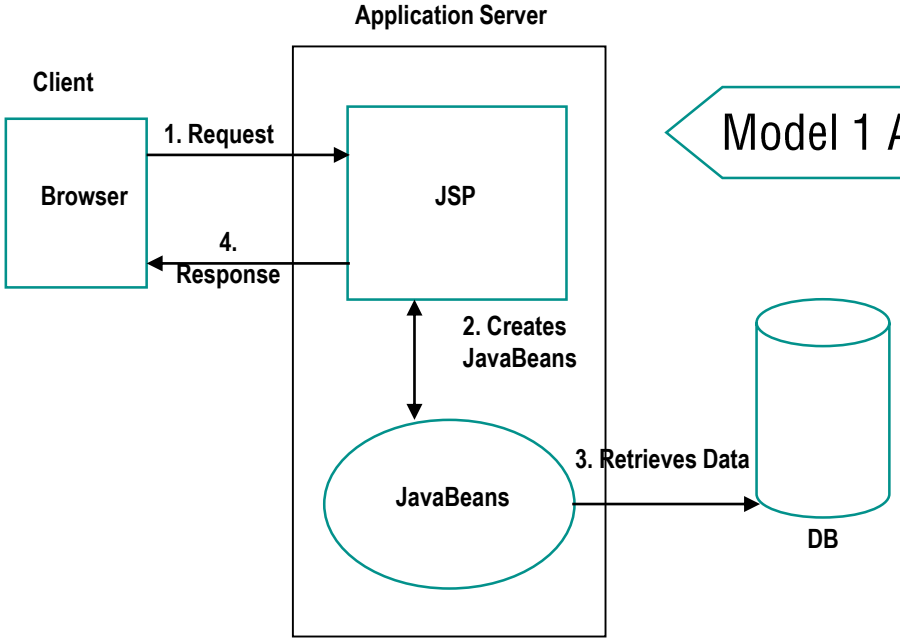
Generated Servlet equivalent Class has following Methods:

- jspInit() :
  - The container calls the jspInit() to initialize the servlet instance.
  - It is called before any other method, and is called only once for a servlet instance
  - Has access to ServletConfig and ServletContext
  - It can be overridden in a JSP.

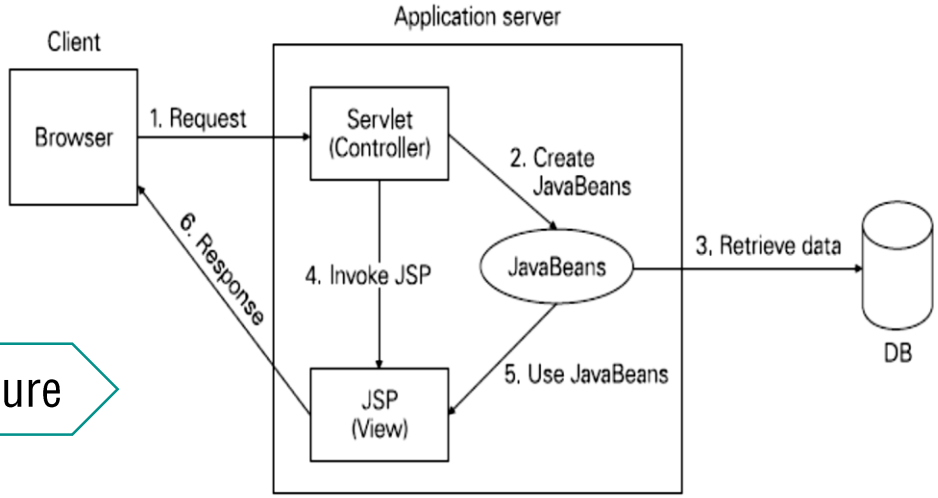
- _jspservice():
  The container calls this method for each request, passing it the request and response object.
  Cannot be overridden

- jspDestroy() :
  - The container calls this to take the instance out of service
  - It is the last method called in the servlet instance.
  - Can be overridden

JSP Architecture Models



Model 1 Architecture



Model 2 Architecture

Elements of a JSP Page

Three main types of elements that can be embedded in a JSP:

- Scripting elements:
  JSP scripting elements let you insert Java code in the JSP

  ```
  <% Java Statement %>
  ```

- Directives
  Directive is a way to give special instructions to the Container at page translation time

  ```
  <%@ directive attribute="value" %>
  ```

- Actions
  JSP Actions tags enables the programmer to use predefined functions.
  JSP Actions are XML tags that can be used in the JSP page

  ```
  <jsp:action_name attribute="value" />
  ```

Scripting Elements

- Scripting elements are used to insert Java code into the servlet that will be generated from the JSP page

  - Expressions

    Shorthand for printing out strings and contents of variables

    <%= Java Expression/variable%>

  - Scriptlets

    Insert any valid Java code into the JSP

    <% Java Code %>

  - Declarations

    Declaring Java variables and methods

    <%! Field/Method Declaration %>

Expressions

- A JSP expression is used to insert Java values directly into the output

```
<%= Java Expression %>
<jsp:expression> Java Expression </jsp:expression>
```

- Expression is evaluated, converted to a string, and inserted in the page

```
<p>
        Current time       :  <%= new java.util.Date() %>
        Random number : <%= Math.random() %>
        Server                 :
        <jsp:expression> application.getServerInfo() </jsp:expression>
</p>
```

Implicit Objects

- Implicit objects are declared and assigned by Container at the beginning of the service method during Translation

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

Implicit objects

- request
  - request is a reference to the HttpServletRequest object
  - Helps to get the request parameters (via getParameter), request attributes, headers etc
  - Passed to the service method by the Container
- response
  - response object is a reference to the HttpServletResponse object
  - Encapsulates the response that is sent back to the client
  - Passed as parameter to service method
- session
  - HttpSession object associated with the request
  - sessions are created automatically in JSP
- out
  - buffered version of PrintWriter called JspWriter
  - used to send output to the client

Implicit Objects

- application
  - Instance of ServletContext got by getServletConfig().getServletContext()
  - Contains methods that provide information about the context

- config
  - Represents Servlet configuration
  - Instance of javax.servlet.ServletConfig

- pageContext
  - Provides access to all the implicit objects associated with a JSP page
  - Encapsulates other implicit objects
  - Can be used to get or set attributes in any scope
  - Can be used to forward requests to other web components

    findAttribute(String name**)**

    Searches for attribute in page, request, session and application scope in order and returns the value associated or null.

- page
  - Instance of java.lang.Object
  - Refers to the current servlet instance "this"

Declarations

- Declarations are used to define methods or fields
- Get inserted into the main body of the servlet class (outside of the service method)
- Initialized when JSP is initialized
- The scope of a declaration is in a JSP file and any included files in JSP

Syntax: <%! Declaration %>

```
<%! int count = 0;%>

<%! Public double getArea(int r){
        double area = Math.PI*r*r
        return area;
    }
 %>
```

```
<%!
  public  com.test.Employee   GetEmployee(int id, String name){
        com.test.Employee e1 = new com.test.Employee();
        e1.setId(id);
        e1.setName(name);
        return e1;
   }
%>
```

Scriplets

- Block of valid Java code between <% and %> tags
- Executes when the service method gets executed

```
<% out.println("Hello");%>
 <% String empName = request.getParameter("name"));
      out.println(empName);
%>
 <% com.test.Employee emp1 = getEmployee(1,"Jimmy");
      com.test.Employee emp2 = getEmployee(2,"Cathy");
      com.test.Employee[] empArr = new com.test.Employee[2]
      empArr[0]= emp1; empArr[1] = emp2;
      for(com.test.Employee e:empArr){
          out.println(e.id);
          out.println(e.name);
      }
 %>
```

- Declarations are executed at initialization time
- Expressions and Scriptlets are executed at request Time

Deployment descriptor

```
<jsp-config>
   <jsp-property-group>
            <url-pattern>*.jsp</url-pattern>
            <scripting-invalid>true</scripting-invalid>
   </jsp-property-group>
</jsp-config>
```

- Scripting elements are disabled
- Translation error occurs, If JSP contains JSP scriptlet, expression, or declaration

```
<servlet>
   <servlet-name>test</servlet-name>
   <jsp-file>/library/checkout.jsp</jsp-file>
   <load-on-startup>3</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>test</servlet-name>
    <url-pattern>/library/*</url-pattern>
</servlet-mapping>
```

Generated Servlet Representation

This is just a representation not actual generated code. Check Tomcat directory for Actual code

```
<html>
<body>
   <%! int doubleCount() {
        count = count*2;
        return count;
        }
    %>
    <%! int count=1; %>
   The page count is now:
   <%= doubleCount() %>
   <% out.println(doubleCount() )%>
</body>
</html>
```

```
public class test_jsp extends SomeSpecialHttpServlet {
        int doubleCount() {
        count = count*2;
        return count;
        }

        int count=1;

        public void _jspService(HttpServletRequest request, HttpServletResponse
        response)throws java.io.IOException {

             PrintWriter out = response.getWriter();
             response.setContentType("text/html");
             out.write("<html><body>");
             out.write("The page count is now:");
             out.print( doubleCount());
             out.println(doubleCount());
             out.write("</body></html>");

        }
}
```


19

JSP Comments

- Comments

  JSP Comments

  Syntax

  <% -- comments --%>

  HTML Comments

  Syntax

  <! -- comments -- !>

Directives

- Directive is a way for you to provide special instructions to the Container during page translation
- Affects the overall translation of the JSP page
- Three types
  - **page**
  - **include**
  - **taglib**

Syntax:

```
<%@directivename   [attribute name="value"
                attributename="value" ........]%>
<jsp:directive. directivename  ...../>
```

Example:

```
<%@page import="java.io.*"%>
<jsp:directive.page import="java.io.*" />
```

Page directive

- Defines page-specific properties
- Page Directives can be used to

  - Import classes using import attribute

```
<%@page import="java.util.*"%>
<%@page import="java.io.* ,java.util.*"%>
```

  - Define if jsp should not take part in a session

```
<%@ page session="false" %>
```

  - Set the page content type using contentType attribute

```
<%@ page contentType="text/html" %>
```

Page directive

- Handle error messages
  - specifies JSP page which will be displayed in case of error/exception

```
<%@ page errorPage="Relative URL" %>
```

  - isErrorPage attribute indicates whether the current page can act as the error page for another JSP page with exception implicit object available to the jsp

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

Makes the implicit object 'exception' available to JSP. which by default is not available to a JSP

Include directive

- Inserts contents of another file in the main JSP file at the location of the directive
- Useful for including header, Footer, copyright information etc.
- Promotes reuse
- The included file can be an HTML file, a JSP file, a text file, or a code file written in the Java programming language.
- Container includes the content of the file during Translation

Syntax:

```
<%@include file="file name" %>
```

Example:

```
<%@include file="copyright.html"%>
<jsp:directive.include file="copyright.html" />
```

JSP Actions

- Container provides many built in functions to ease the development
- JSP Actions tags enables the programmer to use predefined functions
- JSP Actions are XML tags that can be used in the JSP page
- Helps in eliminating scripting code in JSP

Action tag	Use
jsp:include	Used for dynamic inclusion
jsp:param	Used to add the specific parameter to current request
jsp:forward	Used to hand off the request and response to another JSP or servlet
jsp:useBean	Used to specify the Java bean
jsp:getProperty	Used to get property of a java bean object
jsp:setProperty	Used to set property of a java bean object

JSP Include action

- Action tag insert files into the page being generated

```
<jsp:include page="relative URL"/>
```

- Action tag inserts the file at the runtime
- Whereas include directive inserts the file during JSP translation
- Provides flexibility with a small hit on efficiency

JSP Forward Action

- Forwards a client request to an JSP, servlet or and HTML file for processing

```
<jsp:forward page= "relativeURL" />
```

- Has a single attribute, page, which consist of a relative URL
- Attribute could be a static value, or could be computed at request time

```
<jsp:forward page="/bank/credit.jsp" />
<jsp:forward page="<%= JavaExpression %>" />
```

In an include or forward element, parameters can be appended to the request object by using the jsp:param element

```
<jsp:forward page="..." >
        <jsp:param name="param1" value="value1"/>
</jsp:forward>
```

Java Beans

- An architecture for using and building components in Java
- Reusable software components that can be integrated to create software applications
- Classes conforming to a particular convention
- The required conventions are as follows
  - The class must have a public default constructor (no-argument).
  - The class properties must be accessible using get, is, set following a standard naming convention
  - The class should be Serializable

Employee Bean

```
import java.io.Serializable;
public class Employee implements Serializable{
        int employee_id;
        String employee_Name;
        boolean indian;
        public int getEmployee_id() {
                return employee_id;
        }
        public void setEmployee_id(int employee_id) {
                this.employee_id = employee_id;
        }
        public String getEmployee_Name() {
                return employee_Name;
        }
        public void setEmployee_Name(String employee_Name) {
                this.employee_Name = employee_Name;
        }
        public boolean isIndian() {
                return indian;
        }
        public void setIndian(boolean indian) {
                this.indian = indian;
        }
}
```

A No Argument constructor should be present in a Java bean. Java compiler provides no arg constructor, if programmer has not coded any other constructor

Action tag for Java Beans

- Can be instantiated in JSP using the

  <jsp:useBean> tag

- <jsp:setProperty> is used to set the property of the bean

- <jsp:getproperty> is used to get the property of the bean

- To use a java bean
  - Create a java bean class
  - Place it in the web server
  - Create the JSP page
  - Call the java bean from the JSP page

JSP UseBean

Declaring bean

```
<jsp:useBean id="employee" class="com.test.Employee" scope="request" />
```

If useBean can't find an attribute object named "employee", it will make new bean

Getting empl_id using getProperty

```
<jsp:getProperty name="employee" property="empl_id" />
```

Setting empl_id using setProperty

```
        <jsp:setProperty name=" employee" property="empl_name"
value="Charles" />
```

Setting property employee name in case bean doesn't exist

```
        <jsp:useBean id="employee" class="com.test.Employee" scope="page" >
        <jsp:setProperty name=" employee" property=" empl_name"
value="Charles" />
        </jsp:useBean >
```

Scripting Vs Standard Actions

Scripting

```
<html> <body>
        <% com.test.Employee e1 = (com.test.Employee) request.getAttribute("empl"); %>
        Employee Name is: <%= e1.getName() %>
  </body> </html>
```

With Standard Action

```
<html>  <body>
        <jsp:useBean id="empl" class="com.test.Employee" scope="request" />
        Employee Name is: <jsp:getProperty name="empl" property="name"/>
</body>  </html>
```

Using Polymorphic reference (Employee extends abstract class Person)

```
    <jsp:useBean id="employee" type="com.test.Person"
class="com.test.Employee" scope="page">
```

Passing request to JSP: Using param

- Using param to set the property value

```
<%  int quantity = 0;
      double tax = 0.0;
      try {
          quantity = Integer.parseInt(request.getParameter("prodQuantity"));
          tax = Double.parseDouble(request.getParameter("prodTax"));
      } catch (NumberFormatException nfe) {
      }
%>
```

Passing request to JSP: Using param

- Using param to set the property value

<jsp:setProperty name="getBill" property="prodQuantity"value="<%= quantity %>" />
<jsp:setProperty name="getBill" property="prodTax" value="<%= tax %>" />

<jsp:setProperty name="getBill" property="prodQuantity" **param="prodQuantity" />**
<jsp:setProperty name="getBill" property="prodTax" **param="prodTax"** />

If parameter name and the bean property name is same
<jsp:setProperty name="getBill" property="prodTax" />

If all the parameters have the same name as the bean properties
<jsp:setProperty name="getBill" property="*" />

Custom Tags

- User-defined JSP language element
- Eliminates the possibility of Scriptlet tag and separates the business logic from the JSP page
- Usually distributed in the form of a Tag Library.

Empty Tags <mg:printDate/>

Tag with Attributes <mg:printDate pattern="dd/MM/yyyy"/> </mg:printDate>

Tag with Body <mg:printDate >
Body content here
</mg:printDate>

Tag Library Descriptor

- TLD file:
  - An XML document that describes a tag library
  - Contains information about a library as a whole and about each tag contained in the library
  - Used by a Web container to validate the tags
  - The <taglib> is the root tag in the TLD file

Elements of TLD

Elements	Description
taglib	The tag library
tlib-version	The tag library's version
short-name	Optional name that could be used by a JSP page authoring tool to create names with a mnemonic value
uri	A URI that uniquely identifies the tag library
description	Optional tag-specific information

```
<taglib>
  <tlib-version>1.0</tlib-version>
  <uri>/myproject/tags</uri>
<tag>
      <name>oTime</name>
       <tag-class>com.tags.OfficeTimingTag</tag-class>
       <body-content>JSP</body-content>
    </tag>
</taglib>
```

Location
WEB-INF/myproject.tld

Inside META-INF folder of a jar file in lib folder

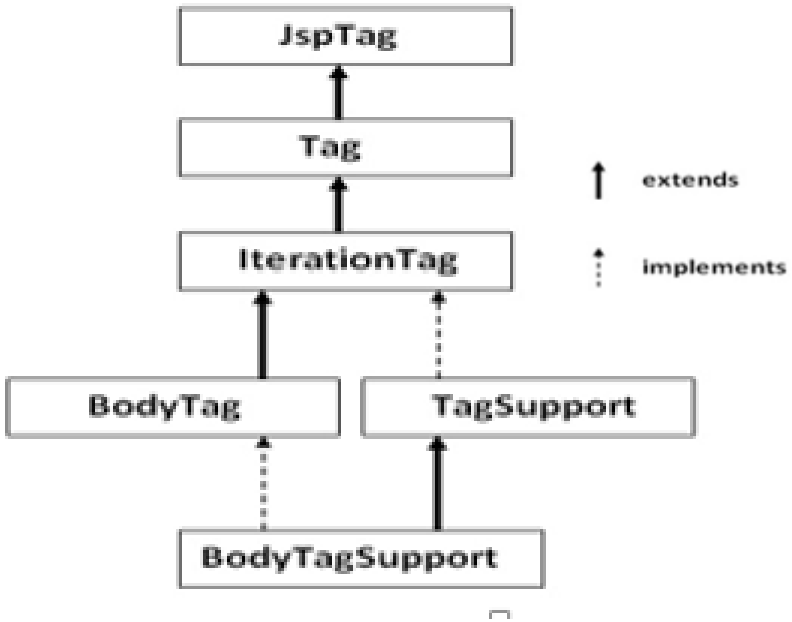
Creating Custom tags

- Create the Tag Library Descriptor File
- Create Tag Handlers (classes which will be executed on using the Custom Tag)
- Package Tag Handlers and TLD file into Tag Library (either unpacked or packed form)

- Using Custom tags in JSP
  - Provide the taglib directive in the JSP with the uri of TLD
  - Use the tag in the jsp

Tag Handler Classes

- Every tag has an associated tag handler class which provides the functionality of the tag
- Implements one of the interface or extend classes provided in javax.servlet.jsp.Tagext package



Tag interface:

It defines the methods that are called by the JSP implementation class during the lifecycle of the Tag.

IterationTag

Allows re-evaluation of the body content of the Custom Tag

BodyTag

Extends the IterationTag interface and defines methods that enable a Tag Handler to manipulate the body content of a Custom Tag.

Methods implemented in Tag Handler

- doStartTag
  - Declared in the Tag interface
  - Executed when the start Tag of the Custom Tag is encountered
  - Returns SKIP_BODY or EVAL_BODY_INCLUDE

- doEndTag
  - Declared in the Tag interface
  - Executed when the end Tag of the Custom Tag is encountered
  - Returns EVAL_PAGE or SKIP_PAGE

TagHandler

```
public class OfficeTimingTag extends TagSupport{
    @Override
    public int doStartTag() throws JspException {
        Calendar c1 = Calendar.getInstance();
        int hour = c1.get(Calendar.HOUR_OF_DAY);
        if (hour >= 9 && hour < 15){
            return EVAL_BODY_INCLUDE;
        }else{
            return SKIP_BODY;
        }
    }
```

JSP

```
<%@ taglib prefix='mg' uri="/project/tags" %>
<!DOCTYPE html >
<html>
<body>
    Current Date : <mg:fmtDate/>
    <br/><br/>
    <mg:oTime>Contact us at 1800-999-9999</mg:oTime>
```

Integrating Web Application with Database

- Configuring resource in context.xml

```
<WatchedResource>WEB-INF/web.xml</WatchedResource>
<Resource name="jdbc/or1DB" auth="Container"
        maxActive="25" maxIdle="10"
        username="HR" password="HR"
        driverClassName="oracle.jdbc.OracleDriver"
        url="jdbc:oracle:thin:@localhost:1521/XE"
        type="javax.sql.DataSource" />
```

- Adding Resource reference in web.xml

```
<resource-ref>
   <res-ref-name>jdbc/or1DB</res-ref-name>
   <res-type>javax.sql.DataSource</res-type>
   <res-auth>Container</res-auth>
</resource-ref>
```

- Getting Datasource in Servlets

```
Context ctx = new  InitialContext();
DataSource ds = DataSource)ctx.lookup("java:/comp/env/jdbc/or1DB");
Connection conn = ds.getConnection();
```

SUMMARY

Java Server Pages - JSP

SUMMARY

In this lesson, you've learned to:

- Introduction
- Elements of JSP
- Implicit Objects
- Using Java Beans in JSP
- Custom Tags