

JAVA SERVER PAGES



Introduction

- ❑ Java Server Page (JSP) is a template for a Web page that uses Java code to generate an HTML document dynamically.
- ❑ JSPs are run in a server-side component known as a JSP container, which translates them into equivalent Java servlets.
- ❑ It can be thought of as an extension to Servlet because it provides more functionality than servlet.

Introduction

- ❑ The JSP 1.2 specification is an important part of the Java 2 Platform, Enterprise Edition.
- ❑ Using JSP and Enterprise JavaBeans technologies together is a great way to implement distributed enterprise applications with web-based front ends.
- ❑ A JSP page consists of HTML tags and JSP tags.
- ❑ The JSP pages are easier to maintain than Servlet because we can separate designing and development

Advantages of JSPs

- Have better performance and scalability because they are persistent in memory and multithreaded.
- No special client setup is required.
- Have built-in support for HTTP sessions.
- Have full access to Java technology.
- Automatically recompiled when necessary.
- JSP enables the separation of static contents from dynamic contents.
- Greater compatibility with Web development tools.

Example

A simple JSP: HelloWorld.jsp

```
<HTML>
    <BODY>
        Hello World.
    </BODY>
</HTML>
```

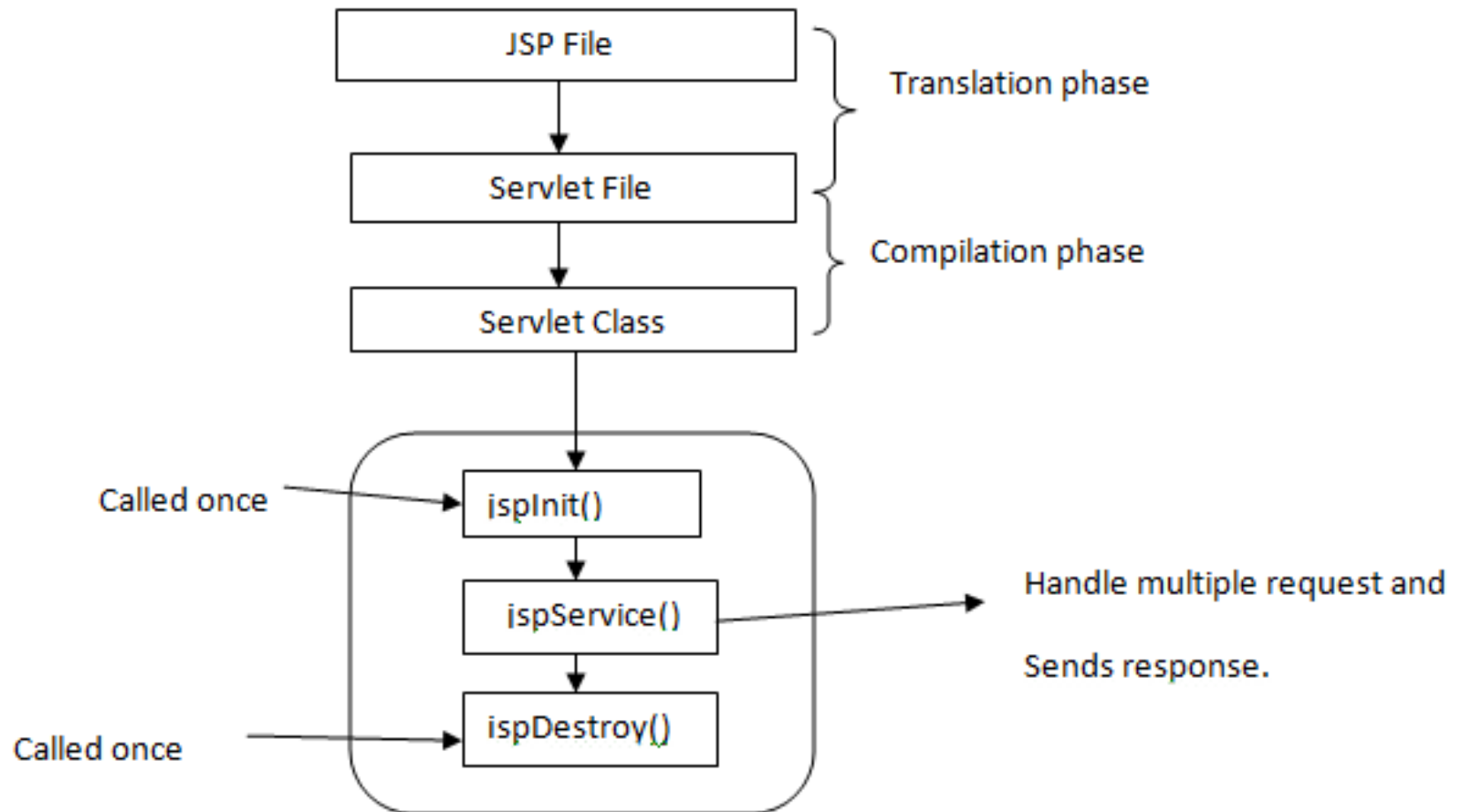
A dynamic JSP: HelloWorld2.jsp

```
<HTML>
    <BODY>
        Hello World. The local server time is
        <%= new java.util.Date() %>.
    </BODY>
</HTML>
```

Lifecycle of a JSP

- Translation of JSP Page
- Compilation of JSP Page
- Loading of Class file
- Instantiation
- Initialization
- Processing the Request
- Destroy

Lifecycle of a JSP



Lifecycle of a JSP

□ **Translation of JSP Page**

- ▣ This translation phase deals with Syntactic correctness of JSP.
- ▣ Here test.jsp file is translated to test.java.

□ **Compilation of JSP Page**

- ▣ Here the generated java servlet file (test.java) is compiled to a class file (test.class).

□ **Loading of Class file**

- ▣ Servlet class which has been loaded from JSP source is now loaded into container.

Lifecycle of a JSP

□ **Instantiation**

- ▣ Here instance of the class is generated.
- ▣ The container manages one or more instance by providing response to requests.

□ **Initialization**

- ▣ `jspInit()` method is called only once during the life cycle immediately after the generation of Servlet instance from JSP.

Lifecycle of a JSP

□ Processing the Request

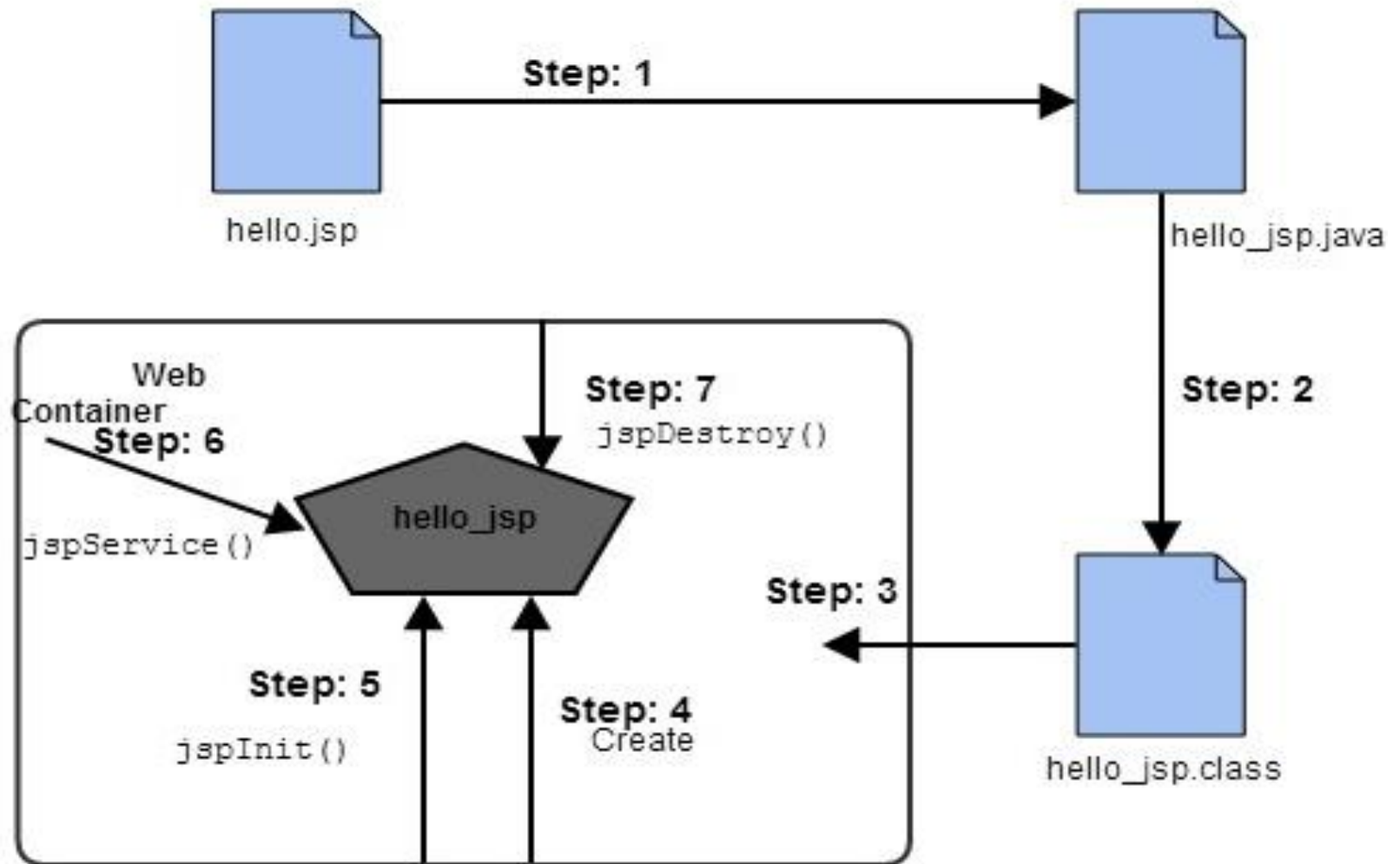
- ▣ `_jspService()` method is used to serve the raised requests by JSP.
- ▣ It takes request and response object as parameters.
- ▣ This method cannot be overridden.

□ Destroy

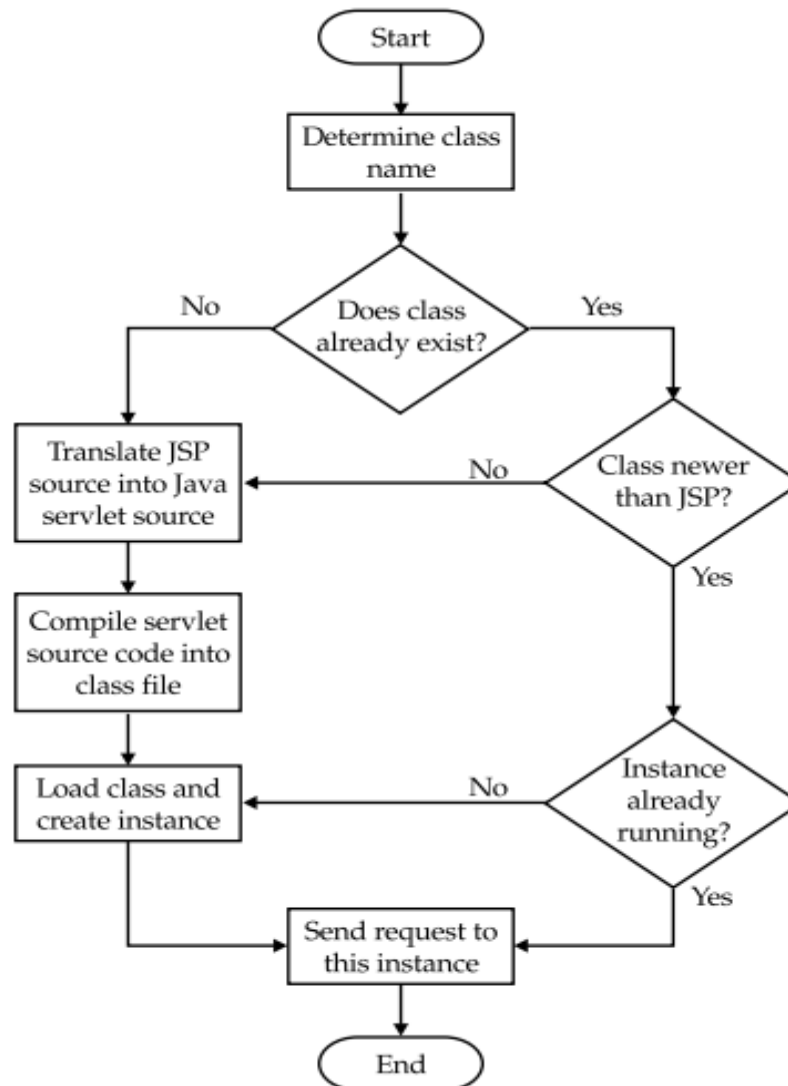
- ▣ To remove the JSP from use by the container.
- ▣ Called only once.
- ▣ If you need to perform any cleanup task like closing open files, releasing database connections `jspDestroy()` can be overridden.

Lifecycle of a JSP

Web Server



JSP Container



JSP Scripting Element

- JSP Scripting element are written inside `<% %>` tags and are processed by the JSP engine during translation of the JSP page.
- Any other text in the JSP page is considered as HTML code or plain text.

Scripting Element	Example
Comment	<code><%-- comment --%></code>
Scriptlet	<code><% scriptlets %></code>
Declaration	<code><%! declarations %></code>
Directive	<code><% @ directive%></code>
Expression	<code><%= expression %></code>

JSP Scripting Element

□ JSP Comment

▣ **Syntax:** `<%-- JSP comment --%>`

□ Example

```
<html>
```

```
<body>
```

```
    <%-- Code to print Welcome Message--%>
```

```
    Welcome
```

```
    <% out.println(request.getParameter("name"));%>
```

```
</body>
```

```
</html>
```

JSP Scripting Element

□ Scriptlet Tag:

- Scriptlet Tag allows you to write java code inside JSP page.
- Scriptlet tag implements the `_jspService` method functionality by writing script/java code.
- **Syntax:** `<% Java Code %>`

□ Example

```
<html>  
    <% int count = 0; %>  
    <body>  
        Page Count is <% out.println(++count); %>  
    </body>  
</html>
```

JSP Scripting Element

□ Declaration Tag

- ▣ Declare a variable or method in JSP inside Declaration Tag.
- ▣ You can declare static member, instance variable and methods inside Declaration Tag.

□ Syntax: `<%! declaration %>`

□ Example

```
<html>
```

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

```
    <body>
```

```
        Page Count is: <% out.println(++count); %>
```

```
    </body>
```

```
</html>
```


JSP Scripting Element

□ Directive Tag

- ▣ Gives special instruction to Web Container at the time of page translation.

- ▣ Directive tags are of three types:

1. **<%@ page ... %>**: Defines page dependent properties such as language, session, errorPage etc.
2. **<%@ include ... %>**: Defines file to be included.
3. **<%@ taglib ... %>**: Declares tag library used in the page.

JSP Scripting Element

□ Expression Tag

- ▣ Used to print out java language expression that is put between the tags.
- ▣ An expression tag can hold any java language expression that can be used as an argument to the out.print() method.

□ Syntax: `<%= JavaExpression %>`

- ▣ `<%= (2*5) %>` → `out.print((2*5));`

□ Example

```
<html>
```

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

```
    <body>
```

```
        Page Count is <%= ++count %>
```

```
    </body>
```

```
</html>
```

Difference between Scriptlet and Declaration

□ Declaration Tag

- ▣ Used for declaring variables and methods.
- ▣ During translation and compilation phase of JSP life cycle all variables declared in jsp declaration become instance variables of servlet class.
- ▣ All the methods become instance methods.
- ▣ Since instance variables are automatically initialized all variables declared in jsp declaration section gets their default values.

Difference between Scriptlet and Declaration

□ Scriptlet Tag

- ▣ Used for embedding java code fragments in JSP page.
- ▣ During translation phase of JSP Life cycle all scriptlet become part of `_jspService()` method.
- ▣ So we cannot declare methods in scriptlet since we cannot have methods inside other methods.
- ▣ As the variables declared inside scriptlet will get translated to local variables they must be initialized before use.

Greeting Example

```
<html>
  <body>
    <H1><center>Greeting Example</center></H1>
    <%
      Date x = new java.util.Date();
      if (x.getHours() > 5 && x.getHours() < 12 ) { %>
        Good Morning
      <% }
      else if (x.getHours() > 12 && x.getHours() < 16 ) { %>
        Good Afternoon
      <% }
      else if (x.getHours() > 16 || x.getHours() < 19 ) { %>
        Good Evening
      <% }
      else { %>
        Good Night
      <% } %>
    </body>
  </html>
```

Calculator Example

```
<html>
  <title>Sample Example </title>
  <body>
    <h1><center> Example of JSP </center></h1>
    <b> Mathematics</b>
    <hr>
    <form method="post" action="calculate.jsp">
      Enter first Value
      <input type="text" name="t1" value=""><br>

      Enter second Value
      <input type="text" name="t2" value=""><br>

      Enter the Operation to Perform (+, -, *, /)
      <input type="text" name="op" value=""><br>
      <br><br>
      <input type="submit" name="result">
    </form>
  </body>
</html>
```

Calculator Example

```
<%@ page language="java"%>
<%@ page import="java.lang.*"%>
<html>
    <body>
        <H1><center>Calculator Demo</center></H1>
        <%
            int i=Integer.parseInt(request.getParameter("t1"));
            int j=Integer.parseInt(request.getParameter("t2"));
            int res=0;

            String str=request.getParameter("op");
            if(str.equals("+"))
                res=i+j;
            if(str.equals("-"))
                res=i-j;
            if(str.equals("*"))
                res=i*j;
            if(str.equals("/"))
                res=i/j;
        %>
        Result is <%=res%>
    </body>
</html>
```

JSP Implicit Objects

- JSP supports 9 implicit objects.
- Created by the web container.

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

JSP out implicit object

- Used for writing any data to the buffer.
- It is the object of JspWriter.

- **Example**

```
<html>
```

```
  <body>
```

```
    <% out.print("Today is: "+  
      java.util.Calendar.getInstance().getTime()); %>
```

```
  </body>
```

```
</html>
```

JSP request implicit object

- Implicit object of type `HttpServletRequest`.
- Used to get request information such as parameter, header information, remote address, server name etc.
- It can also be used to set, get and remove attributes from the jsp request scope.

- **Example**

- **index.html**

```
<form action="welcome.jsp">  
    <input type="text" name="uname">  
    <input type="submit" value="go">  
</form>
```

JSP request implicit object

□ welcome.jsp

```
<html>
```

```
  <body>
```

```
    <%
```

```
      String name = request.getParameter("uname");
```

```
      out.print("welcome "+name);
```

```
    %>
```

```
  </body>
```

```
</html>
```

JSP response implicit object

- Implicit object of type HttpServletResponse.
- Used to add or manipulate response such as redirect response to another resource, send error etc.

- **Example**

- **index.html**

```
<form action="welcome.jsp">
```

```
    <input type="text" name="search">
```

```
    <input type="submit" value="Google Search"><br/>
```

```
</form>
```

JSP response implicit object

□ welcome.jsp

```
<html>
```

```
  <body>
```

```
    <%
```

```
      String name = request.getParameter("search");
```

```
      response.sendRedirect
```

```
      ("http://www.google.co.in/#q="+name);
```

```
    %>
```

```
  </body>
```

```
</html>
```

JSP config implicit object

- Implicit object of type ServletConfig.
- Used to get initialization parameter for a particular JSP page.
- Generally, it is used to get initialization parameter from the web.xml file.

- **Example**

- **index.html**

```
<form action="welcome">
```

```
    <input type="text" name="uname">
```

```
    <input type="submit" value="go"><br/>
```

```
</form>
```

JSP config implicit object

□ welcome.jsp

<%

```
out.print("Welcome "+request.getParameter("uname"));
```

```
String driver=config.getInitParameter("dname");
```

```
out.print("driver name is="+driver);
```

%>

□ web.xml file

<init-param>

```
<param-name>dname</param-name>
```

```
<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
```

</init-param>

JSP application implicit object

- Implicit object of type ServletContext.
- The instance is created only once by the web container when application or project is deployed on the server.
- Used to get initialization parameter from web.xml.
- Used to get, set or remove attribute from application scope.
- This initialization parameter can be used by all jsp pages.

- **Example**

- **index.html**

```
<form action="welcome">
```

```
    <input type="text" name="uname">
```

```
    <input type="submit" value="go"><br/>
```

```
</form>
```


JSP application implicit object

□ **welcome.jsp**

<%

```
out.print("Welcome "+request.getParameter("uname"));
```

```
String driver=application.getInitParameter("dname");
```

```
out.print("driver name is="+driver);
```

%>

□ **web.xml file**

<web-app>

<context-param>

```
<param-name>dname</param-name>
```

```
<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
```

</context-param>

</web-app>

config object vs application object

- config is an implicit object of type ServletConfig.
- application is an implicit object of type ServletContext.
- config can be used to get initialization parameter for a particular JSP page.
- application initialization parameter can be used by all jsp pages.
- Application is created only once by the web container when application or project is deployed on the server
- config is created by the web container for each jsp page.

JSP session implicit object

- Implicit object of type HttpSession.
- Used to set, get or remove attribute or to get session information.

- **Example**

- **index.html**

```
<form action="welcome.jsp">  
    <input type="text" name="uname">  
    <input type="submit" value="go"><br/>  
</form>
```

JSP session implicit object

□ welcome.jsp

<%

```
String name=request.getParameter("uname");
```

```
out.print("Welcome "+name);
```

```
session.setAttribute("user",name);
```

```
<a href="second.jsp">second jsp page</a>
```

%>

□ second.jsp

<%

```
String name=(String)session.getAttribute("user");
```

```
out.print("Hello "+name);
```

%>

JSP pageContext implicit object

- ❑ Implicit object of type PageContext class.
- ❑ Used to set, get or remove attribute from page, request, session and application scope.
- ❑ In JSP, page scope is the default scope.
- ❑ **Example**
- ❑ **index.html**

```
<form action="welcome.jsp">  
    <input type="text" name="uname">  
    <input type="submit" value="go"><br/>  
</form>
```

JSP pageContext implicit object

□ welcome.jsp

```
<%  
String name=request.getParameter("uname");  
out.print("Welcome "+name);  
pageContext.setAttribute("user",name,PageContext.SESSION_SCOPE);  
<a href="second.jsp">second jsp page</a>  
%>
```

□ second.jsp

```
<%  
String name=(String)pageContext.getAttribute  
("user",PageContext.SESSION_SCOPE);  
out.print("Hello "+name);  
%>
```

JSP page implicit object

- Implicit object of type Object class.
- This object is assigned to the reference of auto generated servlet class.
- It is written as:
 - ▣ **Object page=this;**
- For using this object it must be cast to Servlet type.
- **Example**

```
<% (HttpServletRequest)page.log("message"); %>
```

- Since, it is of type Object it is less used because you can use this object directly in jsp.
- **<% this.log("message"); %>**

JSP exception implicit object

- ❑ Implicit object of type `java.lang.Throwable` class.
- ❑ Used to print the exception.
- ❑ But it can only be used in error pages.

- ❑ **Example**

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

```
<html>
```

```
<body>
```

```
Sorry following exception occurred:<%= exception %>
```

```
</body>
```

```
</html>
```


Session Handling in JSP

- HTTP is a "stateless" protocol.
- Each time when a client retrieves a web page, the client opens a separate connection to the web server.
- The server does not keep any record of previous client request.
- Four ways to maintain the session between client and a server
 - ▣ Cookies
 - ▣ Hidden fields
 - ▣ URL Rewriting
 - ▣ HttpSession

Session Handling in JSP

□ Cookies

- A cookie is a small piece of information created by a JSP program.
- A Cookie is stored in the client's hard disk by the browser.
- Cookies are used to store various kind of information such as username, password, and user preferences, etc.

Session Handling in JSP

□ Sample JSP program to create a Cookie

```
<html>
    <body>
        <%!
            String name = "userID";
            String value = "ABCD";
            Cookie myCookie = new Cookie (name, value);
        %>
        <%
            response.addCookie(myCookie);
        %>
    </body>
</html>
```

Session Handling in JSP

□ Sample JSP program to read the Cookie

```
<html>
```

```
    <body>
```

```
        <%!
```

```
            String cName = "userID";
```

```
            String name;
```

```
            String value;
```

```
            int found = 0;
```

```
        %>
```

```
    <%
```

```
        Cookie[] myCookies = request.getCookies();
```

Session Handling in JSP

□ Sample JSP program to read the Cookie

```
for (int i=0;i<myCookies.length;i++) {  
    name = myCookies[i].getName();  
    if (name.equals(cName)) {  
        value = myCookies[i].getValue();  
        found = 1;  
    }  
}  
  
if (found==1) { %>  
    <p> Cookie Name: <%= name %> </p>  
    <p> Cookie Value: <%= value %> </p>  
  
<% }  
else { %>  
    <p> Cookie NOT FOUND </p>  
  
<%>} %>  
</body>  
</html>
```

Session Handling in JSP

□ Hidden Field

- A hidden field is a field in HTML form whose value isn't displayed on the HTML page.
- Assign a value to the hidden field in a JSP program and send the HTML page to the browser.
- Consider a login screen which asks for username and password.
- Here is how the session is tracked using hidden field:
 - ▣ Upon submitting the form, the browser sends the username and password to the JSP program.

Session Handling in JSP

- The JSP program then validates the username and password and generates another dynamic HTML page.
- This newly generated HTML page has a form that contains hidden field which is assigned a userID along with other fields as well.
- When the user submits the form in the new HTML page, the userID stored in the hidden field and other information on the form are sent to the JSP program.
- This cycle continues. Each HTML page and subsequent JSP program has access to userID and therefore can track the session.

Session Handling in JSP

- ❑ **Session Object**
- ❑ To share information among JSP programs within a session by using a session object.
- ❑ Each time a session is created, a unique ID is assigned to the session and stored as a cookie.
- ❑ The unique ID enables JSP programs to track multiple sessions simultaneously.
- ❑ In addition to session ID, a session object is also used to store other types of information called attributes.

Session Handling in JSP

□ Sample JSP program to create a session attribute

```
<html>
```

```
    <body>
```

```
        <%!
```

```
            String name = "userID";
```

```
            String value = "ABCD";
```

```
        %>
```

```
    <%
```

```
        session.setAttribute(name, value);
```

```
    %>
```

```
    </body>
```

```
</html>
```

Session Handling in JSP

□ Sample JSP program to read session attribute

```
<html>
    <body>
        <%! Enumeration<String> e; %>
        <%
e = session.getAttributeNames();
while (e.hasMoreElements()) {
    String name = (String) e.nextElement();
    String value = (String) session.getAttribute(name);
    %>
    <p> Attribute name : <%= name %></p>
    <p> Attribute value : <%= value %></p>
    <% } %>
        </body>
    </html>
```

JSP Action Tags

- ❑ JSP Action Tags are used to perform some specific tasks.
- ❑ The Action Tags are used to control the flow between pages.
- ❑ The Action Tags are used to work with Java Bean.
- ❑ The JSPaction tags are:
 - ❑ **jsp:forward:** Forwards the request and response to another resource.
 - ❑ **jsp:include:** Includes another resource.
 - ❑ **jsp:useBean:** Creates or locates bean object.

JSP Action Tags

- The JSPaction tags are:
 - ▣ **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

- 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="pname" value="pvalue | <%=expression%>"
 />
</jsp:forward>`

jsp:forward

- Example to forward the request to the printdate.jsp file with parameter.
- **index.jsp**

```
<html>
    <body>
        <h2>this is index page</h2>
        <jsp:forward page="printdate.jsp" >
            <jsp:param name="name" value="ABCD" />
        </jsp:forward>
    </body>
</html>
```

jsp:forward

- The printdate.jsp file prints the parameter value with date and time.
- **printdate.jsp**

```
<html>
    <body>
        <%= request.getParameter("name") %>
        <% out.print("Today is:" +
java.util.Calendar.getInstance().getTime()); %>
    </body>
</html>
```

jsp:include

- 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.

- **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="pname" value="pvalue" |
<%=expression%>" />`

`</jsp:include>`

jsp include directive vs 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.

jsp:include

- In this example, index.jsp file includes the content of the printdate.jsp file.
- **index.jsp**

```
<html>
```

```
    <body>
```

```
        <h2>this is index page</h2>
```

```
        <jsp:include page="printdate.jsp" />
```

```
        <h2>end section of index page</h2>
```

```
    </body>
```

```
</html>
```

jsp:include

□ printdate.jsp

```
<html>
```

```
<body>
```

```
<% out.print("Today is:" + java.util.Calendar.getInstance().getTime()); %>
```

```
</body>
```

```
</html>
```



this is index page

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

end section of index page

JSP Expression Language (EL)

- ❑ Introduced in JSP 2.0.
- ❑ The main purpose of it is to simplify the process of accessing data from bean properties and from implicit objects.
- ❑ EL includes arithmetic, relational and logical operators too.
- ❑ Syntax:
`${expression}`
- ❑ Whatever is present inside braces gets evaluated at runtime and being sent to the output stream.

JSP Expression Language (EL)

- Example 1: Expression language evaluates the expressions

```
<html>
```

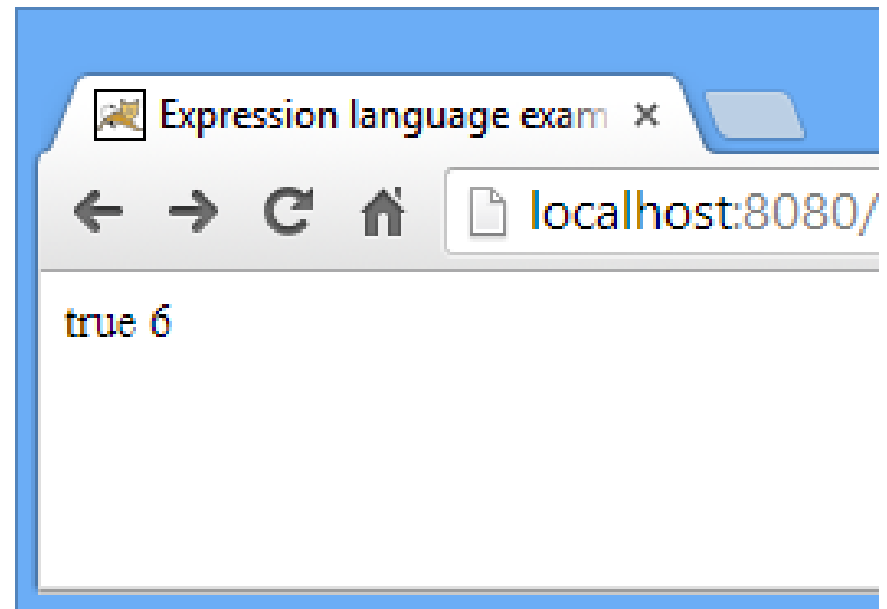
```
    <body>
```

```
        ${ 1<2 }
```

```
        ${ 1+2+3 }
```

```
    </body>
```

```
</html>
```



JSP Expression Language (EL)

- Example 2: Value fetch using param variable of expression language
- **index.jsp**

```
<html>
    <body>
        <form action="display.jsp">
            Student Name: <input type="text" name="name" /><br>
            Student Roll:<input type="text" name="rollno" /><br>
            <input type="submit" value="Submit Details!!"/>
        </form>
    </body>
</html>
```

JSP Expression Language (EL)

- Example 2: Value fetch using param variable of expression language

- **display.jsp**

```
<html>
```

```
    <body>
```

```
        Student name is ${ param.name } <br>
```

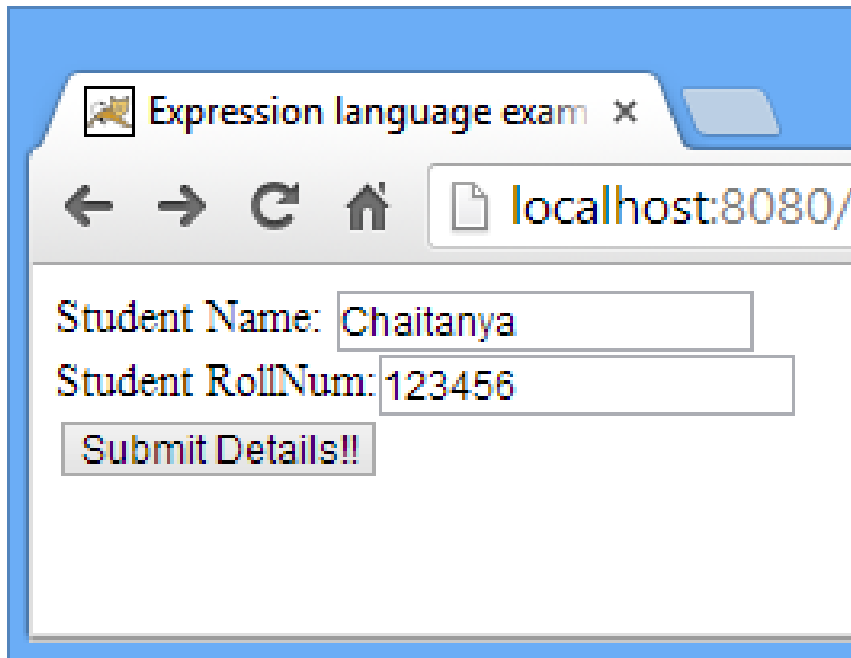
```
        Student Roll No is ${ param.rollno }
```

```
    </body>
```

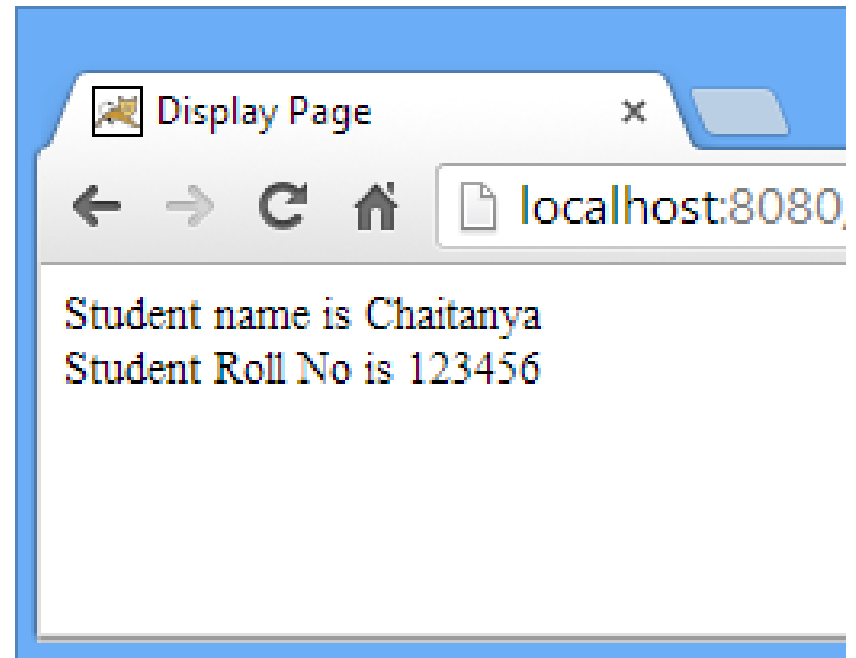
```
</html>
```

JSP Expression Language (EL)

- Example 2: Value fetch using param variable of expression language



A screenshot of a web browser window with the title 'Expression language exam'. The address bar shows 'localhost:8080/'. The form contains two input fields: 'Student Name:' with the value 'Chaitanya' and 'Student RollNum:' with the value '123456'. Below the fields is a button labeled 'Submit Details!!'.



A screenshot of a web browser window with the title 'Display Page'. The address bar shows 'localhost:8080/'. The page displays the submitted values: 'Student name is Chaitanya' and 'Student Roll No is 123456'.

JSP Expression Language (EL)

- EL predefined variables:
- Similar to implicit objects in JSP we have predefined variables in EL.
 - ▣ **pageScope:** It helps in getting the attribute stored in Page scope.
 - ▣ **pageContext:** Same as JSP PageContext object.
 - ▣ **sessionScope:** Fetches attributes from session scope, set by session object.
 - ▣ **requestScope:** It used for getting the attributes from request scope. The attribute which are set by request implicit object.

JSP Expression Language (EL)

- EL predefined variables:
 - ▣ **param:** Similar to `ServletRequest.getParameter`.
 - ▣ **applicationScope:** Used for getting Application level attributes.
 - ▣ **header:** It helps in getting HTTP request headers as Strings.
 - ▣ **headerValues:** Used for fetching all the HTTP request headers.
 - ▣ **initParam:** It links to context initialization parameters.
 - ▣ **paramValues:** Same as `ServletRequest.getParameterValues`.
 - ▣ **cookie:** It maps to Cookie object.

JSP Standard Tag Library (JSTL)

- ❑ The JSP Standard Tag Library (JSTL) represents a set of tags to simplify the JSP development.
- ❑ Advantage of JSTL
 1. Fast Development JSTL provides many tags that simplifies the JSP.
 2. Code Reusability We can use the JSTL tags in various pages.
 3. No need to use scriptlet tag It avoids the use of scriptlet tag.

JSP Standard Tag Library (JSTL)

- There JSTL mainly provides 5 types of tags:
- **Core tags:**
 - ▣ The JSTL core tag provide variable support, URL management, flow control etc.
 - ▣ url: <http://java.sun.com/jsp/jstl/core>.
 - ▣ The prefix of core tag is c.
- **Function tags:**
 - ▣ The functions tags provide support for string manipulation and string length.
 - ▣ url: <http://java.sun.com/jsp/jstl/functions>.
 - ▣ The prefix of function tag is fn.

JSP Standard Tag Library (JSTL)

□ **Formatting tags:**

- The Formatting tags provide support for message formatting, number and date formatting etc.
- url: <http://java.sun.com/jsp/jstl/fmt>
- The prefix of Formatting tag is fmt.

□ **XML tags:**

- The xml sql tags provide flow control, transformation etc.
- url: <http://java.sun.com/jsp/jstl/xml>
- The prefix of XML tag is x.

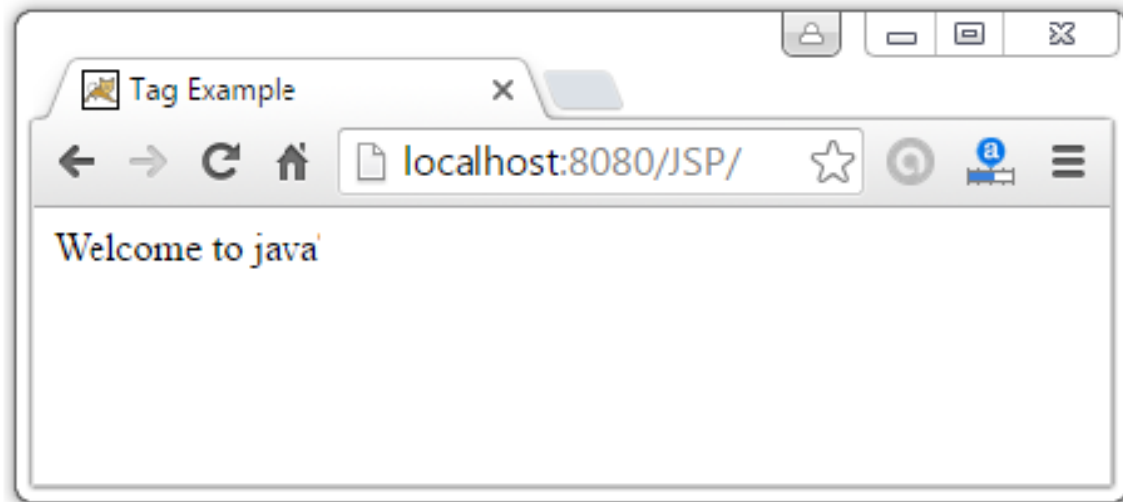
□ **SQL tags:**

- The JSTL sql tags provide SQL support.
- url: <http://java.sun.com/jsp/jstl/sql>
- The prefix of SQL tag is sql.

JSTL Core Tags

□ Example:

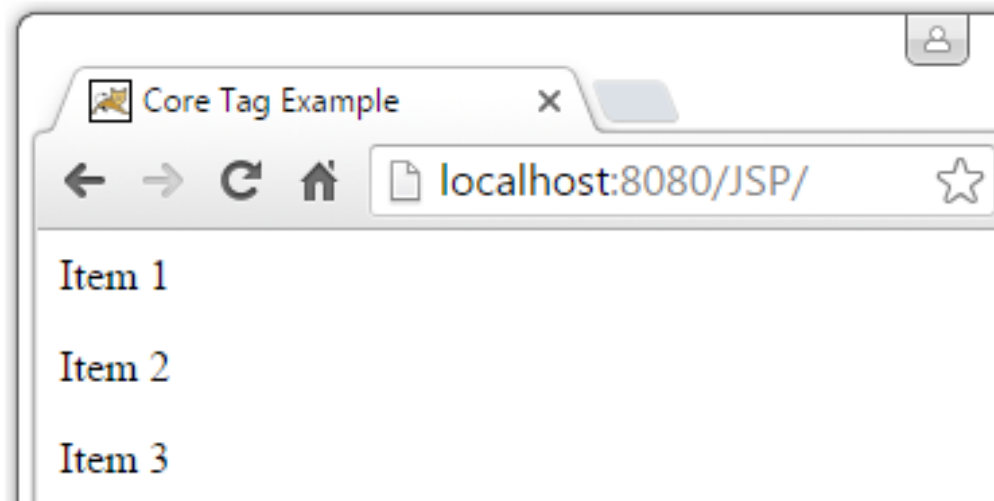
```
<% @ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
    <body>
        <c:out value="${ 'Welcome to java' }"/>
    </body>
</html>
```



JSTL Core Tags

□ Example:

```
<% @ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<html>
    <body>
        <c:forEach var="j" begin="1" end="3">
            Item <c:out value="${j}"/><p>
        </c:forEach>
    </body>
</html>
```



JSTL Core Tags

Tags	Description
<u>c:out</u>	It display the result of an expression, similar to the way <code><%=...%></code> tag work.
<u>c:import</u>	It Retrives relative or an absolute URL and display the contents to either a String in 'var',a Reader in 'varReader' or the page.
<u>c:set</u>	It sets the result of an expression under evaluation in a 'scope' variable.
<u>c:remove</u>	It is used for removing the specified scoped variable from a particular scope.
<u>c:catch</u>	It is used for Catches any Throwable exceptions that occurs in the body.
<u>c:if</u>	It is conditional tag used for testing the condition and display the body content only if the expression evaluates is true.
<u>c:choose, c:when, c:otherwise</u>	It is the simple conditional tag that includes its body content if the evaluated condition is true.

JSTL Core Tags

Tags	Description
<u>c:forEach</u>	It is the basic iteration tag. It repeats the nested body content for fixed number of times or over collection.
<u>c:forEachTokens</u>	It iterates over tokens which is separated by the supplied delimiters.
<u>c:param</u>	It adds a parameter in a containing 'import' tag's URL.
<u>c:redirect</u>	It redirects the browser to a new URL and supports the context-relative URLs.
<u>c:url</u>	It creates a URL with optional query parameters.

JSTL Function Tags

- ❑ The `fn:contains()` is used for testing if the string containing the specified substring.
- ❑ If the specified substring is found in the string, it returns true otherwise false.
- ❑ Syntax used for including the `fn:contains()` function is:
- ❑ `boolean contains(java.lang.String, java.lang.String)`

JSTL Function Tags

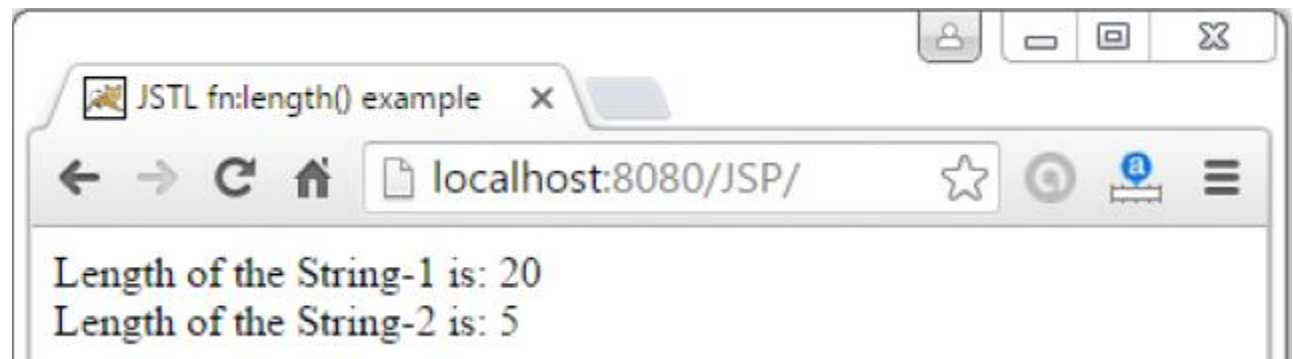
□ Example

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
<html>
    <body>
        <c:set var="String" value="Welcome to javalab"/>
        <c:if test="${fn:contains(String, 'javalab')}">
            <p>Found javalab string<p>
        </c:if>
        <c:if test="${fn:contains(String, 'JAVALAB')}">
            <p>Found JAVALAB string<p>
        </c:if>
    </body>
</html>
```

JSTL Function Tags

□ Example

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
<html>
    <body>
        <c:set var="str1" value="This is first string"/>
        <c:set var="str2" value="Hello"/>
        Length of the String-1 is: ${fn:length(str1)}
        <br>
        Length of the String-2 is: ${fn:length(str2)}
    </body>
</html>
```



JSTL Function Tags

JSTL Functions	Description
<u>fn:contains()</u>	It is used to test if an input string containing the specified substring in a program.
<u>fn:containsIgnoreCase()</u>	It is used to test if an input string contains the specified substring as a case insensitive way.
<u>fn:endsWith()</u>	It is used to test if an input string ends with the specified suffix.
<u>fn:escapeXml()</u>	It escapes the characters that would be interpreted as XML markup.
<u>fn:indexOf()</u>	It returns an index within a string of first occurrence of a specified substring.
<u>fn:trim()</u>	It removes the blank spaces from both the ends of a string.
<u>fn:startsWith()</u>	It is used for checking whether the given string is started with a particular string value.

JSTL Function Tags

JSTL Functions	Description
<u>fn:split()</u>	It splits the string into an array of substrings.
<u>fn:toLowerCase()</u>	It converts all the characters of a string to lower case.
<u>fn:toUpperCase()</u>	It converts all the characters of a string to upper case.
<u>fn:substring()</u>	It returns the subset of a string according to the given start and end position.
<u>fn:substringAfter()</u>	It returns the subset of string after a specific substring.
<u>fn:substringBefore()</u>	It returns the subset of string before a specific substring.
<u>fn:length()</u>	It returns the number of characters inside a string, or the number of items in a collection.
<u>fn:replace()</u>	It replaces all the occurrence of a string with another string sequence.

JSTL Formatting tags

Formatting Tags	Descriptions
<u>fmt:parseNumber</u>	It is used to Parses the string representation of a currency, percentage or number.
<u>fmt:timeZone</u>	It specifies a parsing action nested in its body or the time zone for any time formatting.
<u>fmt:formatNumber</u>	It is used to format the numerical value with specific format or precision.
<u>fmt:parseDate</u>	It parses the string representation of a time and date.
<u>fmt:bundle</u>	It is used for creating the ResourceBundle objects which will be used by their tag body.
<u>fmt:setTimeZone</u>	It stores the time zone inside a time zone configuration variable.
<u>fmt:setBundle</u>	It loads the resource bundle and stores it in a bundle configuration variable or the named scoped variable.
<u>fmt:message</u>	It display an internationalized message.
<u>fmt:formatDate</u>	It formats the time and/or date using the supplied pattern and styles.

JSTL XML tags

XML Tags	Descriptions
<u>x:out</u>	Similar to <%= ... > tag, but for XPath expressions.
<u>x:parse</u>	It is used for parse the XML data specified either in the tag body or an attribute.
<u>x:set</u>	It is used to sets a variable to the value of an XPath expression.
<u>x:choose</u>	It is a conditional tag that establish a context for mutually exclusive conditional operations.
<u>x:when</u>	It is a subtag of that will include its body if the condition evaluated be 'true'.
<u>x:otherwise</u>	It is subtag of that follows tags and runs only if all the prior conditions evaluated be 'false'.
<u>x:if</u>	It is used for evaluating the test XPath expression and if it is true, it will processes its body content.
<u>x:transform</u>	It is used in a XML document for providing the XSL(Extensible Stylesheet Language) transformation.
<u>x:param</u>	It is used along with the transform tag for setting the parameter in the XSLT style sheet.

JSTL SQL Tags

SQL Tags	Descriptions
<u>sql:setDataSource</u>	It is used for creating a simple data source suitable only for prototyping.
sql:query	It is used for executing the SQL query defined in its sql attribute or the body.
sql:update	It is used for executing the SQL update defined in its sql attribute or in the tag body.
sql:param	It is used for sets the parameter in an SQL statement to the specified value.
sql:dateParam	It is used for sets the parameter in an SQL statement to a specified java.util.Date value.
<u>sql:transaction</u>	It is used to provide the nested database action with a common connection.

THANK YOU

