UNIT-2 JSP

ADVANCE JAVA PROGRAMMING

TOPICS TO BE DISCUSSED

- Basics of JSP: Life Cycle Of JSP,
- JSP API
- Scripting Elements: Scriptlet Tag, Expression Tag, Declaration Tag;
- Implicit Objects: Out, Request, Response, Config, Application, Session, Page Context page, Exception;
- Directive Elements: Page Directive Include Directive, Taglib Directive;
- Action Elements

INTRODUCTION

- JSP is based on the Java technology and is an extension of the Java Servlet technology
- Platform independence and extensibility of servlets are easily incorporated in JSP
- Using the Java server-side modules, JSP can fit effortlessly into the framework of a Web server with minimal overhead
- The use of XML-like tags and Java-like syntax in JSP facilitate building Web-based applications

BENEFITS OF JSP

- **Platform independence.** The use of JSP adds versatility to a Web application by enabling its execution on any computer
- **Enhanced performance.** The compilation process in JSP produces faster results or output
- **Separation of logic from display.** The use of JSP permits the HTML-specific static content and a mixture of HTML, Java, and JSP to be placed in separate files
- **Ease of administration.** The use of JSP eliminates the need for high-level technical expertise
- **Ease of use.** All JSP applications run on major Web servers and operating systems

JSP VS OTHER SCRIPTING LANGUAGE

- **JSP versus ASP.** The dynamic content of JSP is written in Java, in contrast to that of ASP, which is written using an ASP-specific language
- **JSP versus PHP.** PHP is similar to ASP and JSP to a certain extent. With basic HTML knowledge, however, a VBScript programmer can write ASP applications and a Java programmer can create JSP applications, whereas PHP requires learning an entirely new language
- **JSP versus JavaScript.** JavaScript is a client-side scripting language used to build parts of HTML Web pages while the browser loads a document. As a result, the pages generated in JavaScript create dynamic content that is solely based on the client environment

JSP CHARACTERISTICS

- JSP is an extension of the Java Servlet technology
- It uses Java objects to receive Web requests and subsequently builds and sends back appropriate responses to the browser
- The compilation of an JSP page generates a servlet that incorporates all servlet functionality

ADVANTAGES OF JSP OVER SERVLET

- There are many advantages of JSP over the Servlet. They are as follows:
- 1) **Extension to Servlet**
- JSP technology is the extension to Servlet technology. We can use all the features of the Servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.
- 2) **Easy to maintain**
- JSP can be easily managed because we can easily separate our business logic with presentation logic. In Servlet technology, we mix our business logic with the presentation logic.

ADVANTAGES OF JSP OVER SERVLET

- 3) **Fast Development: No need to recompile and redeploy**
- If JSP page is modified, we don't need to recompile and redeploy the project. The Servlet code needs to be updated and recompiled if we have to change the look and feel of the application.
- 4) **Less code than Servlet**
- In JSP, we can use many tags such as action tags, JSTL, custom tags, etc. that reduces the code. Moreover, we can use EL, implicit objects, etc.

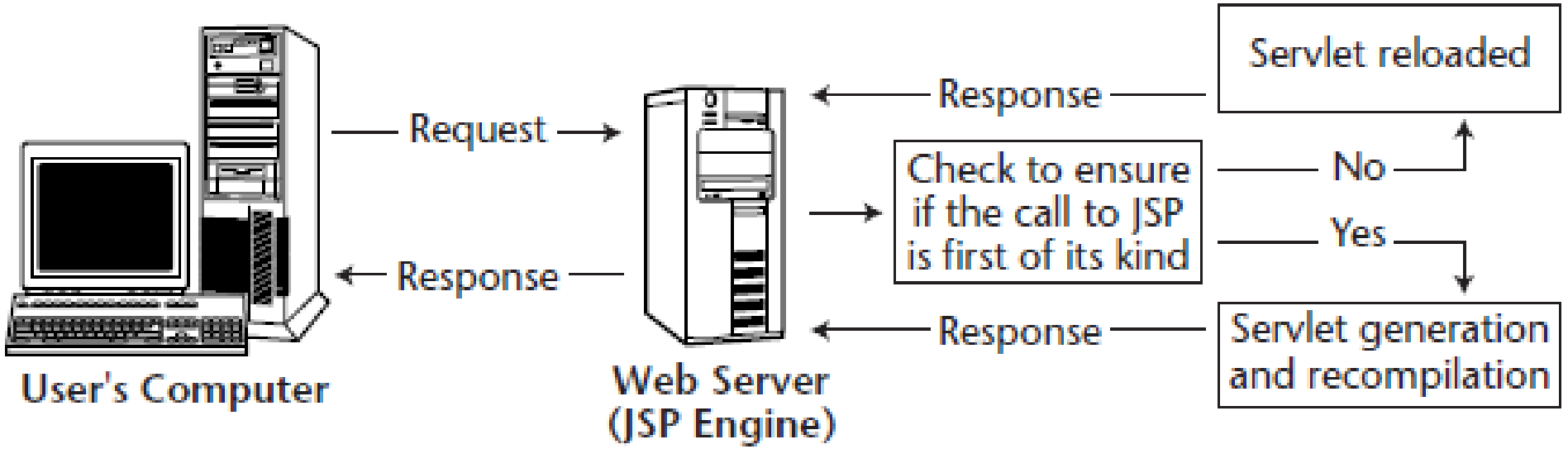
JSP AND SERVLETS

- A JSP file, when compiled, generates a servlet
- A JSP file is much easier to deploy because the JSP engine performs the recompilation for the Java code automatically
- JSP also aims to relieve the programmers from coding for servlets by auto-generation of servlets
- Servlets and JSP share common features, such as platform independence, creation of database-driven Web applications, and server-side programming capabilities

JSP AND SERVLETS

- Servlets tie up files to handle the static presentation logic and the dynamic business logic independently
- An HTML file is used for the static content and a Java file for the dynamic content
- A change made to any file requires the recompilation of the corresponding servlet
- JSP allows Java code to be embedded directly into an HTML page by using special tags
- The HTML content and the Java content can also be placed in separate files

REQUEST-RESPONSE CYCLE

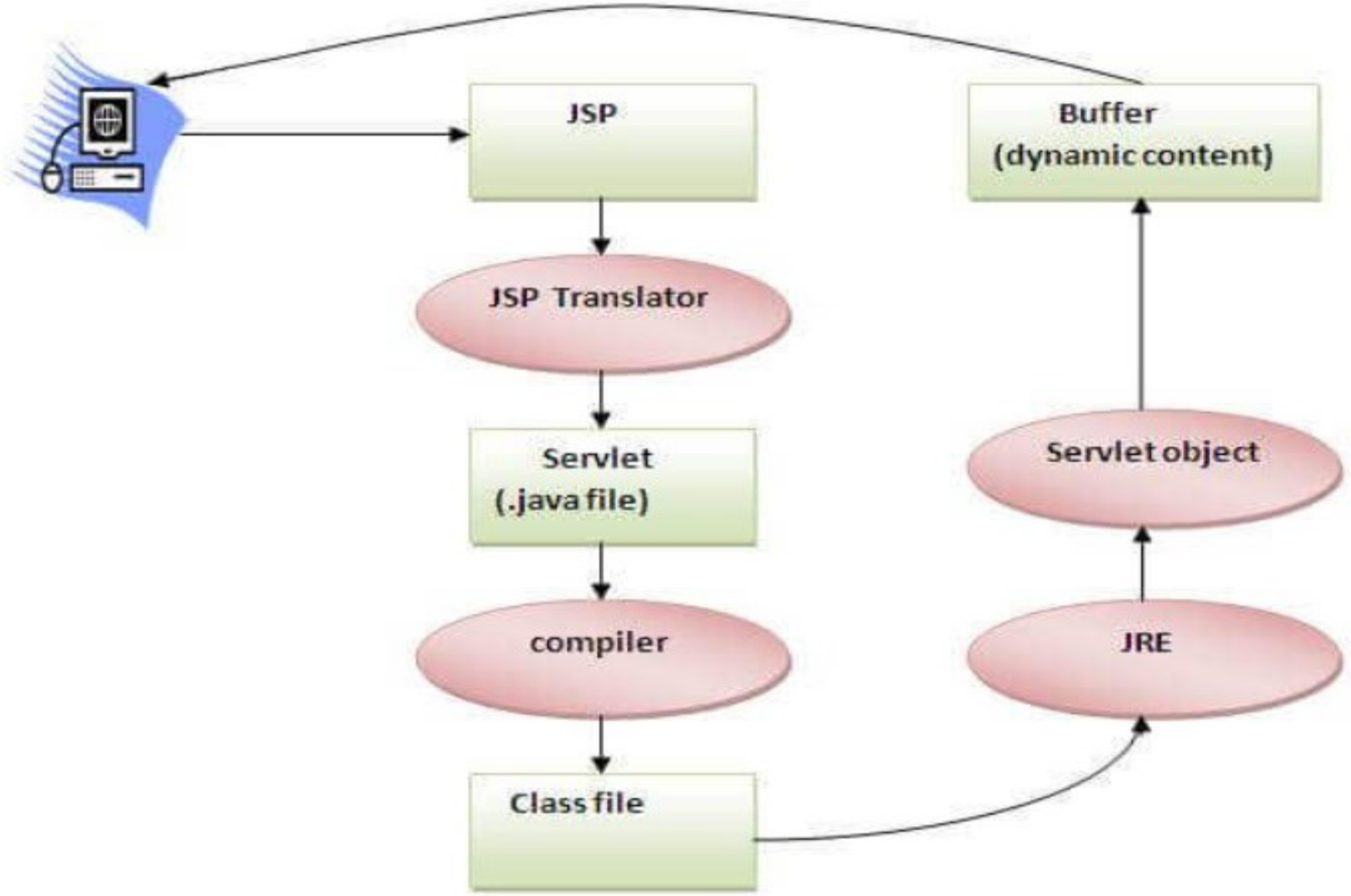


LIFECYCLE OF JSP PAGE

The JSP pages follow these phases:

- Translation of JSP Page
- Compilation of JSP Page
- Classloading (the classloader loads class file)
- Instantiation (Object of the Generated Servlet is created).
- Initialization (the container invokes jspInit() method).
- Request processing (the container invokes _jspService() method).
- Destroy (the container invokes jspDestroy() method).

Note: jspInit(), _jspService() and jspDestroy() are the life cycle methods of JSP.


JSP
Buffer
(dynamic content)
JSP Translator
Servlet
(.java file)
Servlet object
compiler
JRE
Class file

CREATING A SIMPLE JSP PAGE

- To create the first JSP page, write some HTML code as given below, and save it by .jsp extension.
- We have saved this file as index.jsp. Put it in a folder and paste the folder in the web-apps directory in Apache tomcat to run the JSP page.

```
<html>
<body>
<%out.println (2*5);%>
<body>
</html>
```

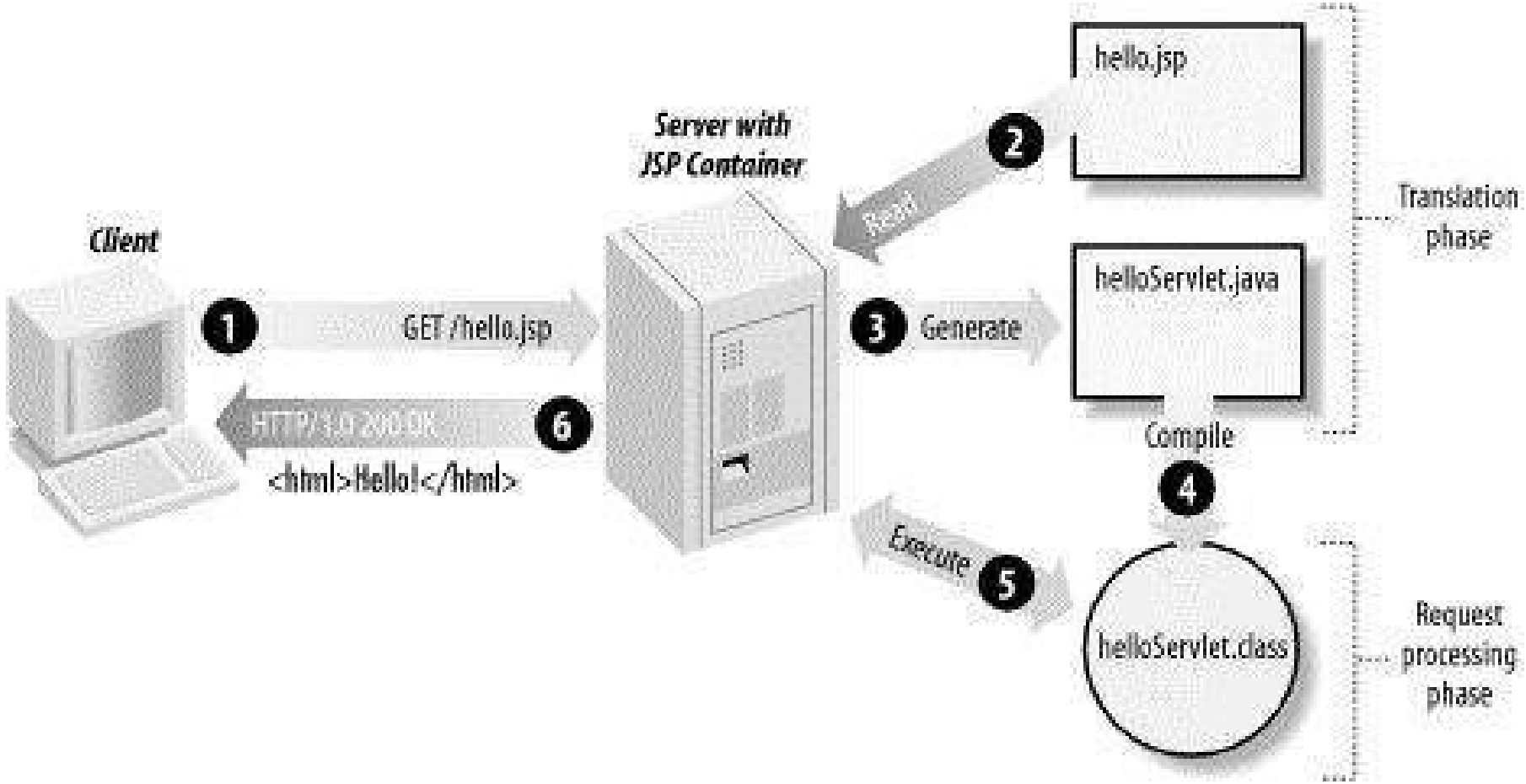
RUN A JSP PAGE

- Follow the following steps to execute this JSP page:
- Start the server
- Put the JSP file in a folder and deploy on the server
- Visit the browser by the URL http://localhost:portno/contextRoot/jspfile, for example, http://localhost:8888/myapplication/index.jsp

COMPILATION PROCESS

- The compilation of a JSP page builds a request and a response cycle with two phases:
  - the translation phase
  - the request-processing phase
- When the client requests a JSP page, the server sends a request to the JSP Engine
- The JSP container compiles the corresponding translation unit
- Then, the JSP engine automatically generates a servlet
- This results in the creation of a class file for the JSP page
- The response is generated according to the request specifications
- The servlet then sends back the response corresponding to the request received

COMPILATION PROCESS



INSTALLATION

- Download an implementation of the **Java Software Development Kit (SDK)**
- Set the **PATH and JAVA_HOME** environment variables to refer to the directory that contains **java** and **javac**, typically **java_install_dir/bin** and **java_install_dir**
- A number of Web Servers that support JavaServer Pages and Servlets development
- Apache Tomcat is an open source software implementation of the JavaServer Pages and Servlet technologies
- After a successful startup, the default web-applications included with Tomcat will be available by visiting **http://localhost:8080/**

CREATING A JSP PAGE

- A JSP page is very similar to an HTML document except for the addition of some tags containing Java code
- These tags, known as JSP tags, help to differentiate and segregate
- Following is the syntax of Scriptlet

<% code fragment %>

- You can write the XML equivalent of the above syntax as follows

<jsp:scriptlet> code fragment </jsp:scriptlet>

CREATING A JSP PAGE

```
<html>
        <head>
                <title>Hello World</title>
        </head>
<body>
Hello World!<br/>
<% out.println("Your IP address is " + request.getRemoteAddr()); %>
</body>
</html>
```

CONTENTS OF A JSP PAGE

- You can categorize the contents of a JSP page as follows:
  - HTML components consisting of HTML tags
  - JSP components consisting of JSP tags
- The JSP tags can also be broadly classified into the following groups
  - **Translation-time** tags
  - **Comments** that are used to provide information about code snippets
  - **Directives** that are used to convey overall information about a JSP page
  - **Request-time** tags
  - **Scripting elements** such as scriptlets, expressions, and declarations that consist of Java code snippets
  - **Actions** that are used to influence the runtime behavior of a JSP

CODE EXECUTION

- Copy the code for the page with the acknowledgment message into a text file and save it as e.g., main.jsp
- Copy main.jsp in the appropriate folder e.g., CATALINA_HOME/webapps/ROOT - c:/Tomcat8/webapps/ROOT
- Start the Tomcat server
- Start your browser if it is not already running
- In the address area of the browser, type http://localhost:8080/main.jsp
- The output of your JSP page will be displayed

DECLARATIONS

- <%! declaration; [declaration;]+ ... %>

- **<%! int i = 0; %>**
- **<%! int a, b, c; %>**
- **<%! Circle a = new Circle(2.0); %>**

EXPRESSIONS

- <%= expression %>
- <body>
- Today's date: **<%= (new java.util.Date()).toLocaleString()%>**
-
- </body>

JSP DIRECTIVES

- The **jsp directives** are messages that tells the web container how to translate a JSP page into the corresponding servlet.
- There are three types of directives:
- page directive
- include directive
- Taglib directive
- **Syntax of JSP Directive**
- **<%@ directive attribute="value" %>**

PAGE DIRECTIVE

- The page directive defines attributes that apply to an entire JSP page.
- Syntax of JSP page directive

<%@ page attribute="value" %>

- **Attributes of JSP page directive**
- **Import**-Specifies a list of packages or classes for use in the JSP as the Java import statement does for Java classes.
- **contentType**- Defines the character encoding scheme.
- **Extends-**Specifies a superclass that the generated servlet must extend.
- **Info**- Defines a string that can be accessed with the servlet's getServletInfo() method.
- **Buffer-**Specifies a buffering model for the output stream.

ATTRIBUTES PAGE DIRECTIVE

- **Language-** Defines the programming language used in the JSP page.
- **isThreadSafe**-Defines the threading model for the generated servlet.
- **autoFlush**-Controls the behavior of the servlet output buffer.
- **Session**-Specifies whether or not the JSP page participates in HTTP sessions
- **errorPage**-Defines the URL of another JSP that reports on Java unchecked runtime exceptions.
- **isErrorPage**-Indicates if this JSP page is a URL specified by another JSP page's errorPage attribute.

1)import

- The import attribute is used to import class, interface or all the members of a package.
- It is similar to import keyword in java class or interface.

```
<html>
<body>

<%@page import="java.util.Date" %>
Today is :<%=new Date()%>

</body>
</html>
```

2) CONTENT TYPE

- The contentType attribute defines the MIME(Multipurpose Internet Mail Extension) type of the HTTP response.
- The default value is "text/html;charset=ISO-8859-1".

```
<html>
<body>
<%@page contentType=“application/msword” %>
Today is :<%=new Date()%>
</body>
</html>
```

3) BUFFER

- The buffer attribute sets the buffer size in kilobytes to handle output generated by the JSP page.
- The default size of the buffer is 8Kb.

```
<html>
<body>
<%@page buffer="16Kb" %>
Today is :<%=new Date()%>
</body>
</html>
```

INCLUDE DIRECTIVE

- The include directive is used to include the contents of any resource it may be jsp file, html file or text file.
- The include directive includes the original content of the included resource at page translation time (the jsp page is translated only once so it will be better to include static resource)
- Code Reusability
- The filename in the include directive is actually a relative URL. If you just specify a filename with no associated path, the JSP compiler assumes that the file is in the same directory as your JSP.

INCLUDE DIRECTIVE

- **Syntax-**
- **<%@include=“relative url”%>**

TAGLIB DIRECTIVE

- The **taglib** directive declares that your JSP page uses a set of custom tags, identifies the location of the library, and provides means for identifying the custom tags in your JSP page.
- In the custom tag section we will use this tag so it will be better to learn it in custom tag.
- **Syntax-**
- **<%@ taglib uri="uri" prefix="prefixoftag" %>**
- the **uri** attribute value resolves to a location the container understands and the **prefix** attribute informs a container what bits of markup are custom actions.

DECISION MAKING

```
<%! int day = 3; %>
<html> <head>
<title>IF...ELSE Example</title>
</head>
<body>
  <% if (day == 1 || day == 7) { %>
        <p> Today is weekend</p> <% }
  else { %>
        <p> Today is not weekend</p> <% } %>
</body> </html>
```

OPERATORS

Category	Operator
Postfix	() [] . (dot operator)
Unary	++ - - ! ~
Multiplicative	* / %
Additive	+ -
Shift	>> >>> <<
Relational	> >= < <=
Equality	== !=
Bitwise AND	&
Bitwise XOR	^
Bitwise OR	|
Logical AND	&&
Logical OR	||
Conditional	?:
Assignment	= += -= *= /= %= >>= <<= &= ^= |=

DECISION MAKING

```
<%! int day = 3; %>
<html> <head><title>SWITCH...CASE Example</title></head>
<body>
<% switch(day) {
        case 0: out.println("It\'s Sunday."); break;
        case 1: out.println("It\'s Monday."); break;
        ...
        default: out.println("It's Saturday."); } %>
</body> </html>
```

LOOPS

```
<%! int fontSize; %>
<html> <head><title>FOR LOOP Example</title></head>
<body>
        <%for ( fontSize = 1; fontSize <= 3; fontSize++){ %>
        <font color = "green" size = "<%= fontSize %>"> JSP Tutorial
</font><br />
        <%}%>
</body> </html>
```

LOOPS

```
<%! int fontSize; %>
<html> <head><title>WHILE LOOP Example</title></head>
<body>
        <%while ( fontSize <= 3){ %>
        <font color = "green" size = "<%= fontSize %>"> JSP Tutorial </font><br
/>
        <%fontSize++;%>
        <%}%>
</body> </html>
```

IMPLICIT OBJECTS

- These Objects are the Java objects that the JSP Container makes available to the developers in each page and the developer can call them directly without being explicitly declared.
- JSP Implicit Objects are also called **pre-defined variables**.
- There are 9 implicit objects in JSP:

1. **Request-**This is the **HttpServletRequest** object associated with the request.
2. **Response**-This is the **HttpServletResponse** object associated with the response to the client.
3. **Out**- This is the **PrintWriter** object used to send output to the client.

IMPLICIT OBJECTS

4. **Session**-This is the **HttpSession** object associated with the request.
5. **Application**-This is the **ServletContext** object associated with the application context.
6. **Config**-This is the **ServletConfig** object associated with the page.
7. **pageContext**-This encapsulates use of server-specific features like higher performance **JspWriters**.
8. **Page-**This is simply a synonym for **this**, and is used to call the methods defined by the translated servlet class.
9. **Exception**-The **Exception** object allows the exception data to be accessed by designated JSP.

REQUESTS

- The request object is an instance of a **javax.servlet.http.HttpServletRequest** object
- Each time a client requests a page, the JSP engine creates a new object to represent that request
- The request object provides methods to get HTTP header information including **form data, cookies, HTTP methods**, etc

REQUESTS

- Request object methods (examples)
  - **Cookie[] getCookies().** Returns an array containing all of the Cookie objects the client sent with this request
  - **Enumeration getAttributeNames().** Returns an Enumeration containing the names of the attributes available to this request
  - **HttpSession getSession().** Returns the current session associated with the this request, or if the request does not have a session, creates one
  - **Object getAttribute(String name).** Returns the value of the named attribute as an Object, or null if no attribute of the given name exists
  - **String getParameter(String name).** Returns the value of a request parameter as a String, or null if the parameter does not exist

REQUESTS

```
<%@page import="java.io.*, java.util.*"%>
<html>
<body>
<center> <h2>HTTP Header Request Example </h2>
<table border="2" width="100%" align="center">
<tr> <th> Header Name</th> <th> Heaher Values</th>
</tr>
<% Enumeration headerNames= request.getHeaderNames();
while(headerNames.hasMoreElements()){
String paraName=)String)headerNames.nectElement();
out.println("<tr><td>"+paraName+"</td>\n");
String paraValue=request.getHeader(paraName);
out.printlm("<td>"+paraName+"</td></tr>\n");
}%>
</table>
</center>
</body>
</html>
```

SERVER RESPONSE

- When a Web server responds to a HTTP request, the response typically consists of a status line, some response headers, a blank line, and the document
- A typical response looks like this

HTTP/1.1 200 OK

Content-Type: text/html

Header2: ... ...

HeaderN: ... (Blank Line)

<!doctype ...>

<html> <head>...</head> <body> ... </body> </html>

SERVER RESPONSE

- The response object is an instance of a **javax.servlet.http.HttpServletResponse object**
- Just as the server creates the request object, it also creates an object to represent the response to the client
- The response object also defines the interfaces that deal with creating new HTTP headers

SERVER RESPONSE

- Response object methods (examples)
  - **void addCookie(Cookie cookie).** Adds the specified cookie to the response
  - **void sendError(int sc).** Sends an error response to the client using the specified status code and clearing the buffer
  - **void sendRedirect(String location).** Sends a temporary redirect response to the client using the specified redirect location URL
  - **void setHeader(String name, String value).** Sets a response header with the given name and value

SERVER RESPONSE

```
<center> <h2>Auto Refresh Header Example</h2>
<% // Set refresh, autoload time as 5 seconds
response.setIntHeader("Refresh", 5);
Calendar calendar = new GregorianCalendar();
String am_pm;
int hour = calendar.get(Calendar.HOUR);
int minute = calendar.get(Calendar.MINUTE);
int second = calendar.get(Calendar.SECOND);
if(calendar.get(Calendar.AM_PM) == 0) am_pm = "AM";
else am_pm = "PM";
String CT = hour+":"+ minute +":"+ second +" "+ am_pm;
out.println("Current Time is: " + CT + "\n"); %>
</center>
```

JSP PROGRAM CALCULATES FACTORIAL VALUES FOR AN INTEGER NUMBER, WHILE THE INPUT IS TAKEN FROM AN HTML FORM.

```
<html>
<body>
<form action="Factorial.jsp">
Enter a value for n: <input type="text" name="val">
<input type="submit" value="Submit">
</form>
</body>
</html>
```

Factorial.jsp

```
<html>
<body>
<%!
long n, result;
String str;
```

JSP program calculates factorial values for an integer number, while the input is taken from an HTML form.

```
long fact(long n) {
if(n==0)
return 1;
else
return n*fact(n-1);
}
%>
<%
str = request.getParameter("val");
n = Long.parseLong(str);
result = fact(n);
%>
<b>Factorial value: </b> <%= result %>
</body>
</html>
```

JSP PROGRAM SHOWS THE SYSTEM DATE AND TIME.

```
<html>
<body>
<%-- JSP comments --%>
<%@page import="java.util.Date"%>
<%!
Date date;
%>
<%
date = new Date();
%>
<b>System date and time: </b> <%= date %>
</body>  </html>
```

JSP PROGRAM CALCULATES POWERS OF 2 FOR INTEGERS IN THE RANGE 0-10.

```
<html>
<head>
<title>Powers of 2</title>
</head>
<body>
<center>
<table border="2" align="center">
<th>Exponent</th>
<th>2^Exponent</th>
<% for (int i=0; i<=10; i++) { //start for loop
%>
<tr>
<td><%= i%></td>
<td><%= Math.pow(2, i) %></td>
</tr>
<% } %> //end for loop
</table>
</center>
</body>
</html>
```

JSP ACTIONS

- These actions use constructs in XML syntax to control the behavior of the servlet engine.
- You can dynamically insert a file, reuse JavaBeans components, forward the user to another page, or generate HTML for the Java plugin.
- There is only one syntax for the Action element, as it conforms to the XML standard –
- **<jsp:action_name attribute="Value"/>**

JSP ACTIONS

- Action elements are basically predefined functions.
- **<jsp:include>-** Includes a file at the time the page is requested.
- **<jsp:useBean>** -Finds or instantiates a JavaBean.
- **<jsp:setProperty>** - Sets the property of a JavaBean.
- **<jsp:getProperty>-**Inserts the property of a JavaBean into the output.
- **<jsp:forward>** -Forwards the requester to a new page.
- **<jsp:plugin>** -Generates browser-specific code that makes an OBJECT or EMBED tag for the Java plugin.

JSP ACTION ELEMENTS

- **<jsp:element>** -Defines XML elements dynamically.
- **<jsp:attribute>-**Defines dynamically-defined XML element's attribute.
- **<jsp:body>-**Defines dynamically-defined XML element's body.
- **<jsp:text>-**Used to write template text in JSP pages and documents.

COMMON ATTRIBUTES IN JSP ACTION

- There are two attributes that are common to all Action elements:

1. the **id** attribute
2. the **scope** attribute.

- **Id attribute**
- The id attribute uniquely identifies the Action element, and allows the action to be referenced inside the JSP page. If the Action creates an instance of an object, the id value can be used to reference it through the implicit object PageContext. The id attribute and the scope attribute are directly related, as the scope attribute determines the lifespan of the object associated with the id

- **Scope attribute**
- This attribute identifies the lifecycle of the Action element. The scope attribute has four possible values: **page, request, session**, and **application**.

EXAMPLE- USE OF INCLUDE ACTION

```
<html>
<body> <h2 align="center"> The include action example >/h2?
<jsp:include page="Date.jsp" flush="true"/>
</body>
</html>
```

Date.jsp

```
<html>
<body>
<p> Today's Date: <%= (new.java.util.Date()).toLocaleString()%> </p>
</body>    </html>
```

JSP PROGRAM SHOWS THE USE OF JSP FORWARD ACTION TAG.

```
<html>
<head>
<title>JSP Page</title>
</head>
<body>
<% if (Math.random() > 0.5) { %>
<jsp:forward page="PowersOf2.jsp" />
<% }
```

- else { %>

```
<jsp:forward page="SineTable.jsp"
/>
<% } %>
</body>
</html>
```

FORMS

- JSP handles form data parsing automatically using the following methods
  - **getParameter()** – You call **request.getParameter()** method to get the value of a form parameter
  - **getParameterValues()** – Call this method if the parameter appears more than once and returns multiple values, for example checkbox
  - **getParameterNames()** – Call this method if you want a complete list of all parameters in the current request
  - **getInputStream()** – Call this method to read binary data stream coming from the client

FORMS

- The following URL will pass two values to HelloForm program using the GET method.

http://localhost:8080/main.jsp?first_name=ME&last_name=METOO

```
<ul> <li><p><b>First Name:</b> <%= request.getParameter("first_name")%> </p>
</li>
<li><p><b>Last Name:</b> <%= request.getParameter("last_name")%> </p></li>
</ul>
```

FORMS

```
<form action = "main.jsp" method = “POST">
First Name: <input type = "text" name = "first_name"> <br />
Last Name: <input type = "text" name = "last_name" />
<input type = "submit" value = "Submit" />
</form>
```

FORMS

```
<h1>Using POST Method to Read Form Data</h1>

<ul> <li><p><b>First Name:</b>

<%= request.getParameter("first_name")%> </p></li>

<li><p><b>Last Name:</b>

 <%= request.getParameter("last_name")%> </p></li>
</ul>
```

FORMS-EXAMPLE WITH CHECKBOXES:

```
<form action = "main.jsp" method = "POST" target = "_blank">
<input type = "checkbox" name = "maths" checked = "checked" /> Maths
<input type = "checkbox" name = "physics" /> Physics
<input type = "checkbox" name = "chemistry" checked = "checked" /> Chemistry
<input type = "submit" value = "Select Subject" />
</form>

<h1>Reading Checkbox Data</h1>
<ul> <li><p><b>Maths Flag:</b> <%= request.getParameter("maths")%> </p></li>
<li><p><b>Physics Flag:</b> <%= request.getParameter("physics")%> </p></li>
<li><p><b>Chemistry Flag:</b> <%= request.getParameter("chemistry")%> </p></li>
</ul>
```

Results:

- **Maths Flag :**: on
- **Physics Flag:**: null
- **Chemistry Flag:**: on

COOKIES

- A JSP that sets a cookie might send headers that look something like this

  HTTP/1.1 200 OK

  Date: Fri, 04 Feb 2000 21:03:38 GMT

  Server: Apache/1.3.9 (UNIX) PHP/4.0b3

  Set-Cookie: name = xyz; expires = Friday, 04-Feb-07 22:03:38 GMT; path = /; domain = blabla.com

  Connection: close Content-Type: text/html

- If the browser is configured to store cookies, it will then keep this information until the expiry date
- A JSP script will then have access to the cookies through the request method ***request.getCookies()*** which returns an array of *Cookie* objects

COOKIES

- Setting cookies with JSP involves three steps
  - **Step 1: Creating a Cookie object**
    - You call the Cookie constructor with a cookie name and a cookie value

```
Cookie cookie = new Cookie("key","value");
```

    - Keep in mind, neither the name nor the value should contain white space or any of the following characters [ ] ( ) = , " / ? @ : ;
  - **Step 2: Setting the maximum age**
    - You use **setMaxAge** to specify how long (in seconds) the cookie should be valid
    - The following code will set up a cookie for 24 hours

```
cookie.setMaxAge(60*60*24);
```

COOKIES

- **Step 3: Sending the Cookie into the HTTP response headers**
  - You use **response.addCookie** to add cookies in the HTTP response header as follows

```
response.addCookie(cookie);
```

COOKIES

```
<%
Cookie firstName = new Cookie("first_name", request.getParameter("first_name"));
Cookie lastName = new Cookie("last_name", request.getParameter("last_name"));
// Set expiry date after 24 Hrs for both the cookies.
firstName.setMaxAge(60*60*24);
lastName.setMaxAge(60*60*24);
// Add both the cookies in the response header.
response.addCookie( firstName );
response.addCookie( lastName );
%>
```

COOKIES

- To read cookies, you need to create an array of *javax.servlet.http.Cookie* objects by calling the **getCookies( )** method of *HttpServletRequest*
- Use **getName()** and **getValue()** methods to access each cookie and associated value

```
<%
Cookie cookie = null;
Cookie[] cookies = null;
cookies = request.getCookies();
if( cookies != null ) {
        out.println("<h2> Found Cookies Name and Value</h2>");
        for (int i = 0; i < cookies.length; i++) {
                cookie = cookies[i];
                out.print("Name : " + cookie.getName( ) + ", ");
                out.print("Value: " + cookie.getValue( )+" <br/>"); } }
else { out.println("<h2>No cookies founds</h2>"); } %>
```

COOKIES

- If you want to delete a cookie, then you simply need to follow these three steps
  - Read an already existing cookie and store it in a Cookie object
  - Set cookie age as zero using the **setMaxAge()** method to delete an existing cookie
  - Add this cookie back into the response header

SESSIONS

- JSP makes use of the servlet provided HttpSession Interface
- By default, JSPs have session tracking enabled and a new HttpSession object is instantiated for each new client automatically
- Disabling session tracking requires explicitly turning it off by

<%@ page session = "false" %>

SESSIONS

```
<%@ page import = "java.io.*,java.util.*" %>
<%
Date createTime = new Date(session.getCreationTime());
Date lastAccessTime = new Date(session.getLastAccessedTime());
String title = "Welcome Back to my website";
Integer visitCount = new Integer(0);
String visitCountKey = new String("visitCount");
String userIDKey = new String("userID");
String userID = new String("ABCD");
```

SESSIONS

```
if (session.isNew() ){
        title = "Welcome to my website"; session.setAttribute(userIDKey, userID);
        session.setAttribute(visitCountKey, visitCount); }
        visitCount = (Integer)session.getAttribute(visitCountKey);
        visitCount = visitCount + 1;
%>
```

EXAMPLE-SESSION TRACKING IN JSP

```
<html>
<body> <h2 align="center"> Session Tracking </h2>
<table border="2" align="center">
<tr bgcolor="#949494">
<th> Session Info</th>
<th> Value</th>
</tr>  <tr>
<td> id </td>
<td><% out.print(session.getId());%> </td>
</tr>
<tr>
<td> Creation Time:</td>
<td> <% out.print (createTime());%>
</td></tr>
<tr>
<td> Time of Last Access :</td>
<td> <% out.print (lastAccessTime());%>
</td></tr>
```

EXAMPLE-SESSION TRACKING IN JSP

```
<tr>
<td> UserID:</td>
<td> <% out.print (UserID);%>
</td></tr>
<tr>
<td> Number of visits:</td>
<td> <% out.print (visitCount);%>
</td>
</tr>
</table>
</body>
</html>
```

OUTPUT

Welcome to my website

Session Information

Session info	value
id	0AE3EC93FF44E3C525B4351B77ABB2D5
Creation Time	Tue Jun 08 17:26:40 GMT+04:00 2010
Time of Last Access	Tue Jun 08 17:26:40 GMT+04:00 2010
User ID	ABCD
Number of visits	0