102010505
Advanced Java

Unit-4
Java Server Pages

JSP Overview

What is Java Server Pages (JSP)?

- Java Server Pages (JSP) is a technology for developing web pages that support dynamic content.
- It helps to insert java code in HTML pages by making use of special JSP tags.

  Example

```
<% …JSP Tag… %>
```

- JSP is a server-side program that is similar in design and functionality to java servlet.
- A JSP page consists of HTML tags and JSP tags.
- JSP pages are saved with .jsp extension

JSP vs Servlet

Comparing JSP with Servlet

JSP	Servlet
JSP is a webpage scripting language that generates dynamic content.	Servlets are Java programs that are already compiled which also creates dynamic web content.
A JSP technically gets converted to a servlet It embed the java code into HTML. E.g. <html> <% java code %> </html>	A servlet is a java class. It put HTML into print statements. E.g. out.println("<html code>");
JSPs are extension of servlets which minimizes the effort of developers to write User Interfaces using Java programming.	A servlet is a server-side program and written purely in Java.
JSP runs slower than servlet. As, it has the transition phase for converting from JSP to a Servlet. Once it is converted to a Servlet then it will start the compilation	Servlets run faster than JSP
In MVC architecture JSP acts as view.	In MVC architecture Servlet acts as controller.
We can build custom tags using JSP API	We cannot build any custom tags in servlet.

Advantages of JSP over Servlets

1. In a JSP page visual content and logic are separated, which is not possible in a servlet.

   i.e. JSP separates business logic from the presentation logic.

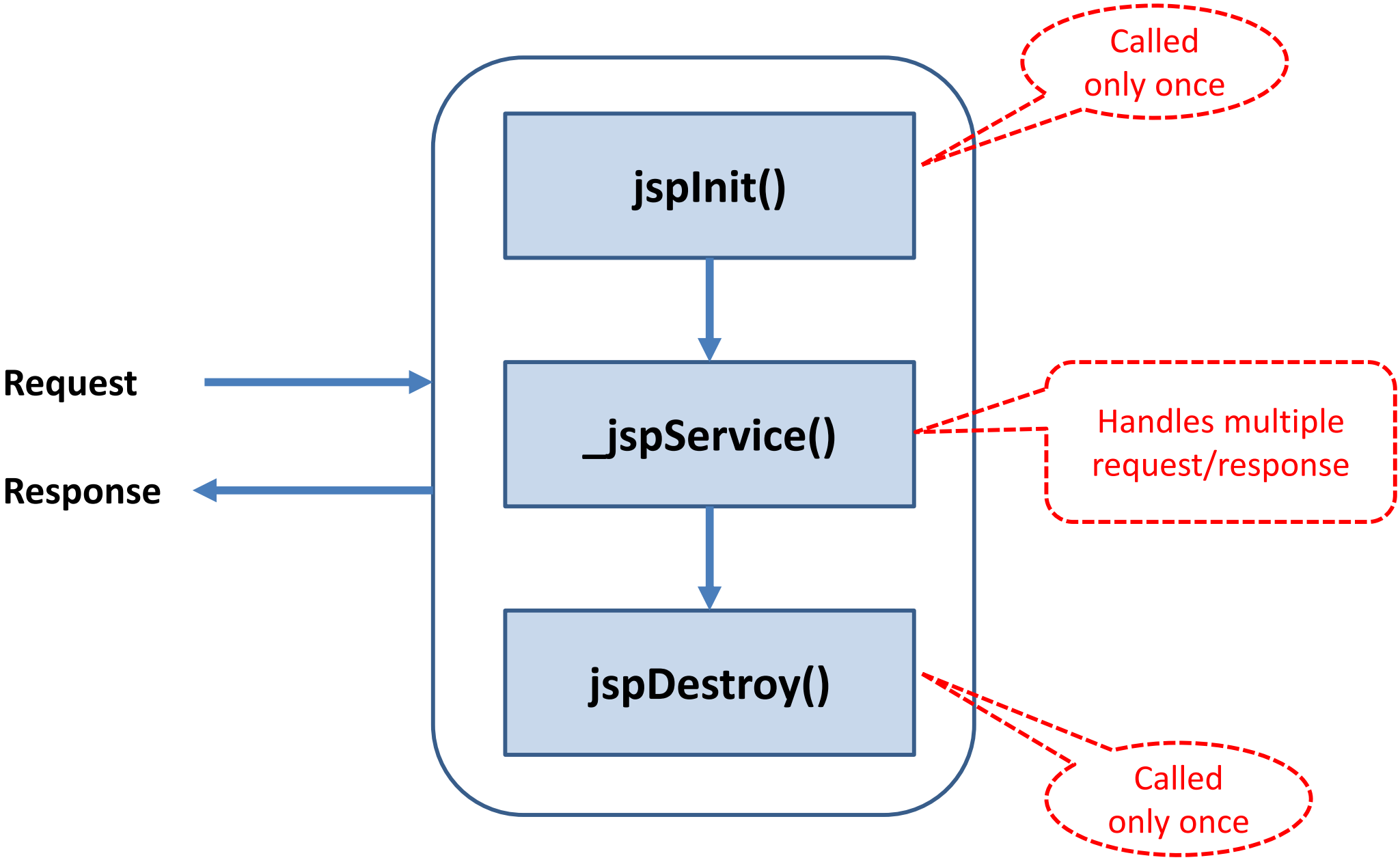
2. Servlets use *println* statements for printing an HTML document which is usually very difficult to use. JSP has no such tedious task to maintain.

Life Cycle of JSP

Life Cycle of JSP

- A JSP life cycle can be defined as the entire process from its creation till the destruction.
- It is similar to a servlet life cycle with an additional step which is required to compile a JSP into servlet.
- A JSP page is converted into Servlet in order to service requests.
- The translation of a JSP page to a Servlet is called **Lifecycle of JSP**.

Life Cycle of JSP

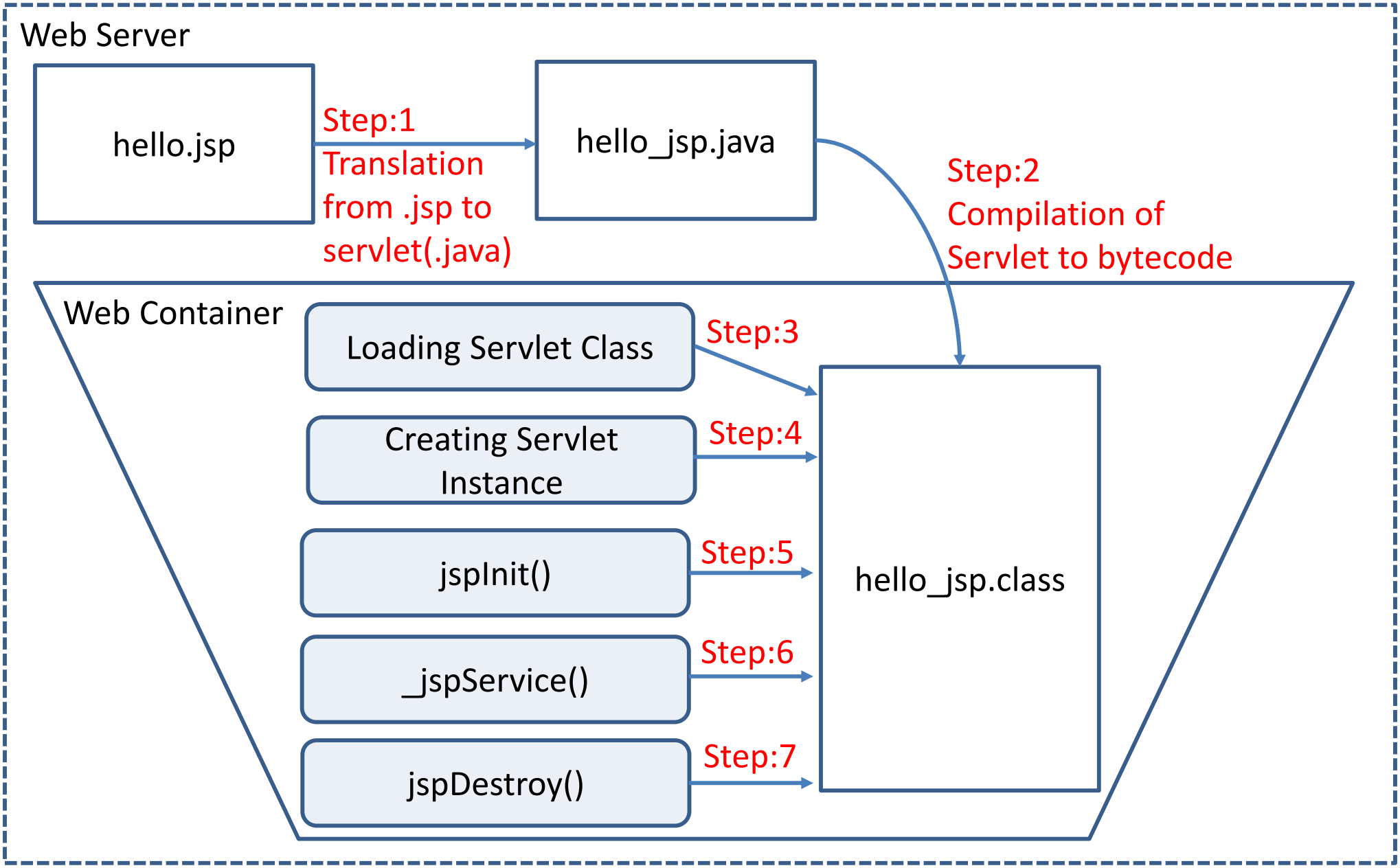


Life Cycle of JSP

JSP Lifecycle steps

1. Translation of JSP to Servlet code.
2. Compilation of Servlet to bytecode.
3. Loading Servlet class.
4. Creating servlet instance.
5. Initialization by calling jspInit() method
6. Request Processing by calling _jspService() method
7. Destroying by calling jspDestroy() method

JSP Processing Life Cycle



Life Cycle of JSP

- **Web Container** translates JSP code into a **servlet source(.java) file.**
- Then compiles that(.java) into a java servlet class **(bytecode)**.
- In the third step, the servlet class bytecode is loaded using classloader in web container.
- The Container then creates an instance of that servlet class.
- The initialized servlet can now service request.
- For each request the **Web Container** call the **_jspService()** method.
- When the Container removes the servlet instance from service, it calls the **jspDestroy()** method to perform any required clean up.

JSP Processing

The following steps explain how the web server creates the web page using JSP:

1. Web browser sends an HTTP request to the web server requesting JSP page. E.g. http://localhost:8080/1.jsp
2. Web server recognizes that the HTTP request by web browser is for JSP page by checking the extension of the file (i.e .jsp)
3. Web server forwards HTTP Request to JSP engine.
4. The JSP engine loads the JSP page from disk and converts it into a servlet content.
5. The JSP engine compiles the servlet into an executable class and forwards the original request to a servlet engine.

JSP Processing

6. Servlet engine loads and executes the Servlet class.
7. Servlet produces an output in HTML format.
8. Output produced by servlet engine is then passes to the web server inside an HTTP response.
9. Web server sends the HTTP response to Web browser in the form of static HTML content.
10. Web browser loads the static page into the browser and thus user can view the dynamically generated page.

"Except the translation phase,
a JSP page is handled exactly like a Servlet"

JSP Processing

Translation Time

Time taken to generate Java Servlet (.java) from .jsp file is termed as **Translation Time**.

Request Time

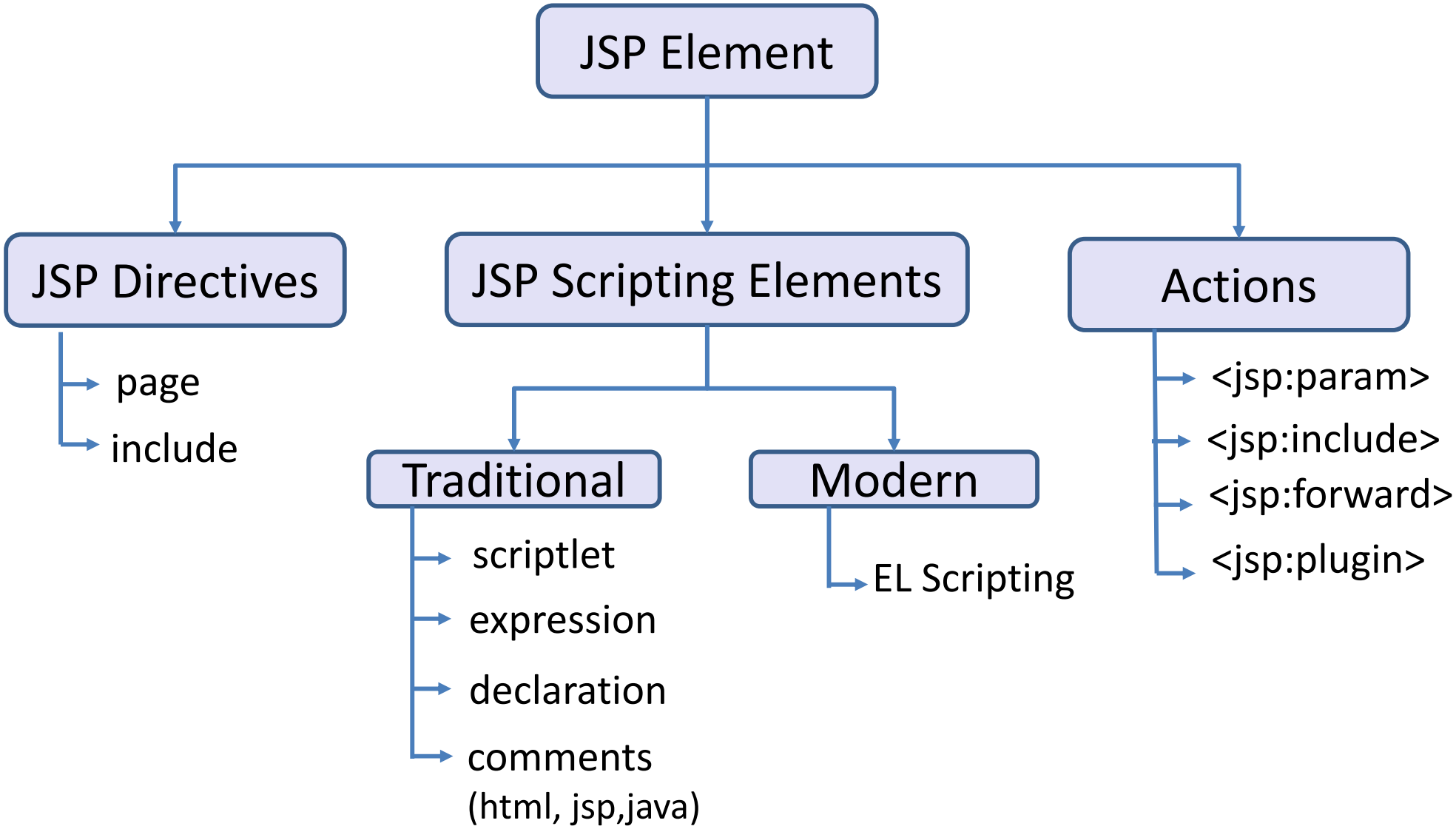
Time taken to invoke a Servlet to handle an HTTP request is termed as **Request Time**.

GTU question

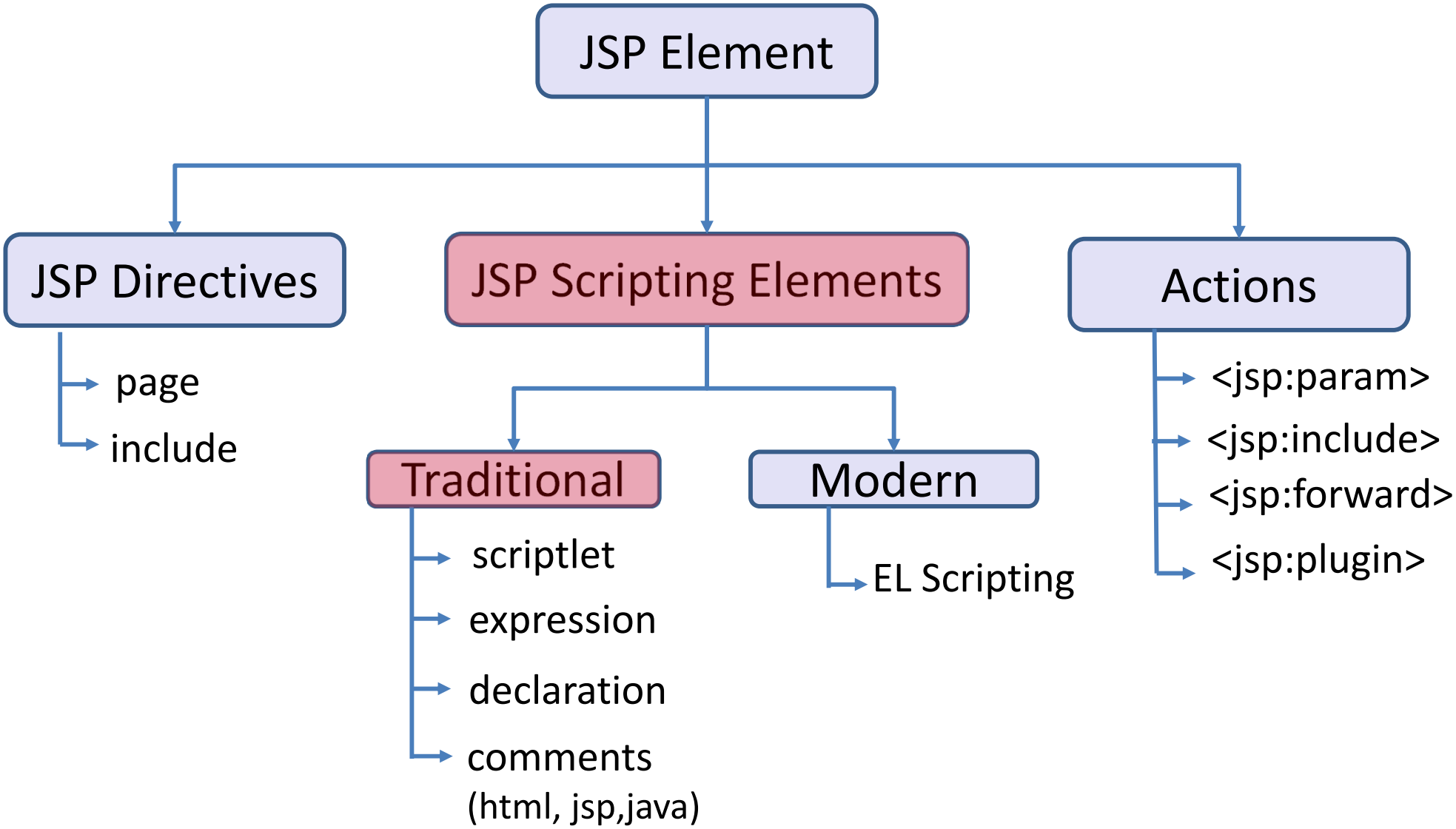
1.	List and explain various phases of JSP life cycle?	Win’17 Win’18 Win’19

JSP Elements

JSP Elements



JSP Elements

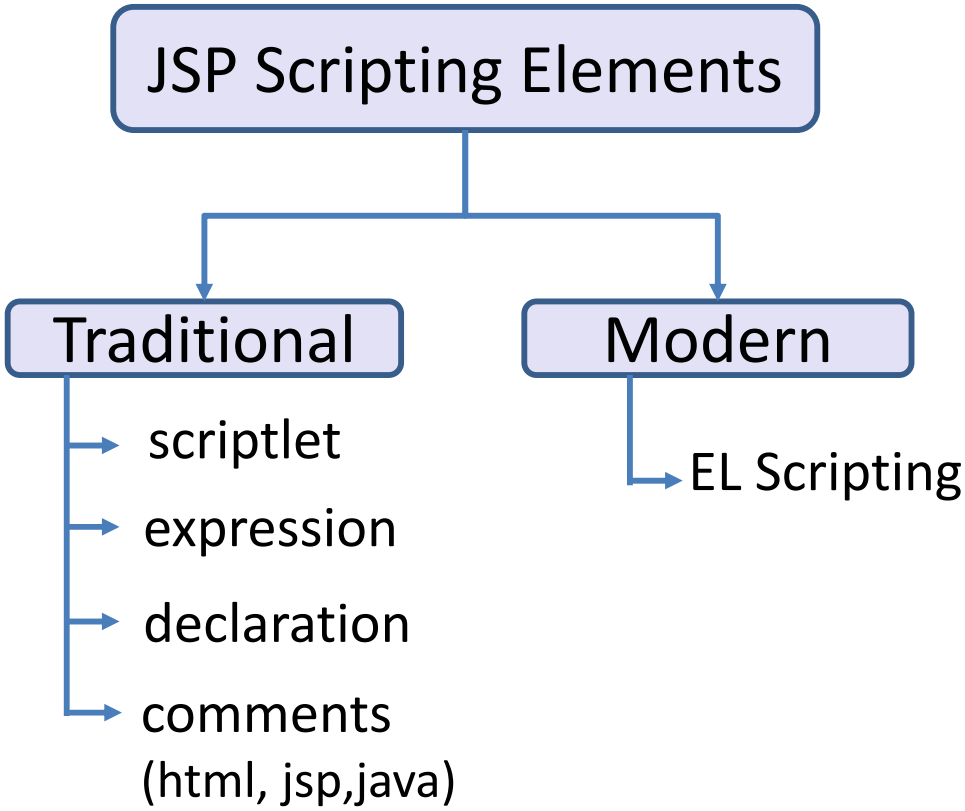


JSP Scripting Elements

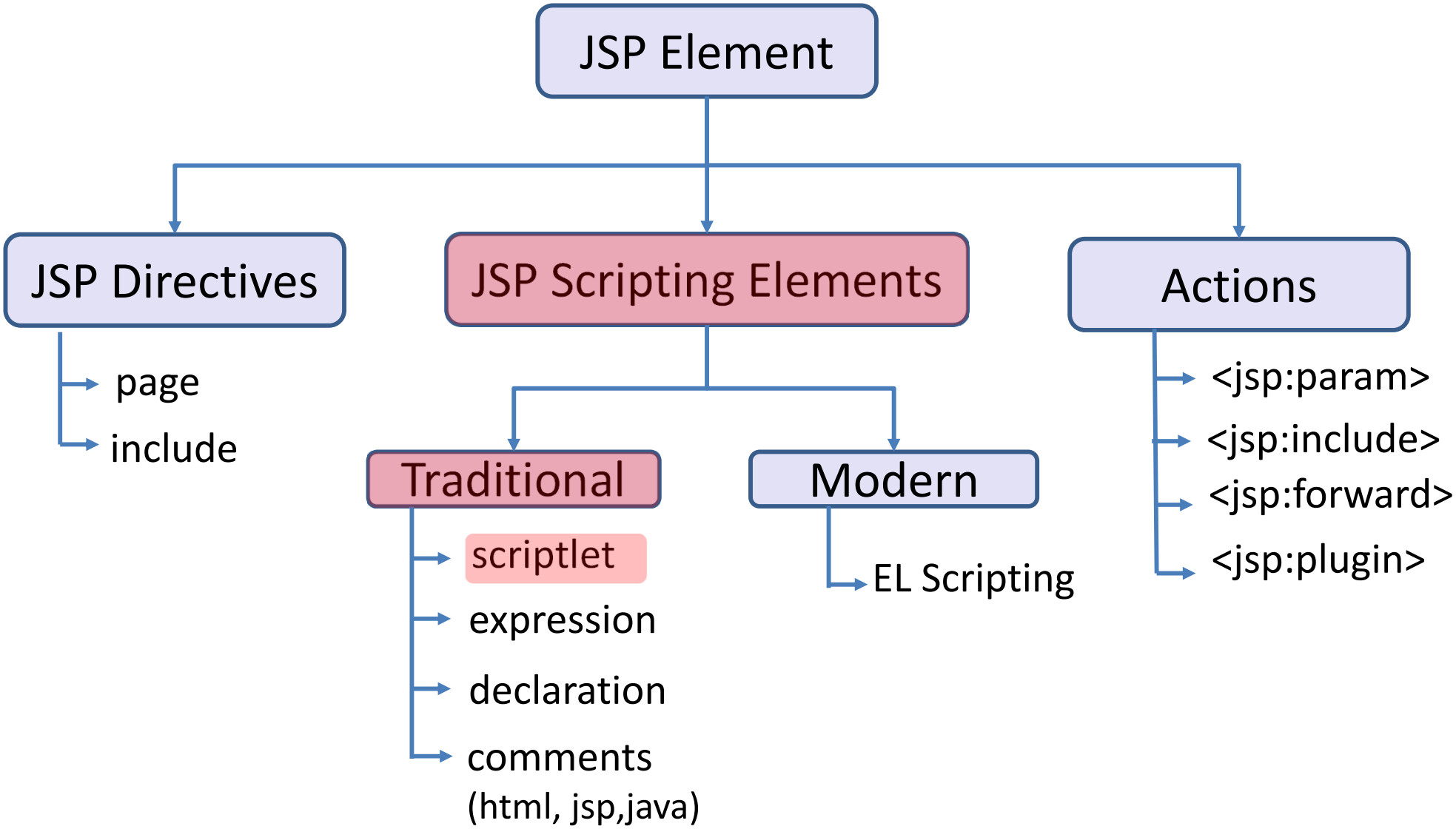
JSP Scripting Elements

- The scripting elements provides the ability to insert java code inside the jsp. There are three types of traditional scripting elements:

1. scriptlet tag
2. expression tag
3. declaration tag



JSP Elements



scriptlet

- A scriptlet tag is used to execute java source code in JSP.
- A scriptlet can contain
  1. Any number of JAVA language statements
  2. Variable
  3. Method declarations
  4. Expressions that are valid in the page scripting language

Syntax

```
<%   // java source code    %>
```

Example

```
<% out.print("welcome to jsp"); %>

<%   int a=10;    %>
```

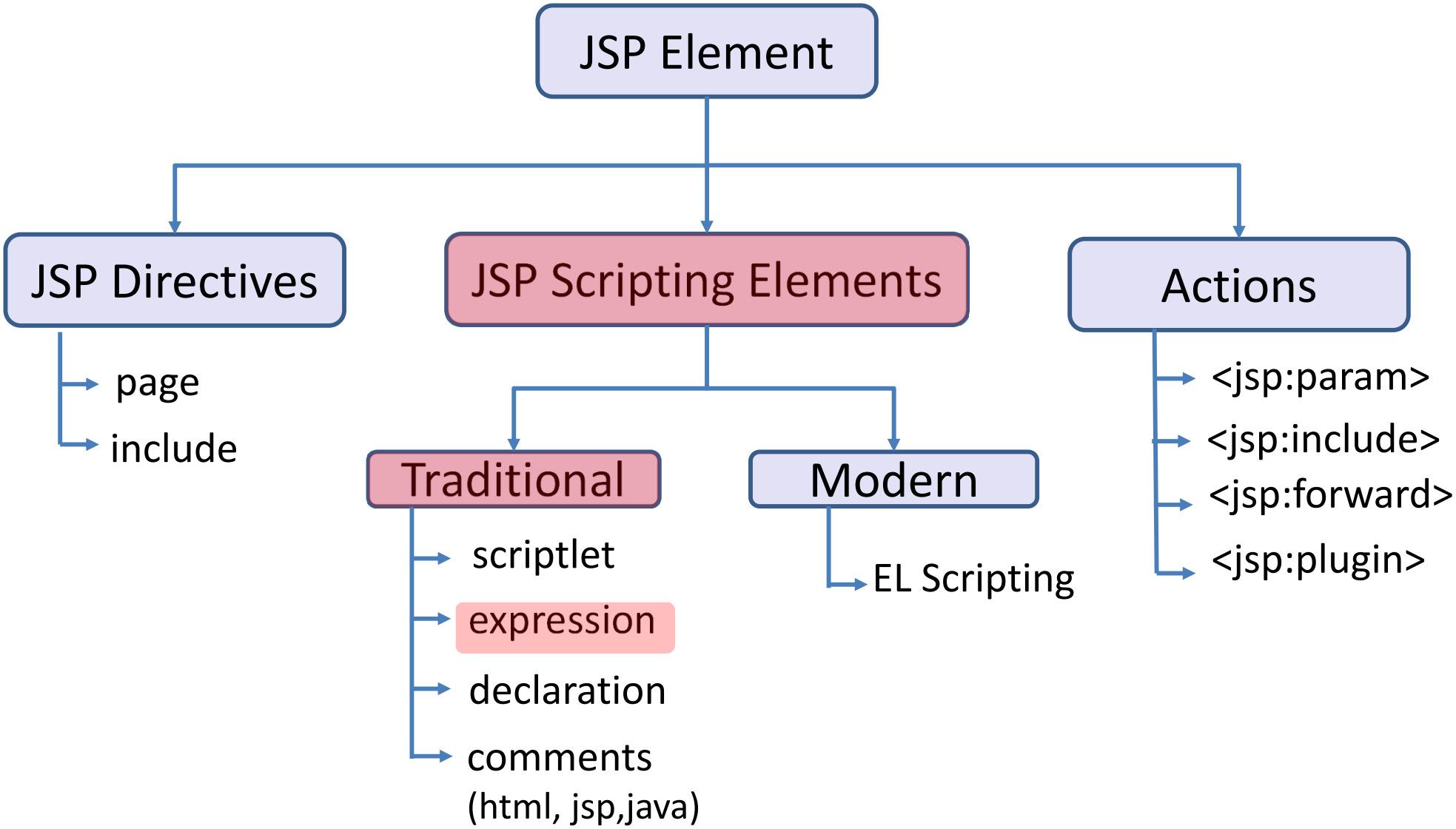
scriptlet

- Everything written inside the scriptlet tag is compiled as java code.
- JSP code is translated to Servlet code, in which **_jspService()** method is executed which has HttpServletRequest and HttpServletResponse as argument.
- JSP page can have any number of scriptlets, and each scriptlets are appended in _jspService ().

First jsp program: First.jsp using scriptlet

```
1. <html>
2. <body>
3. <%out.println("Hello World! My First JSP
                                        Page");%>
4. </body>
5. </html>
```

JSP Elements



expression

- The code placed within **JSP expression tag** is *written to the output stream of the response*.
- So you need not write out.print() to write data.
- It is mainly used to print the values of variable or method.

Syntax

```
<%=statement %>
```

Example

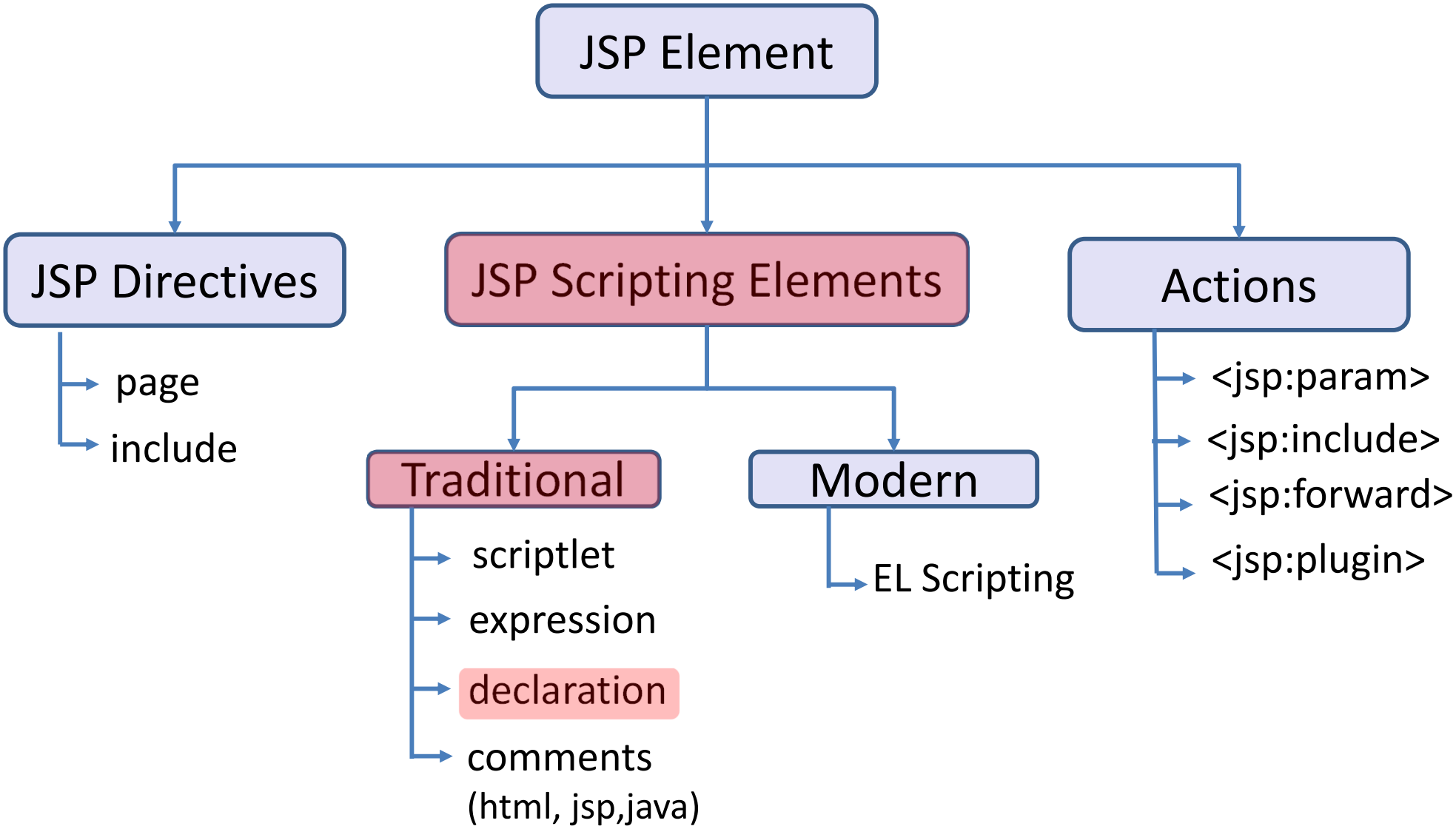
```
<%= (2*5) %>
```

turns out as

```
out.print((2*5));
```

"Do not end the statement with semicolon in case of expression tag."

JSP Elements



declaration

- The **JSP declaration tag** is used *to declare variables and methods*
- The declaration of jsp declaration tag is placed outside the _jspService() method.

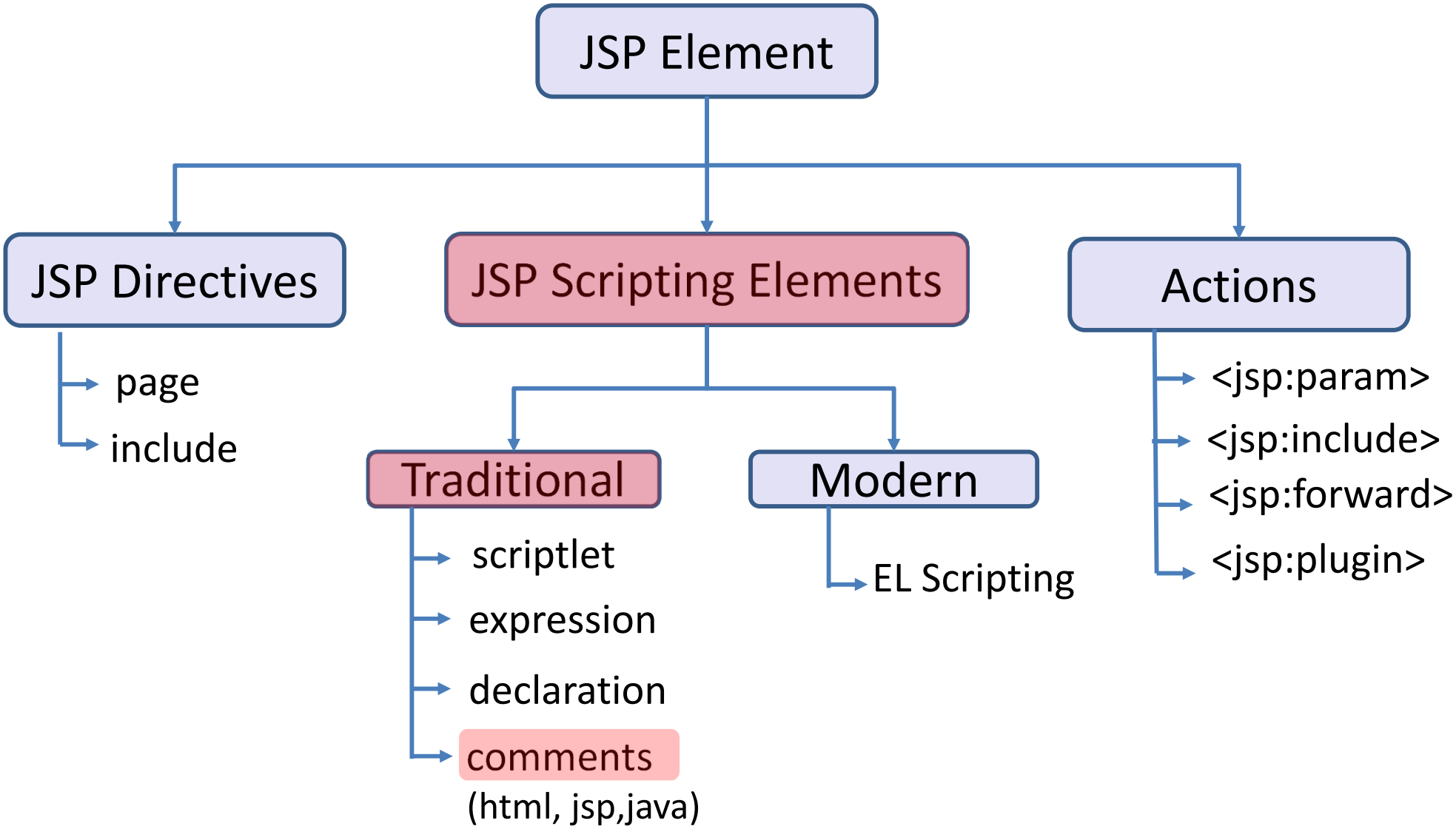
Syntax

```
<%!  variable or method declaration %>
```

Example

```
<%! int a = 10; %>
<%! int a, b, c; %>
<%! Circle a = new Circle(2.0); %>
```

JSP Elements



comments

- The comments can be used for documentation.
- This JSP comment tag tells the JSP container to ignore the comment part from compilation.

Syntax

```
<%-- comments --%>
```

JSP comment	<%-- jsp comment --%>
Java comment	/* java comment */ or // for single line
Html comment	<!-- html comment -->

Scripting Elements: Example

```
1. <html>
2. <body>
3. <%-- comment:JSP Scipting elements --%>
4. <%! int i=0; %>
5. <% i++; %>
6. Welcome to world of JSP!
7. <%= "This page has been accessed " + i + " times" %>
8. </body>
9. </html>
```

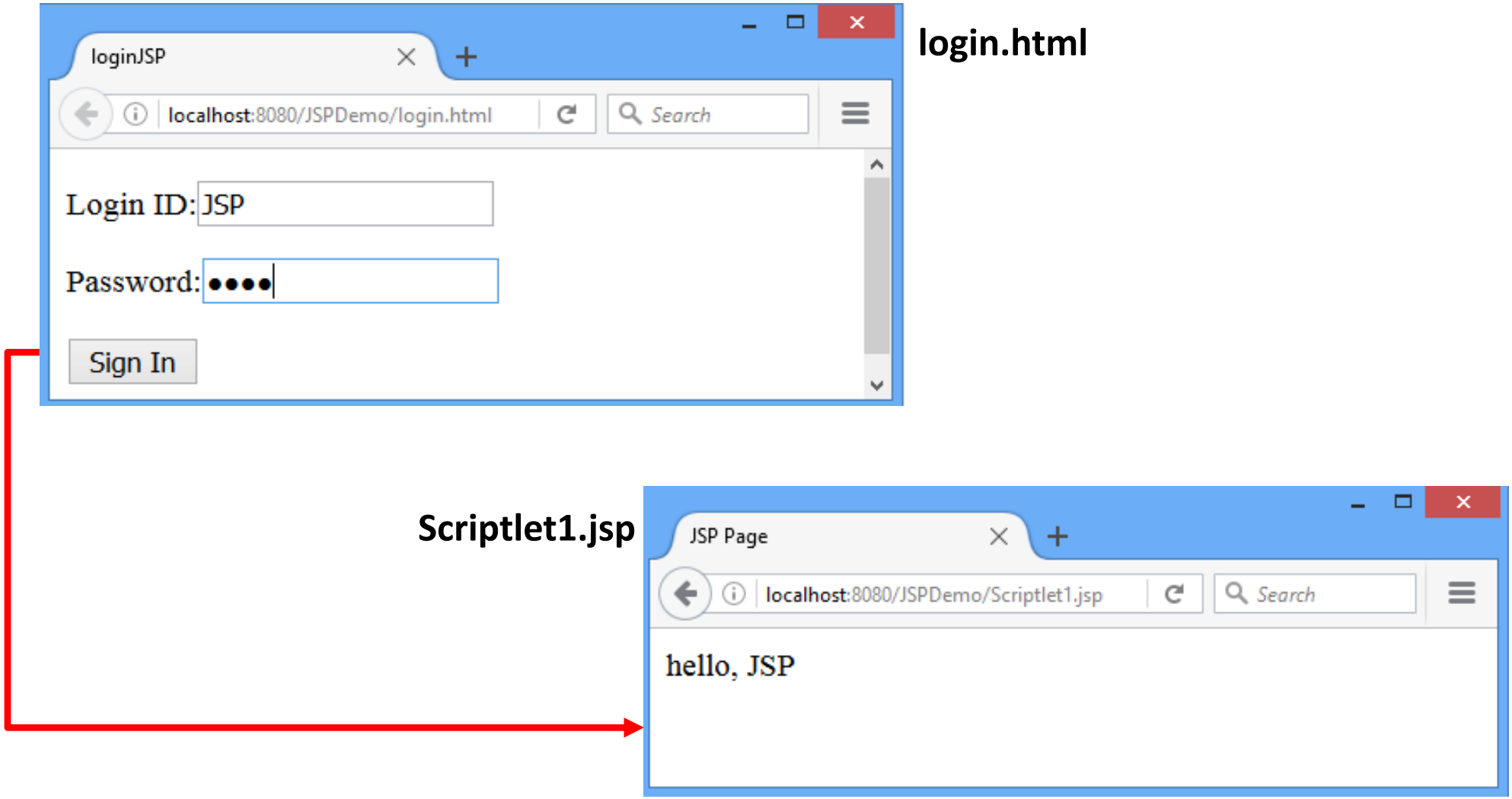
declaration

scriptlet

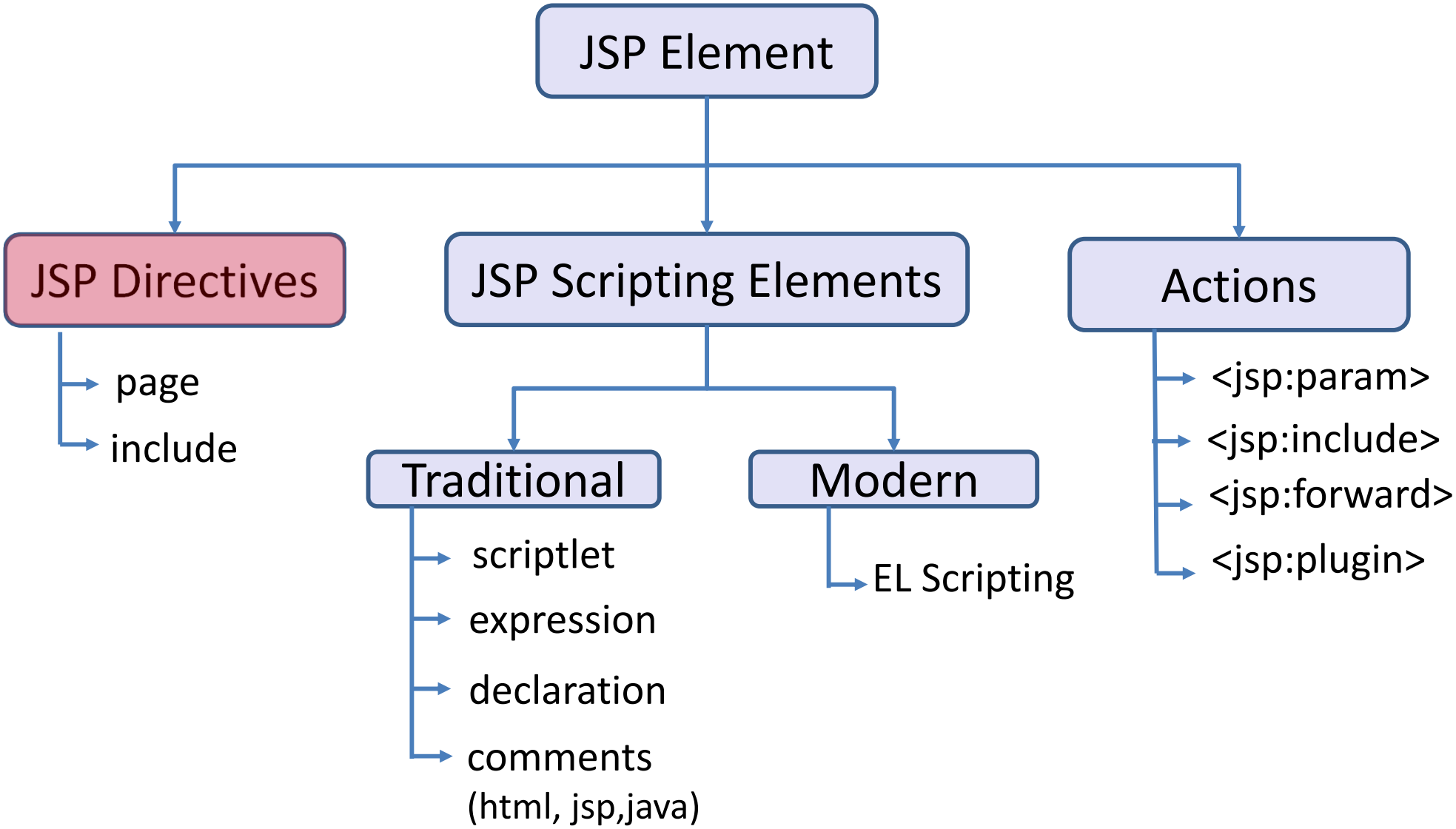
expression

Scriptlet

- Write a JSP program to welcome authenticated user.



JSP Elements



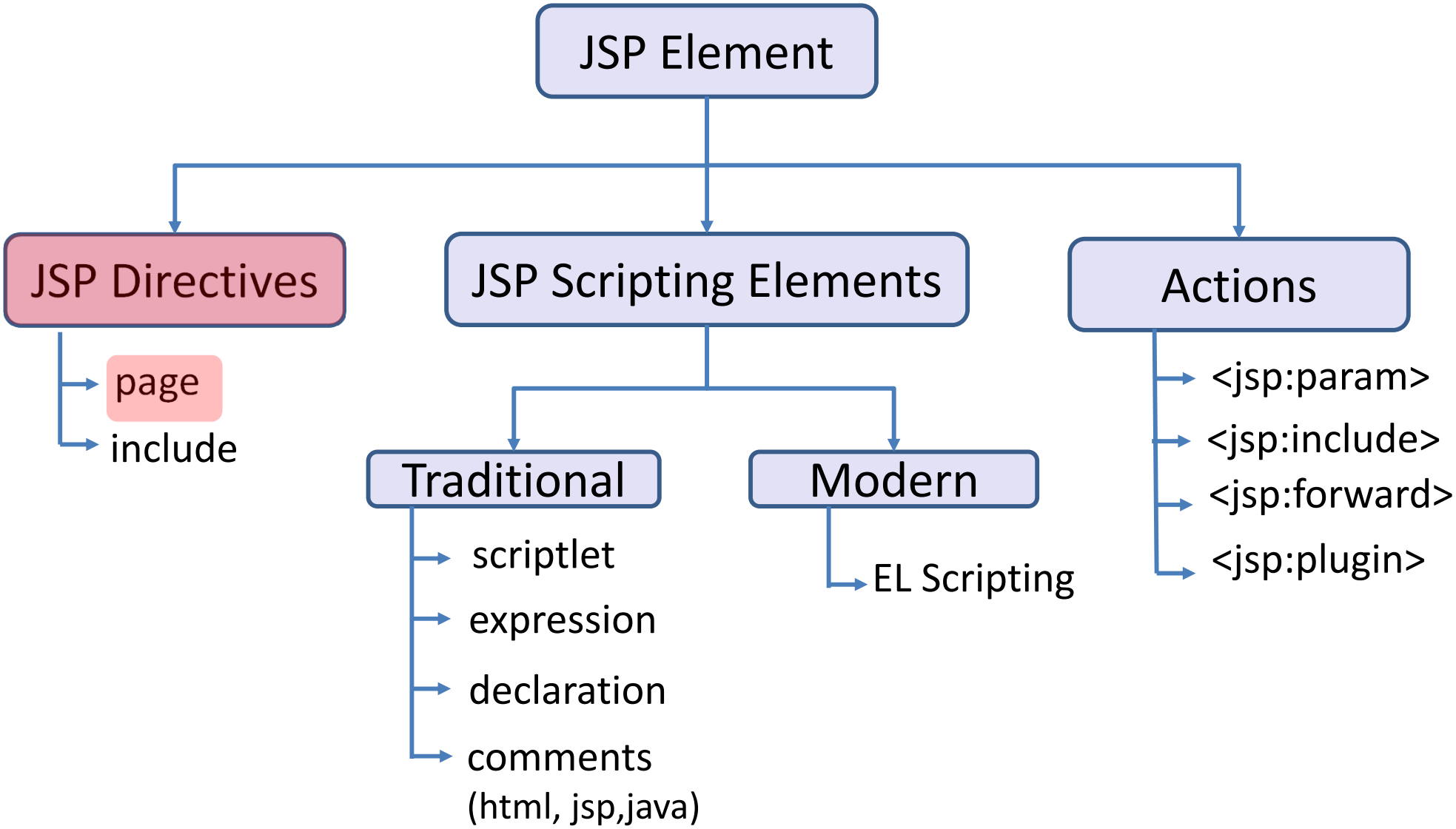
JSP Directives

- JSP directives provide directions and instructions to the container, telling it how to translate a JSP page into the corresponding servlet.
- A JSP directive affects the overall structure of the servlet class.
- JSP engine handles directives at Translation time.
- There are two types of directives:
  1. page directive
  2. include directive

Syntax

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

JSP Elements



JSP Directives

page directive

page directive

- The page directive defines attributes that apply to an entire JSP page.
- You may code page directives anywhere in your JSP page.
- By convention, page directives are coded at the top of the JSP page.

Syntax

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

Example

```
<%@page import="java.util.Date,java.util.List,java.io.*" %>
<%@page contentType="text/html; charset=US-ASCII" %>
```

page directive

Attributes of JSP page directive

1. import
2. contentType
3. extends
4. info

Used to import class, interface or all the members of a package.

<%@ page **import**="java.util.Date" %>

Today is: <%= **new Date()** %>

The contentType attribute defines the MIME type of the HTTP response. The default value is text/html;charset=ISO-8859-1.

<%@ page **contentType**="application/msword %>

The extends attribute defines the parent class that will be inherited by the generated servlet.

<%@ page **extends**="javax.servlet.HttpServlet" %>

This attribute simply sets the information of the JSP page which is retrieved later by using getServletInfo() .

<%@ page **info**="Authored by : AuthorName" %>

page directive

5. buffer
6. language
7. isELIgnored
8. autoFlush

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

```
<%@ page buffer="16kb" %>
<%@ page buffer="none" %>
```

The language attribute specifies the scripting language used in the JSP page. The default value is "java".

```
<%@ page language="java" %>
```

We can ignore the Expression Language (EL) in jsp by the isELIgnored attribute. By default its value is false i.e. EL is enabled by default.

```
<%@ page isELIgnored="true" %>//Now EL will be ignored
```

The **autoFlush** attribute specifies whether buffered output should be flushed automatically when the buffer is filled. By default it is true.

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

page directive

9. isThreadSafe
10. session
11. pageEncoding
12. errorPage
13. isErrorPage

This option marks a page as being thread-safe. By default, all JSPs are considered thread-safe(true). If you set the isThreadSafe = false, the JSP engine makes sure that only one thread at a time is executing your JSP.

<%@ page **isThreadSafe** ="false" %>

The session attribute indicates whether or not the JSP page uses HTTP sessions.

<%@ page **session**="true" %>//Bydefault it is true

We can set response encoding type with this page directive attribute, its default value is “ISO-8859-1”.

<%@ page **pageEncoding** ="US-ASCII" %>

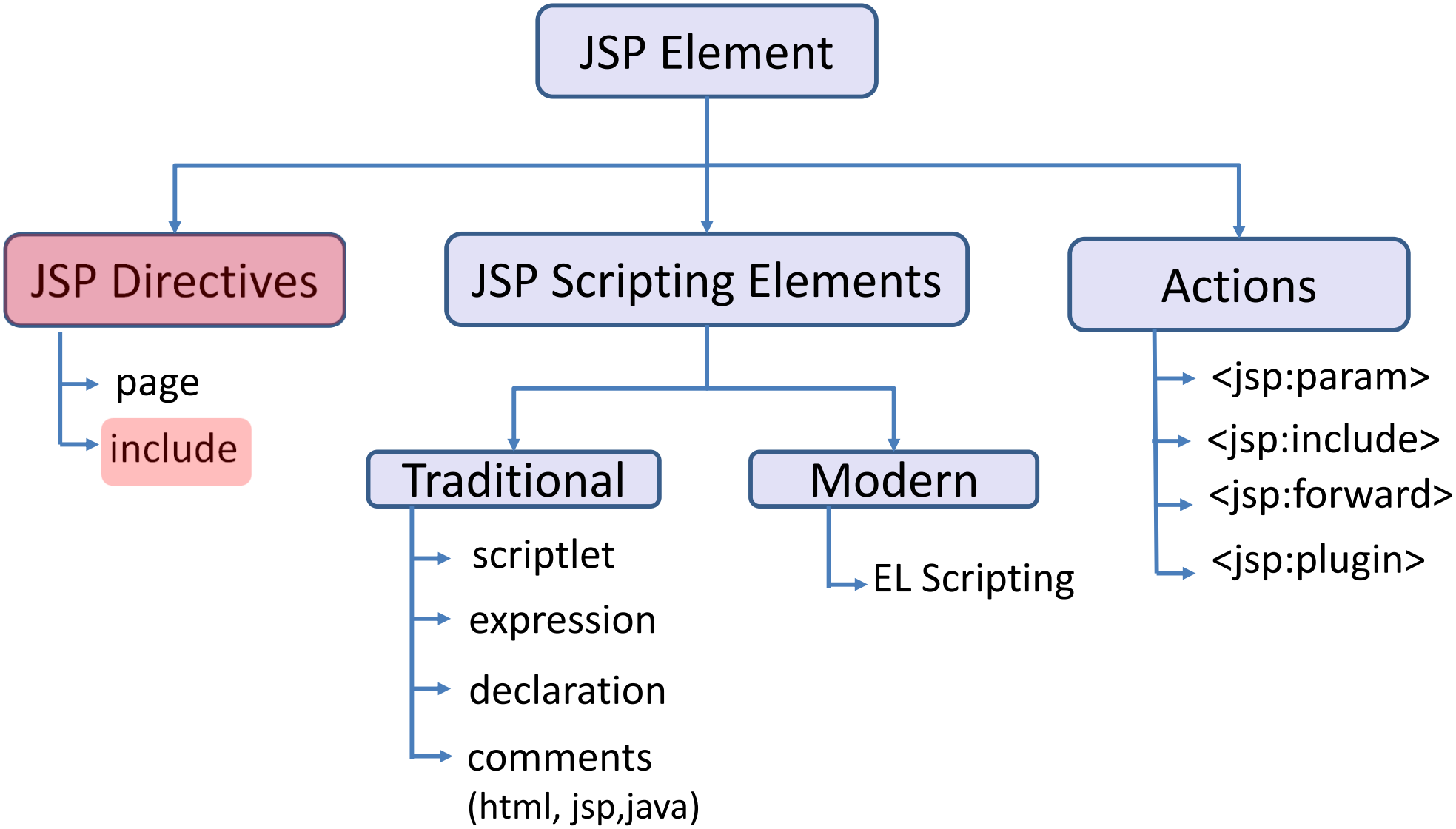
It is used to define the error page, if exception occurs in the current page, it will be redirected to the error page.

<%@ page **errorPage**="myerrorpage.jsp" %>

The isErrorPage attribute is used to declare that the current page is the error page.

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

JSP Elements



JSP Directives

include directive

include directive

- JSP include directive is used to include the contents of another file to the current JSP page during translation time.
- The included file can be HTML, JSP, text files etc.

Advantage of Include directive

- Code Reusability
- Syntax

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

- Example

```
<%@ include file="1.jsp" %>
```

JSP Implicit Object

Jsp Implicit Objects

- There are **9 jsp implicit objects**.
- These objects are *created by the web container* that are available to all the jsp pages.

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

Jsp Implicit Objects: out

- For writing any data to the buffer, JSP provides an implicit object named ***out***.
- It is an object of JspWriter.

Servlet Code

```
PrintWriter out=
     response.getWriter();
response.setContentType
             ("text/html");
out.print("DIET");
```

JSP Code

```
<html>
<body>
<% out.print("DIET"); %>
</body>
</html>
```

Jsp Implicit Objects: request

- Instance of ***javax.servlet.http.HttpServletRequest*** object associated with the request.
- 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 from data, cookies, HTTP methods etc.

Jsp Implicit Objects: request

Resquest.html

```
<form action="welcome.jsp">
   Login:<input type="text" name="login">
         <input type="submit" value="next">
</form>
```

Welcome.jsp

```
hello,
<%
  out.println(request.getParameter("login"));
%>
```

Jsp Implicit Objects: response

- The response object is an instance of a ***javax.servlet.http.HttpServletResponse*** object.
- Through this object the JSP programmer can add new cookies or date stamps, HTTP status codes, redirect response to another resource, send error etc.

Jsp Implicit Objects: response

Response.html

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

Welcome.jsp

```
<%
response.sendRedirect("www.abc.ac.in");
%>
```

Jsp Implicit Objects: config

- config is an implicit object of type ***javax.servlet.ServletConfig.***
- This object can be used to get initialization parameter for a particular JSP page.

Config.html

```
<form action="MyConfig">
Login:<input type="text" name="login">
<input type="submit" value="sign_in">
</form>
```

Jsp Implicit Objects: config

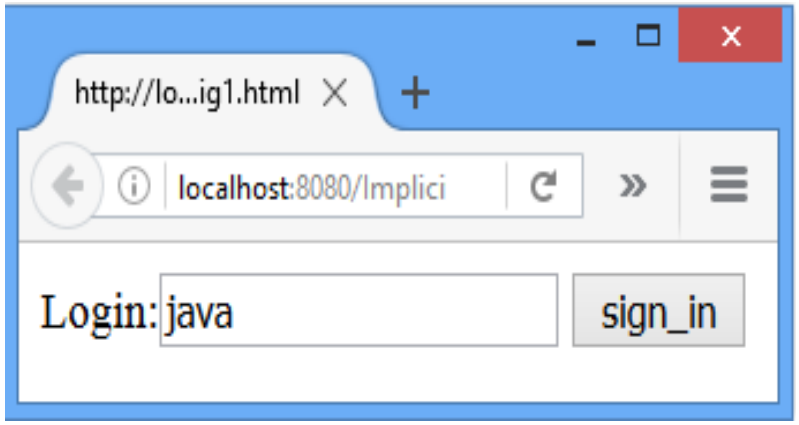
web.xml

```
<servlet>
    <servlet-name>MyConfig</servlet-name>
    <jsp-file>/MyConfig.jsp</jsp-file>
    <init-param>
        <param-name>College</param-name>
        <param-value>DIET</param-value>
    </init-param>
 </servlet>
 <servlet-mapping>
    <servlet-name>MyConfig</servlet-name>
    <url-pattern>/MyConfig</url-pattern>
</servlet-mapping>
```

Jsp Implicit Objects: config

MyConfig.jsp

```
<%
out.print("Welcome "+request.getParameter("login"));
String c_name=config.getInitParameter("College");
out.print("<p>College name is="+c_name+"</p>");
%>
```

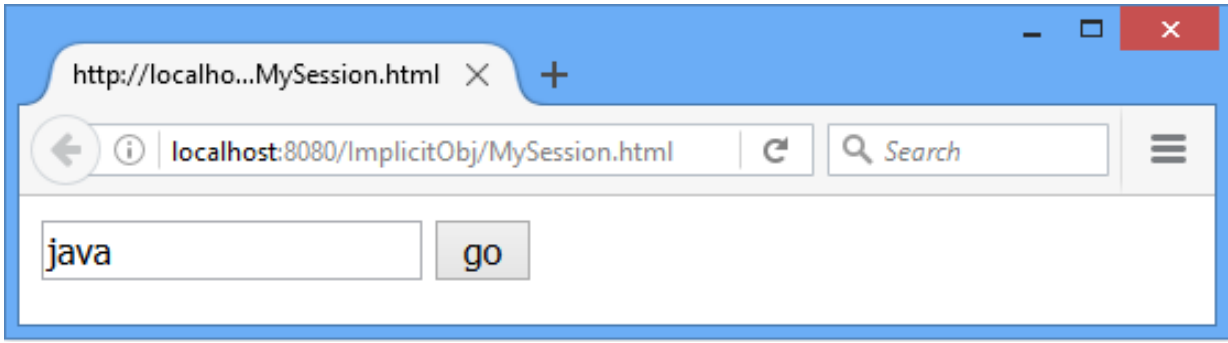


Jsp Implicit Objects: session

- In JSP, session is an implicit object of type ***javax.servlet.http.HttpSession*** .
- The Java developer can use this object to set, get or remove attribute or to get session information.

MySession.html

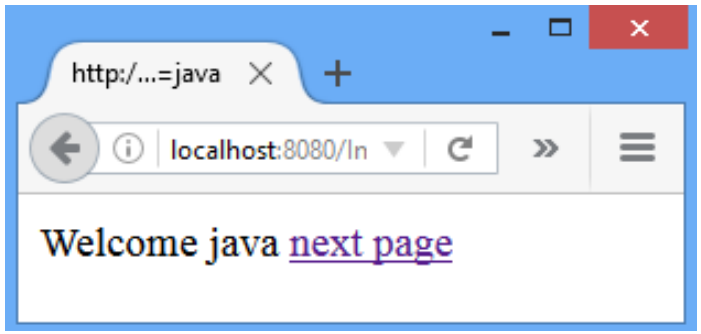
```
<form action="MySession1.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```



Jsp Implicit Objects: session

MySession1.jsp

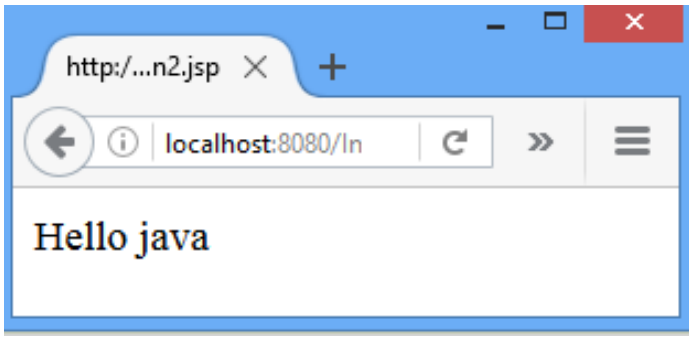
```
1. <html>
2. <body>
3. <%
4. String name=request.getParameter("uname");
5. out.print("Welcome "+name);
6. session.setAttribute("user",name);
7.  %>
8. <a href="MySession2.jsp">next page</a>
9. </body>
10.</html>
```



Jsp Implicit Objects: session

MySession2.jsp

```
1. <html>
2. <body>
3. <%
4. String name=
      (String)session.getAttribute("user");
5. out.print("Hello "+name);
6. %>
7. </body>
8. </html>
```



Jsp Implicit Objects: pageContext

- Instance of ***javax.servlet.jsp.PageContext***
- The pageContext object can be used to set, get or remove attribute.
- The PageContext class defines several fields, including PAGE_SCOPE, REQUEST_SCOPE, SESSION_SCOPE, and APPLICATION_SCOPE, which identify the four scopes.

Jsp Implicit Objects: pageContext

Context1.jsp

```
<% pageContext.setAttribute
      ("user","name",PageContext.APPLICATION_SCOPE);
%>
<a href="Context2.jsp">next page</a>
```

Context2.jsp

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

Jsp Implicit Objects: page

- This object is an actual reference to the instance of the page.
- It is an instance of ***java.lang.Object***
- Direct synonym for the **this** object.

Example: returns the name of generated servlet file

```
<%= page.getClass().getName() %>
```

Jsp Implicit Objects: application

- Instance of ***javax.servlet.ServletContext***
- The instance of ServletContext is created only once by the web container when application or project is deployed on the server.
- This object can be used to get initialization parameter from configuration file (web.xml).
- This initialization parameter can be used by all jsp pages.

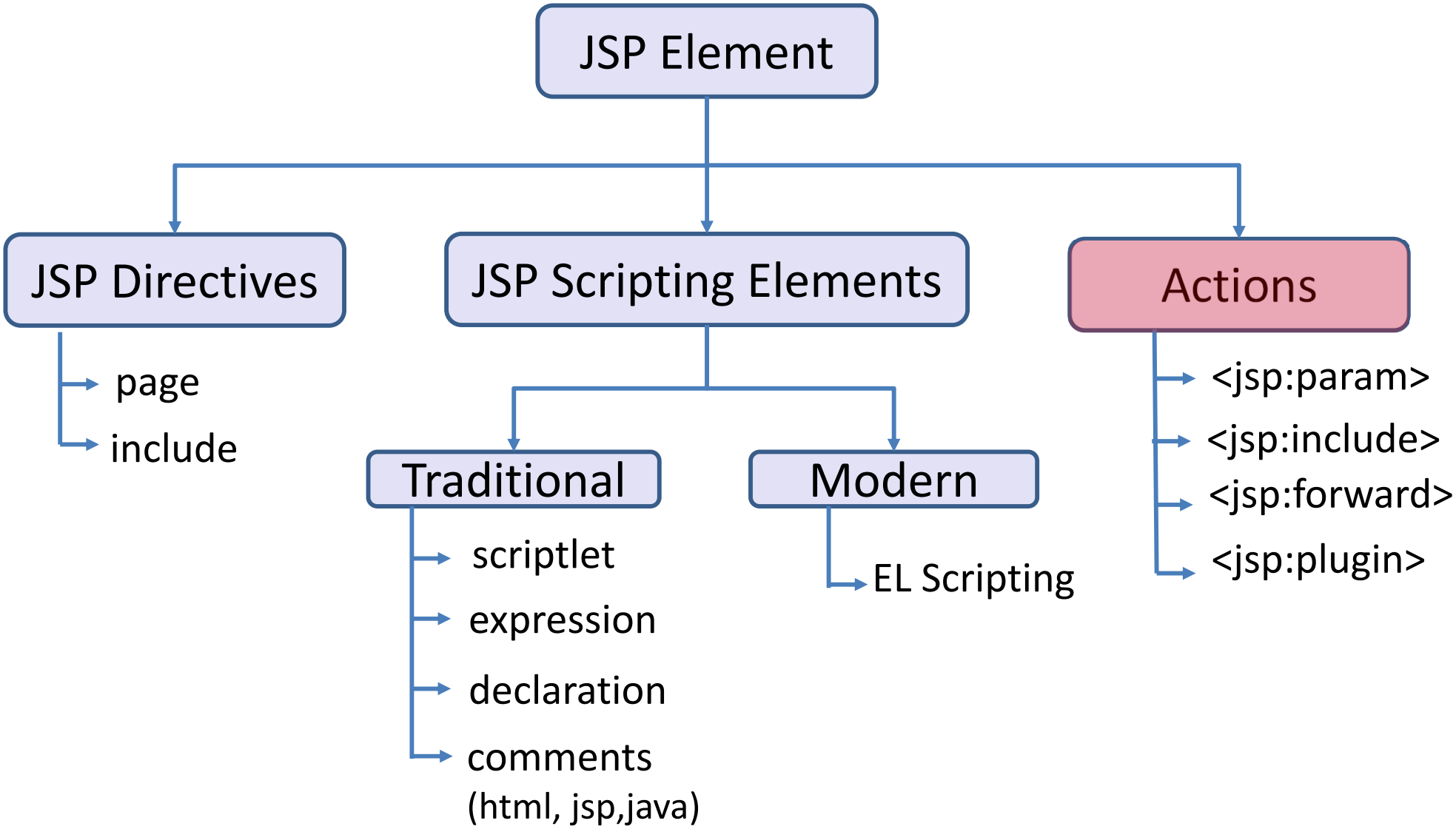
```
<%
//refers to context parameter of web.xml
    String driver=application.getInitParameter("name");
    out.print("name is="+driver);
%>
```

Jsp Implicit Objects: exception

- exception is an implicit object of type ***java.lang.Throwable*** class. This object can be used to print the exception.
- But it can only be used in error pages.

```
<%@ page isErrorPage="true" %>
<html>
<body>
Sorry following exception occured:
                                    <%=exception %>
</body>
</html>
```

JSP Elements



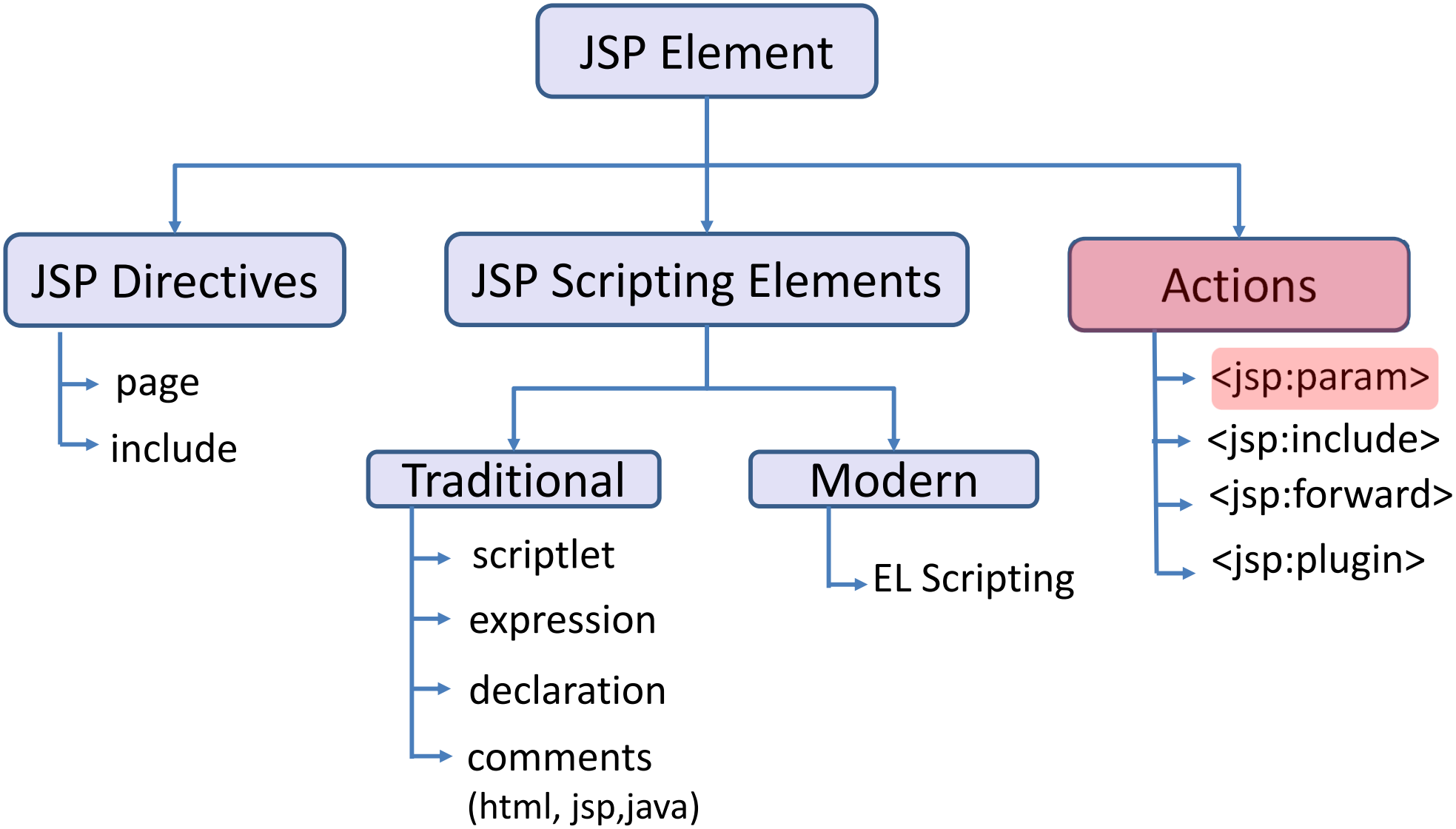
Actions

- JSP actions use constructs in XML syntax to control the behavior of the servlet engine.
- We can dynamically insert a file, reuse JavaBeans components, forward the user to another page, or generate HTML for the Java plugin.

Syntax

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

JSP Elements



<jsp:param>

- This action is useful for passing the parameters to other JSP action tags such as JSP include & JSP forward tag.
- This way new JSP pages can have access to those parameters using request object itself.

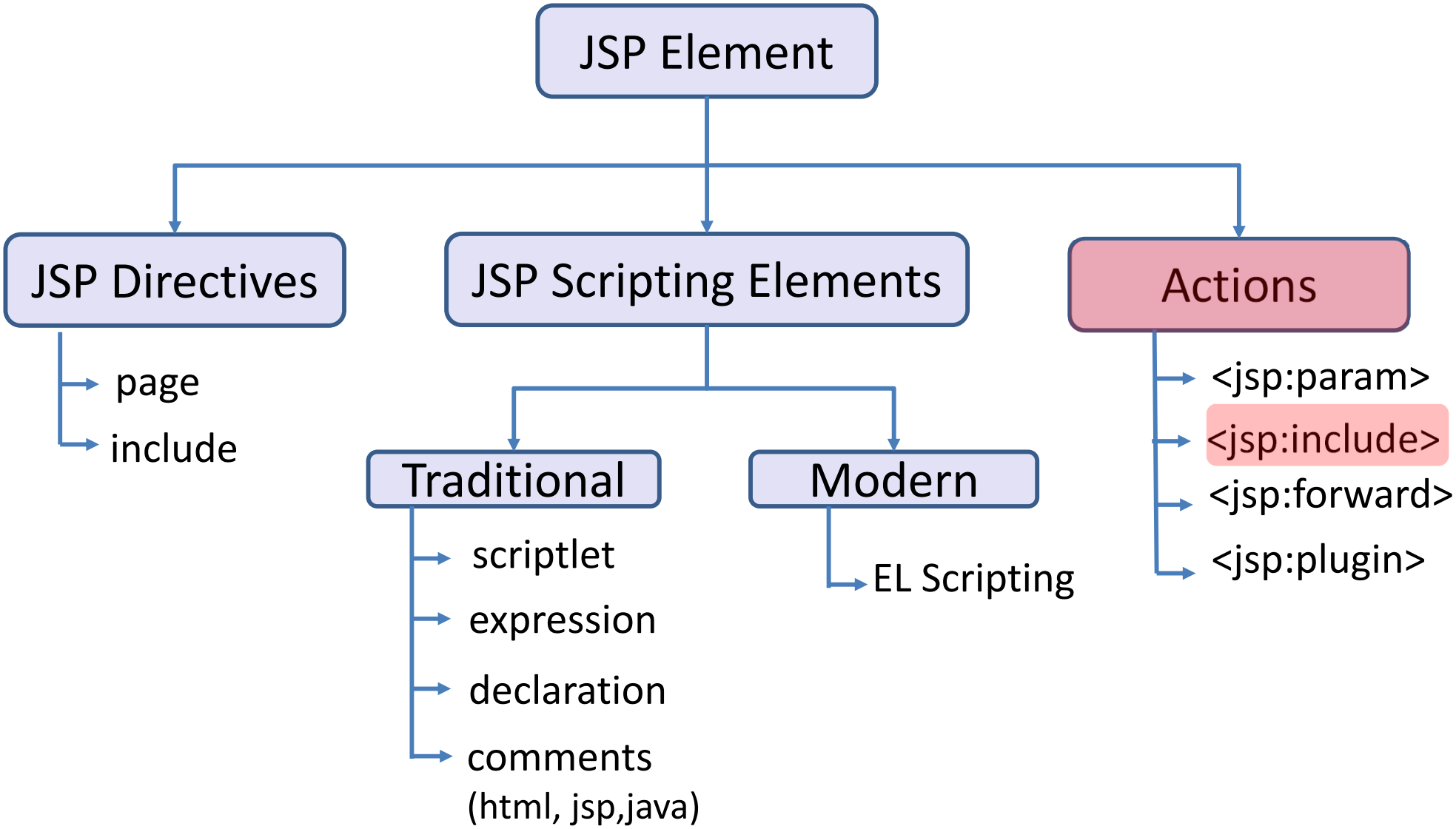
Syntax

```
<jsp:param name ="name" value="value"  />
```

Example

```
<jsp:param name ="date" value="12-02-2018" />
<jsp:param name ="time" value="10:15AM" />
<jsp:param name ="data" value="ABC" />
```

JSP Elements



<jsp:include>

- The **jsp:include action tag** is used to include the content of another resource it may be jsp, html or servlet.
- The jsp:include tag can be used to include static as well as dynamic pages

Attribute	**Description**
page	The relative URL of the page to be included.
flush	The boolean attribute determines whether the included resource has its buffer flushed before it is included. By default value is *false*.

<jsp:include>

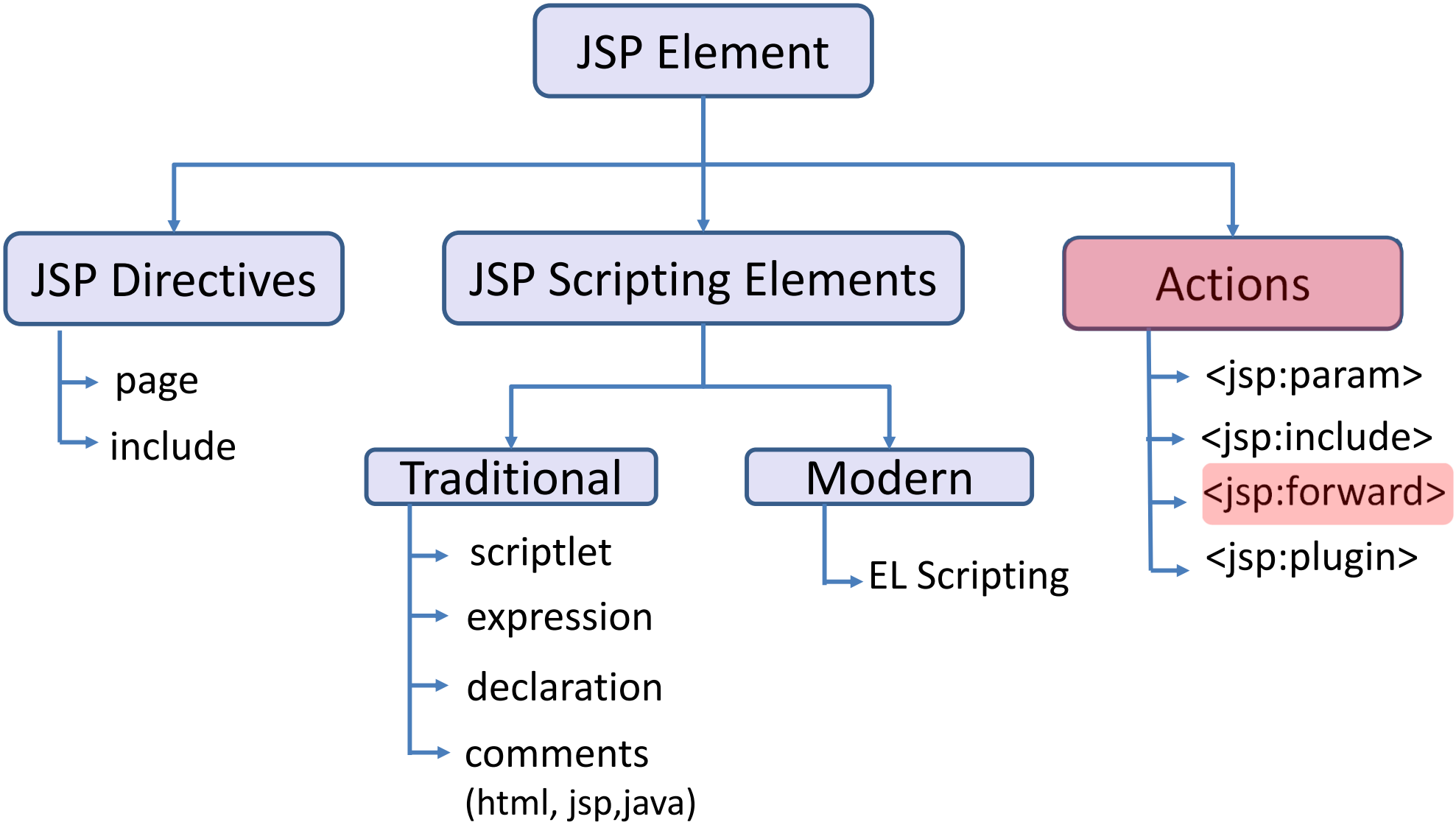
Syntax

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

Example

```
<jsp:include page="Action1.jsp" flush="true">
    <jsp:param name="roll_no1" value="401" />
</jsp:include>
```

JSP Elements



<jsp:forward>

- forwards the request and response to another resource.

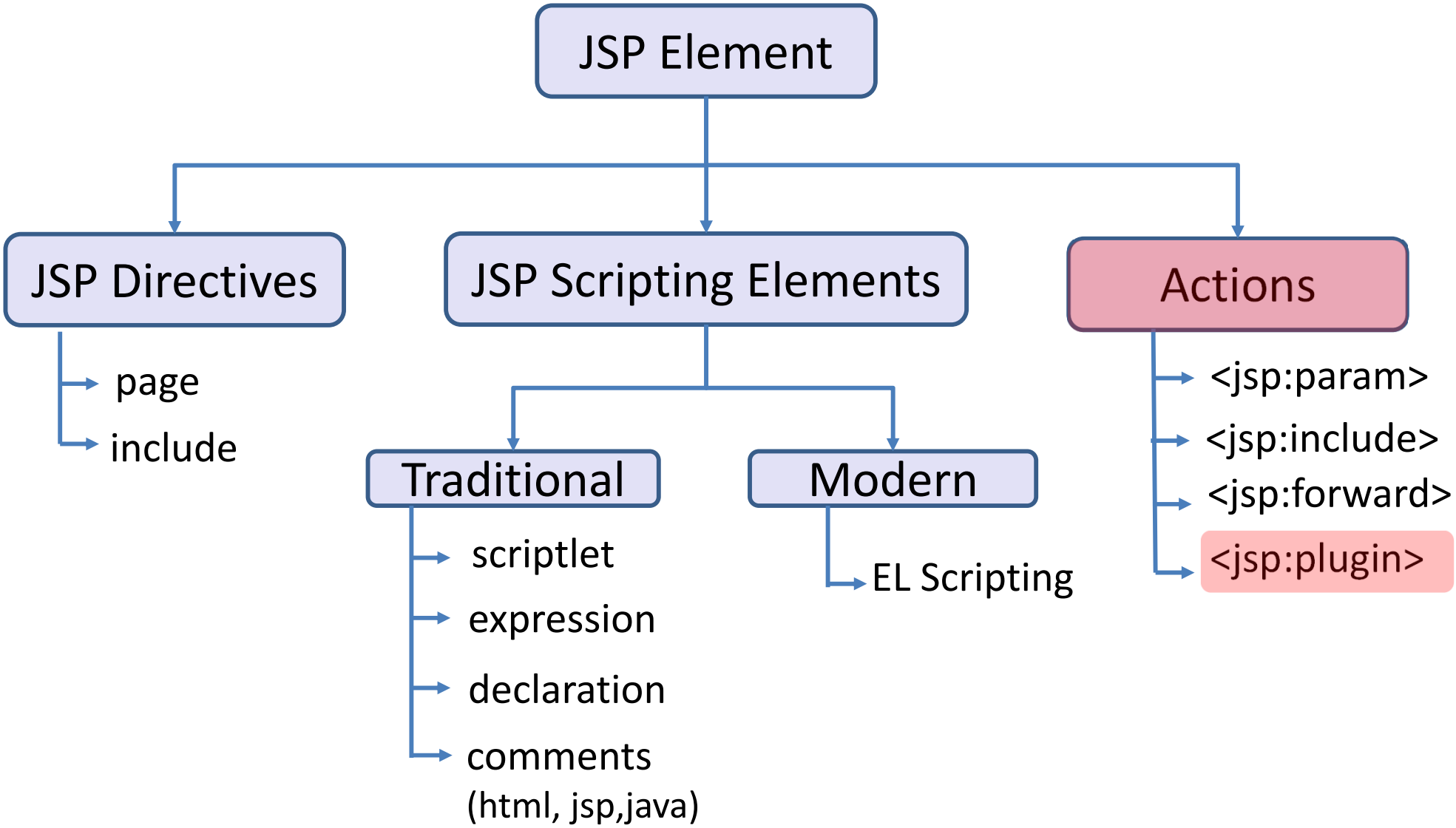
Syntax

```
<jsp:forward page="Relative URL" />
```

Example

```
<jsp:forward page="Action2.jsp">
    <jsp:param name="roll_no" value="301" />
</jsp:forward>
```

JSP Elements



<jsp:plugin>

- This tag is used when there is a need of a plugin to run a Bean class or an Applet.
- The <jsp:plugin> action tag is used to embed applet in the jsp file.
- The <jsp:plugin> action tag downloads plugin at client side to execute an applet or bean.

Syntax

```
<jsp:plugin type="applet|bean"
            code="nameOfClassFile"
            codebase= "URL"
/>
```

<jsp:plugin>

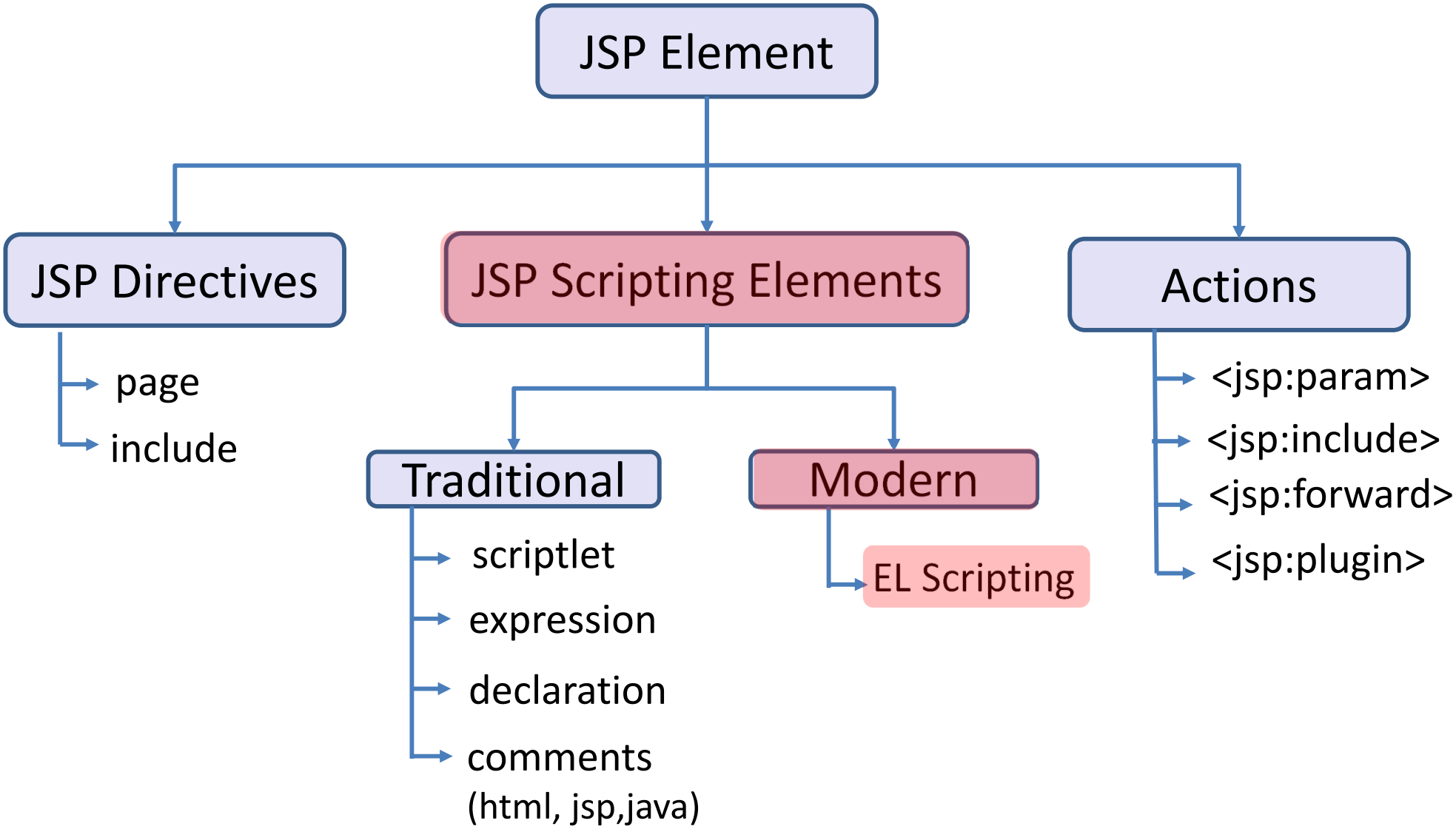
MyApplet.java

```
import java.applet.*;
import java.awt.*;
public class MyApplet extends Applet {
   public void paint(Graphics g) {
      g.drawString("Welcome in Java Applet.",40,20);
}}
```

MyPlugin.jsp

```
<html> <body>
        <jsp:plugin
           type="applet"
           code="MyApplet.class"
           codebase="/JSPClass/MyApplet"/>
</body></html>
```

JSP Elements



EL Scripting

- Expression Language(EL) Scripting.
- It is the newly added feature in JSP technology version 2.0.
- **The purpose of EL is to produce script less JSP pages.**

Syntax

```
${expr}
```

Example

EL	output
${a=10}	10
${10+20}	30
${20*2}	40
${10==20}	false
${'a'<'b'}	true

EL Implicit Object

pageScope	It is used to access the value of any variable which is set in the Page scope
requestScope	It is used to access the value of any variable which is set in the Request scope.
sessionScope	It is used to access the value of any variable which is set in the Session scope
applicationScope	It is used to access the value of any variable which is set in the Application scope
pageContext	It represents the PageContext object.

EL Implicit Object

param	Map a request parameter name to a single value
paramValues	Map a request parameter name to corresponding array of string values.
header	Map containing header names and single string values.
headerValues	Map containing header names to corresponding array of string values.
cookie	Map containing cookie names and single string values.

EL Implicit Object

- An expression can be mixed with static text/values and can also be combined with other expressions

Example

Impicit object

Parameter Name

```
${param.name}

${sessionScope.user}
```

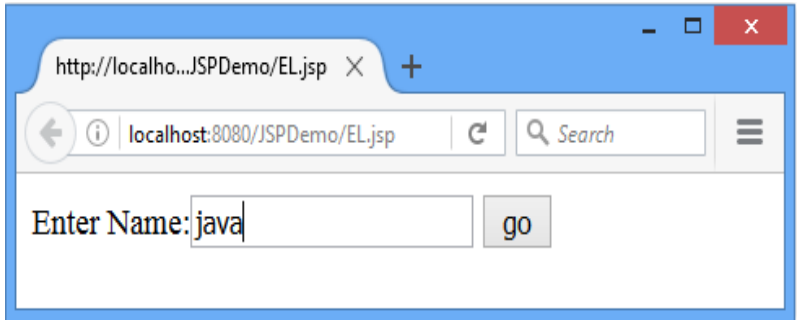
Impicit object

Parameter Name

EL Implicit Object: Example

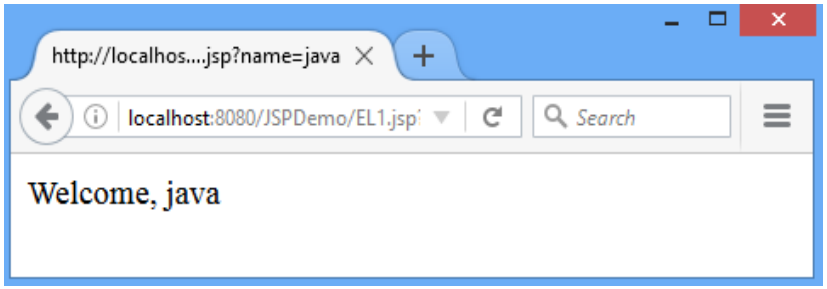
EL.jsp

```
1.  <form action="EL1.jsp">
2.   Enter Name:<input type="text"
                        name="name" >
3.   <input type="submit"
             value="go">
4.  </form>
```



EL1.jsp

```
1. Welcome, ${ param.name }
```



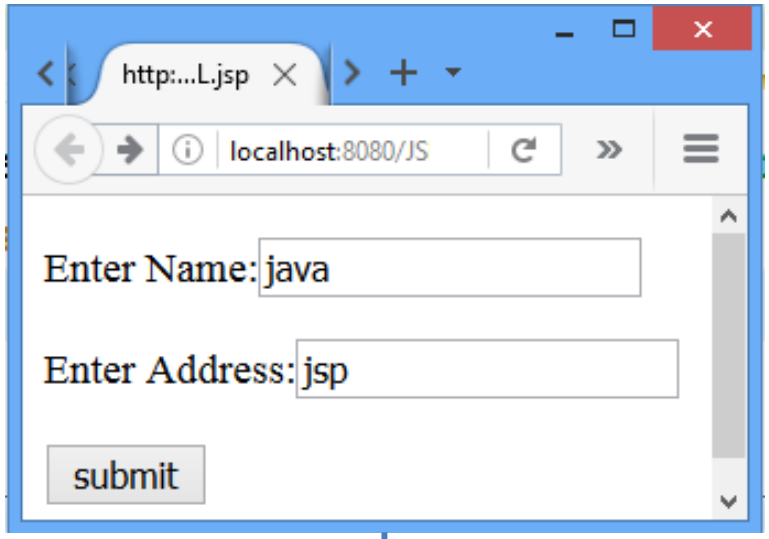
EL Implicit Object: Example

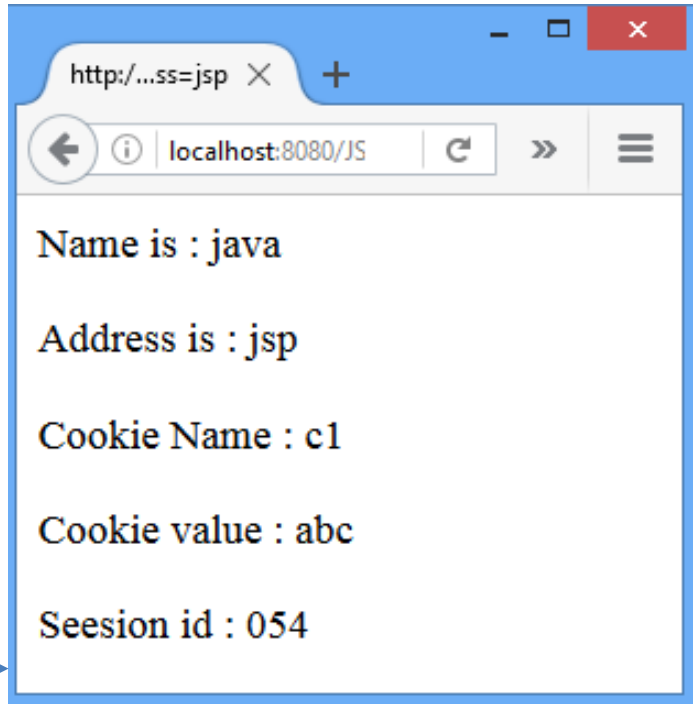
```
1. <form action="EL2.jsp">
2. <% Cookie ck=new Cookie("c1","abc");
3.    response.addCookie(ck);
4.    session.setAttribute("sid","054");  //for session
5. %>
6. Enter Name:<input type="text" name="name" >
7. Enter Address:<input type="text" name="address" >
8. <input type="submit" value="submit">
9. </form>
```

EL Implicit Object: Example

```
1. <p>Name is :          ${param.name}</p>
2. <p>Address is :       ${param.address}</p>
3. <p>Cookie Name :      ${cookie.c1.name}</p>
4. <p>Cookie value :     ${cookie.c1.value}</p>
5. <p>Session id :       ${sessionScope.sid}</p>
```

EL.jsp





EL2.jsp

JSP EL Operator

JSP EL Arithmetic Operators

Arithmetic operators are provided for simple calculations in EL expressions. They are +, -, *, / or div, % or mod.

JSP EL Logical Operators

They are && (and), || (or) and ! (not).

JSP EL Relational Operators

They are == (eq), != (ne), < (lt), > (gt), <= (le) and >= (ge).

JSP EL Important Points

- EL expressions are always within curly braces prefixed with $ sign, for example ${expr}
- We can disable EL expression in JSP by setting JSP page directive isELIgnored attribute value to TRUE.

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

- JSP EL can be used to get attributes, header, cookies, init params etc, but we can’t set the values.
- JSP EL implicit objects are different from JSP implicit objects except pageContext
- JSP EL is NULL friendly, if given attribute is not found or expression returns null, it doesn’t throw any exception.

Exception Handling in JSP

Exception Handling in JSP

JSP provide 3 different ways to perform exception handling:

1. Using simple **try...catch** block.
2. Using **isErrorPage** and **errorPage** attribute of **page directive**.
3. Using <**error-page**> tag in **Deployment Descriptor.**

Exception Handling: try/catch block

Using try...catch block is just like how it is used in Core Java.

Example

```
<html>
<body>
  <%
   try{
      int i = 100;
      i = i / 0;
      out.println("The answer is " + i);
   }
   catch (Exception e){
      out.println("An exception occurred: " + e.getMessage());
   }
  %>
</body>
</html>
```

Exception Handling: Error Page

Exception Handling using **isErrorPage** and **errorPage** attribute of **page directive**.

MyJSP.jsp

```
<%@page errorPage=
            MyErrPage.jsp"%>
```

MyErrorPage.jsp

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

Exception Handling: Error Page

Exception Handling using **isErrorPage** and **errorPage** attribute of **page directive**.

errorPage
1.jsp

```
<%@page  errorPage= "2.jsp" %>
<%     int i=10;
        i=i/0; %>
```

If Exception occurs in 1.jsp
then forward req to 2.jsp

```
<%@page isErrorPage="true" %>
<html> <body>
An Exception had occured
<%  out.println(exception.toString());%>
</body> </html>
```

This attribute designates
.jsp page as **ERROR PAGE**

2.jsp

Exception Handling in JSP: web.xml

- Declaring error page in Deployment Descriptor for entire web application.
- Specify Exception inside

  `<error-page>` tag in the Deployment Descriptor.
- We can even configure different error pages for different exception types, or HTTP error code type(503, 500 etc).

Exception Handling in JSP: web.xml

Declaring an error page for all type of exception

```
<error-page>
      <exception-type>
            java.lang.Throwable
      </exception-type>
      <location>/error.jsp</location>
</error-page>
```

Exception Handling in JSP: web.xml

Declaring an error page for more detailed exception

```
<error-page>
      <exception-type>
            java.lang.ArithmeticException
      </exception-type>
<location>/error.jsp</location>
</error-page>
```

Exception Handling in JSP: web.xml

```
<error-page>
      <error-code>404</error-code>
      <location>/error.jsp</location>
</error-page>

<error-page>
      <error-code>500</error-code>
      <location>/error.jsp</location>
</error-page>
```

Exception Handling in JSP: web.xml

- This approach is better because we don't need to specify the errorPage attribute in each jsp page.
- Specifying the single entry in the web.xml file will handle the exception.
- In this case, either specify ***exception-type*** or ***error-code*** with the location element.

JSP with JDBC

```
1. <%@page import="java.sql.*" %>
2. <%
3. Class.forName("com.mysql.jdbc.Driver");
4. Connection con=DriverManager.getConnection(
5.            "jdbc:mysql://localhost:3306/GTU","root","root");
6. Statement stmt=con.createStatement();
7. ResultSet rs=stmt.executeQuery("select * from diet");
8. while(rs.next())  {
9.     out.println("<p>"+rs.getString(1));
10.            out.println(rs.getString(2));
11.            out.println(rs.getString(3)+"</p>");
12.}
13.con.close();
14.%>
```

JSP Session and Cookies Handling

JSP Cookie Handling

- Cookies are text files stored on the client computer and they are kept for various information tracking purpose.
- JSP transparently supports HTTP cookies using underlying servlet technology.

JSP Cookie Handling

```
<% Cookie cookie = new Cookie("c1","MyCookie1");
cookie.setMaxAge(60 * 60);
response.addCookie(cookie);
%>
<html><body>
   <a href="Cookie2.jsp">Click here</a>
</body></html>
```

Cookie1.jsp

```
<%      Cookie[] c2 = request.getCookies();
        for(int i = 0; i < c2.length; i++) {
            out.print("<p>"+c2[i].getName()+"    "+
                                c2[i].getValue()+"</p>");
        }
%>
```

Cookie2.jsp

JSP Session Handling

- In JSP, session is an implicit object of type HttpSession.
- The Java developer can use this object to set, get or remove attribute or to get session information.
- In Page Directive, session attribute indicates whether or not the JSP page uses HTTP sessions.

```
<%@ page session="true" %> //By default it is true
```

JSP Session Handling

Session1.jsp

```
<%@page session="true" %>
<% session.setAttribute("s1","DIET");%>
<html>
    <body>
        <a href="Session2.jsp">nextPage</a>
    </body>
</html>
```

Session2.jsp

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

<% String str=(String)session.getAttribute("s1");

out.println("session="+str);%>
```

JSP - Standard Tag Library (JSTL)

JSP - Standard Tag Library (JSTL)

- The JSP Standard Tag Library (JSTL) represents a set of tags to simplify the JSP development.

Advantages 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.

*For creating JSTL application, you need to load **jstl.jar** file.*

JSP - Standard Tag Library (JSTL)

Tag Library	Function	URI	prefix
Core tag library	Variable support Flow Control Iterator URL management Miscellaneous	http://java.sun.com/jsp/jstl/core	**c**
Functions Library	Collection length String manipulation	http://java.sun.com/jsp/jstl/functions	**fn**
Internationalization tag library	Message formatting Number and date formatting	http://java.sun.com/jsp/jstl/fmt	**fmt**
SQL tag library	Database manipulation	http://java.sun.com/jsp/jstl/sql	**sql**
XML tag library	Flow control Transformation	http://java.sun.com/jsp/jstl/xml	**x**

JSTL: Core tag library

- The core group of tags are the most frequently used JSTL tags.
- The JSTL core tag provides variable support, URL management, flow control etc.

Syntax to include JSTL Core library in your JSP:

```
<%@ taglib prefix="c"
        uri="http://java.sun.com/jsp/jstl/core"
%>
```

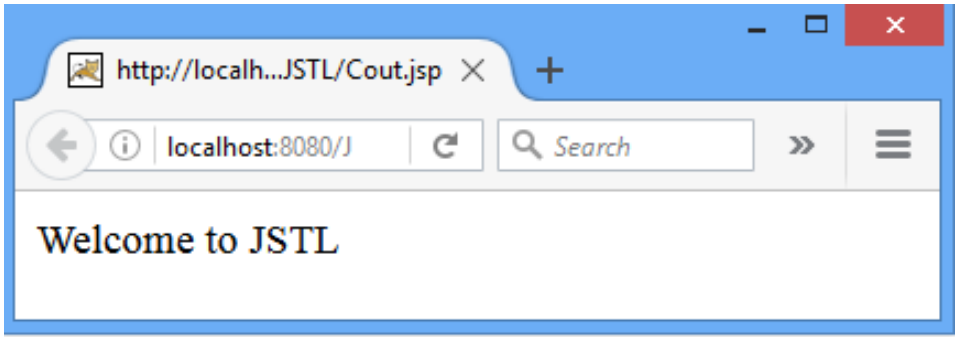
JSTL: Core tag library

Tags	Description
c:out	It display the result of an expression, similar to the way <%=...%> tag work.
c:import	It Retrives relative or an absolute URL and display the contents to either a String in 'var',a Reader in 'varReader' or the page.
c:set	It sets the result of an expression under evaluation in a 'scope' variable.
c:remove	It is used for removing the specified scoped variable from a particular scope.
c:catch	It is used for Catches any Throwable exceptions that occurs in the body.
c:if	It is conditional tag used for testing the condition and display the body content only if the expression evaluates is true.
c:choose, c:when, c:otherwise	It is the simple conditional tag that includes its body content if the evaluated condition is true.
c:forEach	It is the basic iteration tag. It repeats the nested body content for fixed number of times or over collection.
c:forTokens	It iterates over tokens which is separated by the supplied delimeters.
c:param	It adds a parameter in a containing 'import' tag's URL.
c:redirect	It redirects the browser to a new URL and supports the context-relative URLs.
c:url	It creates a URL with optional query parameters.

JSTL: Core tag library

1	c:out	It display the result of an expression, similar to the way <%=...%> tag work.

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



JSTL: Core tag library

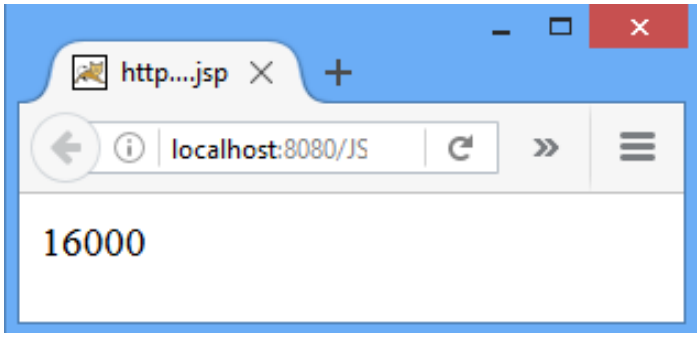
2	c:import	It is similar to jsp 'include', with an additional feature of including the content of any resource either within server or outside the server.

```
1. <%@ taglib uri=
      "http://java.sun.com/jsp/jstl/core"
      prefix="c" %>
2. <html>
3. <body>
4. <c:import var="data"
      url="http://www.abc.ac.in"/>
5. <c:out value="${data}"/>
6. </body>
7. </html>
```

JSTL: Core tag library

3	c:set	It is used to set the result of an expression evaluated in a 'scope'. This tag is similar to jsp:setProperty action tag.

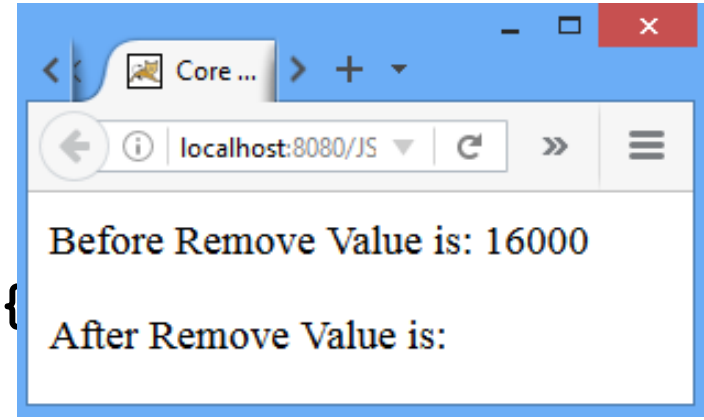
```
1. <%@ taglib uri=
      "http://java.sun.com/jsp/jstl/core"
      prefix="c" %>
2. <html>
3. <body>
4. <c:set var="Income" scope="session"
                                value="${4000*4}"/>
5. <c:out value="${Income}"/>
6. </body>
7. </html>
```



JSTL: Core tag library

4	c:remove	It is used for removing the specified scoped variable from a particular scope

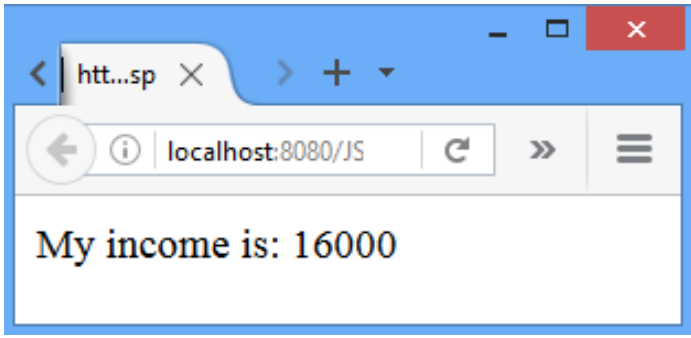
```
1. <%@ taglib uri="http://java.sun.com/jsp/jstl/core"
   prefix="c" %>
2. <c:set var="income" scope="session"
                                    value="${4000*4}"/>
3. <p>Before Remove Value is: <c:out
                                    value="${income}"/></p>
4. <c:remove var="income"/>
5. <p>After Remove Value is: <c:out
                                    value="${
```



JSTL: Core tag library

5	c:if	It is conditional tag used for testing the condition and display the body content only if the expression evaluates is true.

```
1. <%@ taglib uri="http://java.sun.com/jsp/jstl/core"
   prefix="c" %>
2. <c:set var="income" scope="session"
   value="${4000*4}"/>
3. <c:if test="${income > 8000}">
4.     <p>My income is: <c:out value="${income}"/></p>
5. </c:if>
```

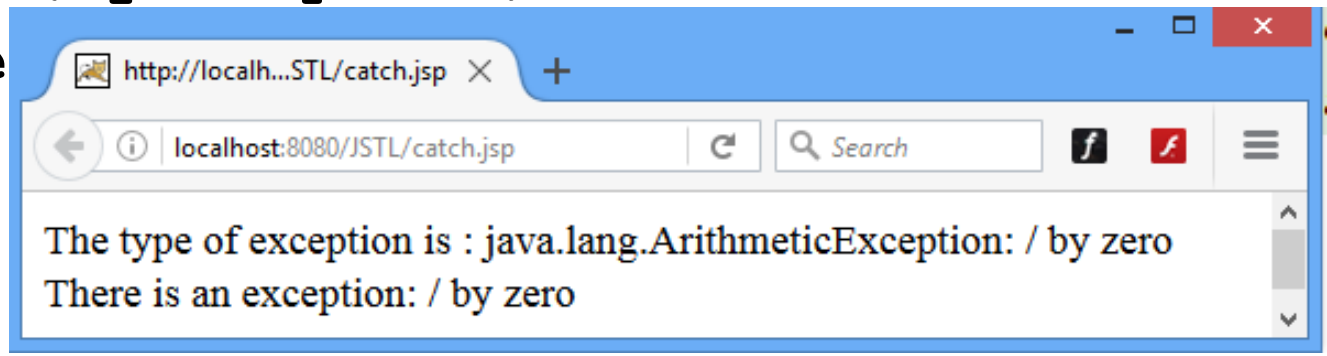


JSTL: Core tag library

6	c:catch	It is used for catching any Throwable exceptions that occurs in the body and optionally exposes it.

```
1. <%@ taglib
   uri="http://java.sun.com/jsp/jstl/core"
   prefix="c" %>
2. <c:catch var ="MyException">
3.     <% int x = 2/0;%>
4. </c:catch>
5. <c:if test = "${MyException != null}">
6.     <p>The type of exception is :
                    ${MyException}<br />
7.     There is an e

8. </c:if>
```



JSTL: Core tag library

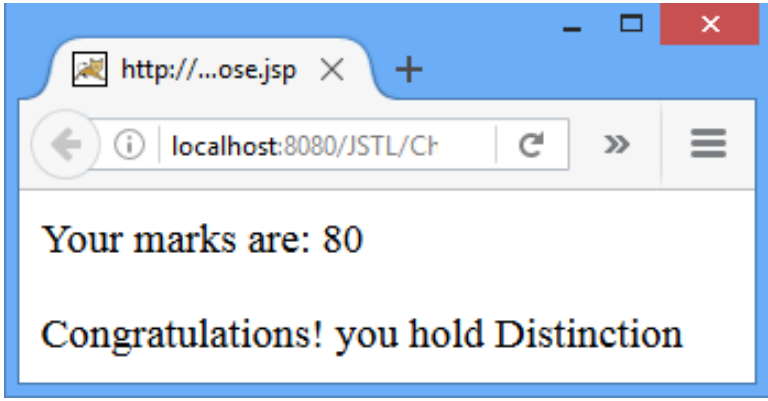
7	c:choose	It is a conditional tag that establish a context for mutually exclusive conditional operations. It works like a Java switch statement in which we choose between a numbers of alternatives.
	c:when	It is subtag of <choose > that will include its body if the condition evaluated be 'true'.
	c:otherwise	It is also subtag of < choose > it follows <when> tags and runs only if all the prior condition evaluated is 'false'.

*The **<c:when>** and **<c:otherwise>** works like if-else statement.*
*But it must be placed inside **<c:choose tag>**.*

JSTL: Core tag library

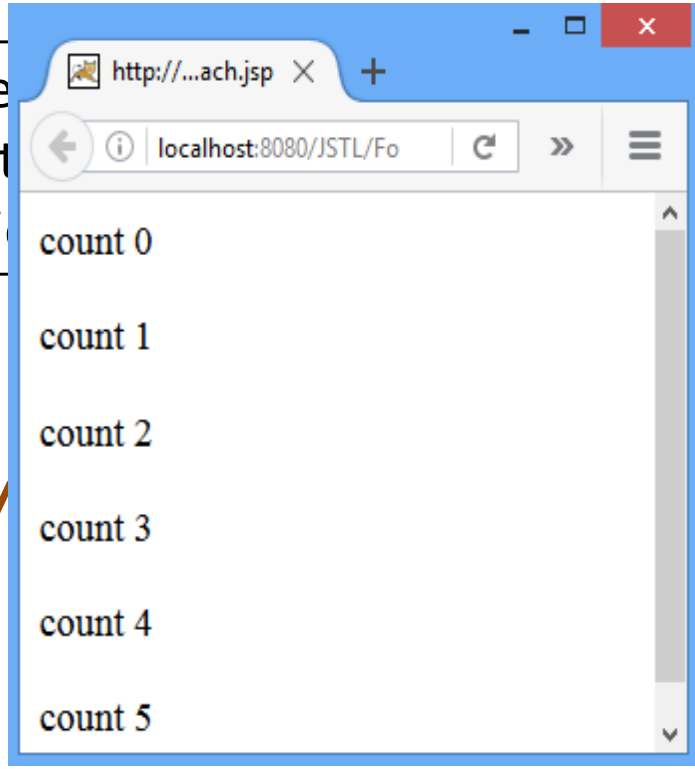
Choose.jsp

```
1. <%@ taglib uri="http://java.sun.com/jsp/jstl/core"
   prefix="c" %>
2. <c:set var="marks" scope="session" value="${80}"/>
3. <p>Your marks are: <c:out value="${marks}"/></p>
4. <c:choose>
5.     <c:when test="${marks <= 35}"> Sorry! you are fail.
6.     </c:when>
7.     <c:when test="${marks > 75}">
8.         Congratulations! you hold Distinction
9.     </c:when>
10.    <c:otherwise>
11.        Sorry! Result is unavailable
12.    </c:otherwise>
13.</c:choose>
```



JSTL: Core tag library

8	c:forEach	It is an iteration tag used for repeating the fixed number of times. The < c:for each > t used tag because it iterates over a collecti

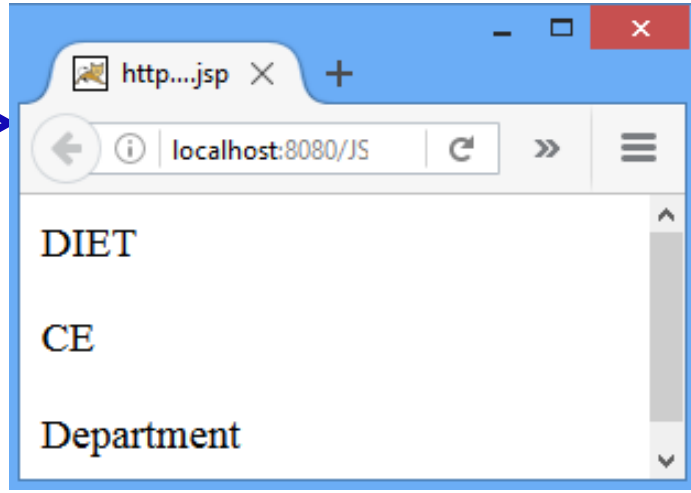



```
1. <%@ taglib uri=
   "http://java.sun.com/jsp/jstl/
   prefix="c" %>
2. <html>  <body>
3. <c:forEach var="i" begin="0" end="5">
4.     count <c:out value="${i}"/><p>
5. </c:forEach>
6. </body>  </html>
```

JSTL: Core tag library

9	c:forTokens	It iterates over tokens which is separated by the supplied delimeters.

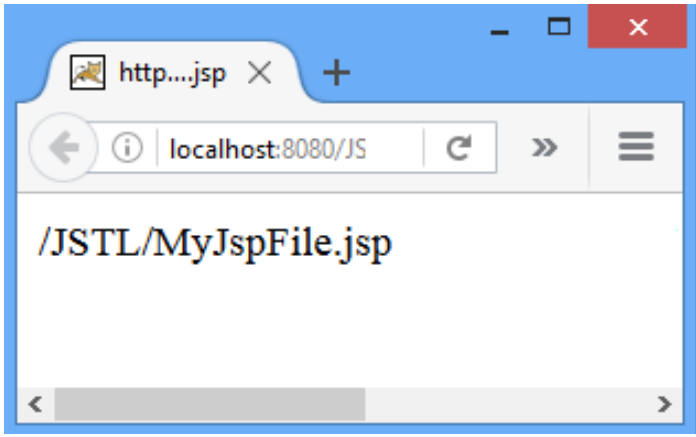
```
1. <%@ taglib uri=
      "http://java.sun.com/jsp/jstl/core"
                                          prefix="c"
   %>
2.  <c:forTokens items="DIET-CE-Department"
   delims="-" var="name">
3.     <c:out value="${name}"/><p>
4. </c:forTokens>
```



JSTL: Core tag library

10	c:url	This tag creates a URL with optional query parameter. It is used for url encoding or url formatting. This tag automatically performs the URL rewriting operation.

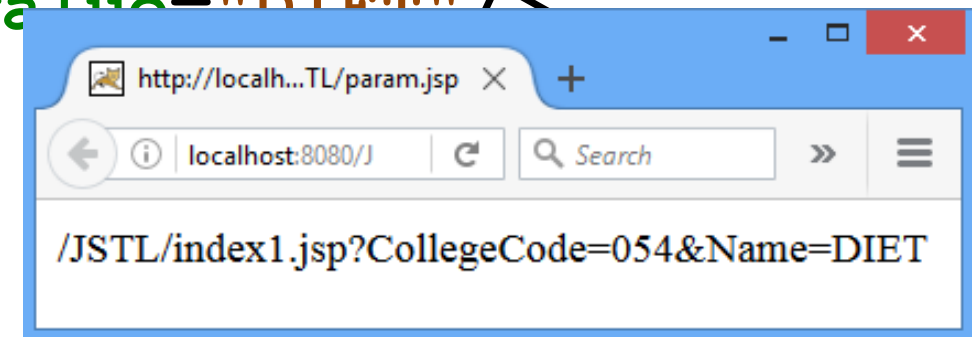
```
<%@ taglib uri=
      "http://java.sun.com/jsp/jstl/core"
                                          prefix="c"
%>
<c:url value="/MyJspFile.jsp"/>
```



JSTL: Core tag library

11	c:param	It allow the proper URL request parameter to be specified within URL and it automatically perform any necessary URL encoding.

```
1. <%@ taglib uri=
      "http://java.sun.com/jsp/jstl/core"
                                        prefix="c"
   %>
2. <c:url value="/index1.jsp"
   var="completeURL">
3.  <c:param name="CollegeCode" value="054"/>
4.  <c:param name="Name" value="DIET"/>
5. </c:url>
6. ${completeURL}
```



JSTL: Core tag library

12	c:redirect	tag redirects the browser to a new URL. It is used for redirecting the browser to an alternate URL by using automatic URL rewriting.

```
1. <%@ taglib
   uri="http://java.sun.com/jsp/jstl/core"
   prefix="c" %>
2. <c:redirect url="http://darshan.ac.in"/>
```

JSP - Standard Tag Library (JSTL)

Tag Library	Function	URI	prefix
Core tag library	Variable support Flow Control Iterator URL management Miscellaneous	http://java.sun.com/jsp/jstl/core	c
Functions Library	Collection length String manipulation	http://java.sun.com/jsp/jstl/functions	fn
Internationalization tag library	Message formatting Number and date formatting	http://java.sun.com/jsp/jstl/fmt	fmt
SQL tag library	Database manipulation	http://java.sun.com/jsp/jstl/sql	sql
XML tag library	Flow control Transformation	http://java.sun.com/jsp/jstl/xml	x

JSTL -Function Tags List

The JSTL function provides a number of standard functions, most of these functions are common *string manipulation functions*.

Syntax:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions"
                                                prefix="fn" %>
```

JSTL -Function Tags List

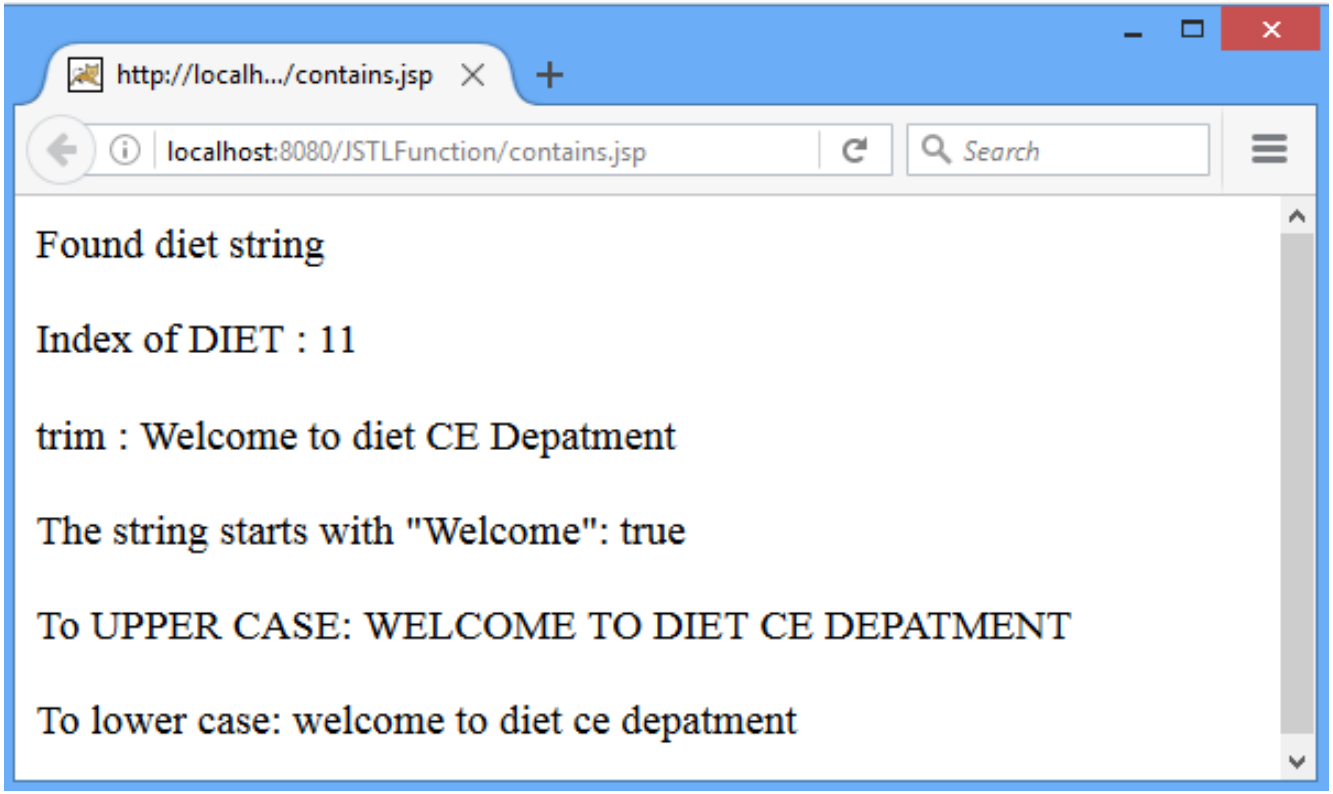
fn:contains	It is used to test if an input string containing the specified substring in a program.
fn:containsIgnoreCase	It is used to test if an input string contains the specified substring as a case insensitive way.
fn:endsWith	It is used to test if an input string ends with the specified suffix.
fn:startsWith	It is used for checking whether the given string is started with a particular string value.
fn:toLowerCase	It converts all the characters of a string to lower case.
fn:toUpperCase	It converts all the characters of a string to upper case.
fn:length	It returns the number of characters inside a string, or the number of items in a collection.
fn:indexOf	It returns an index within a string of first occurrence of a specified substring.
fn:substring	It returns the subset of a string according to the given start and end position.
fn:replace	It replaces all the occurrence of a string with another string sequence.
fn:trim	It removes the blank spaces from both the ends of a string.

JSTL -Function Tags List

Function.jsp

```
1.  <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
2.  <%@ taglib uri="http://java.sun.com/jsp/jstl/functions"
                                                    prefix="fn" %>
3.  <c:set var="String1" value="  Welcome to diet CE Department  "/>
4.  <c:if test="${fn:contains(String1, 'diet')}">
5.      <p>Found diet string<p>
6.      <p>Index of DIET : ${fn:indexOf(String1, "diet")}</p>
7.      <c:set var="str2" value="${fn:trim(String1)}" />
8.      <p>trim : ${str2}</p>
9.      The string starts with "Welcome":
                        ${fn:startsWith(String1, 'Welcome')}
10.     <p>To UPPER CASE: ${fn:toUpperCase(String1)}</p>
11.     <p>To lower case: ${fn:toLowerCase(String1)}</p>
12. </c:if>
```

JSTL -Function Tags List



JSP - Standard Tag Library (JSTL)

Tag Library	Function	URI	prefix
Core tag library	Variable support Flow Control Iterator URL management Miscellaneous	http://java.sun.com/jsp/jstl/core	c
Functions Library	Collection length String manipulation	http://java.sun.com/jsp/jstl/functions	fn
Internationalization tag library	Message formatting Number and date formatting	http://java.sun.com/jsp/jstl/fmt	fmt
SQL tag library	Database manipulation	http://java.sun.com/jsp/jstl/sql	sql
XML tag library	Flow control Transformation	http://java.sun.com/jsp/jstl/xml	x

JSTL-Formatting tags

The formatting tags provide support for message formatting, number and date formatting etc.

Syntax

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"
%>
```

JSTL-Formatting tags

Formatting Tags	**Descriptions**
fmt:parseNumber	It is used to Parses the string representation of a currency, percentage or number.
fmt:formatNumber	It is used to format the numerical value with specific format or precision.

JSTL-Formatting tags

Number.jsp

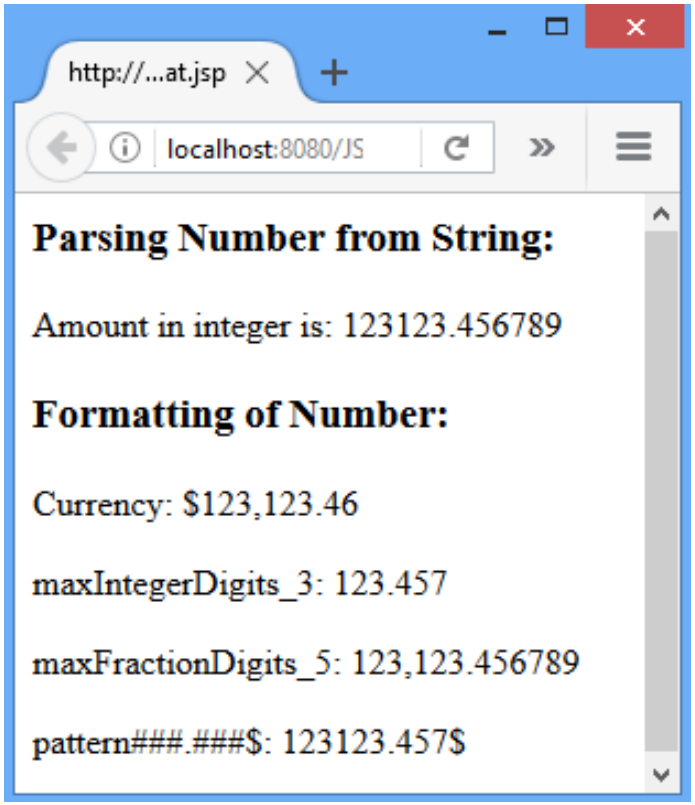
```
1. <%@ taglib prefix="c"
              uri="http://java.sun.com/jsp/jstl/core" %>
2. <%@ taglib prefix="fmt"
              uri="http://java.sun.com/jsp/jstl/fmt" %>
3. <c:set var="Amount" value="123123.456789" />
4. <h3>Parsing Number from String:</h3>
5. <fmt:parseNumber var="j" type="number"
                                        value="${Amount}" />
6. <p>Amount in parsed integer is: <c:out value="${j}"
   /></p>
```

JSTL-Formatting tags

Number.jsp

```
7. <h3>Formatting of Number:</h3>
8. <p> Currency:
9. <fmt:formatNumber value="${Amount}" type="currency"
                                                    /></p>
10.<p>maxIntegerDigits_3:
11.<fmt:formatNumber type="number" maxIntegerDigits="3"
                                  value="${Amount}" /></p>
12.<p>maxFractionDigits_5:
13.<fmt:formatNumber type="number" maxFractionDigits="6"
                                  value="${Amount}" /></p>
14.<p>pattern###.###$:
15.<fmt:formatNumber type="number" pattern="###.###$"
   value="${Amount}" /></p>
```

JSTL-Formatting tags



JSTL-Formatting tags

fmt:formatDate	It formats the time and/or date using the supplied pattern and styles.
fmt:parseDate	It parses the string representation of a time and date.
fmt:setTimeZone	It stores the time zone inside a time zone configuration variable.
fmt:timeZone	It specifies a parsing action nested in its body or the time zone for any time formatting.
fmt:message	It display an internationalized message.

JSTL-Formatting tags

TimeZone.jsp

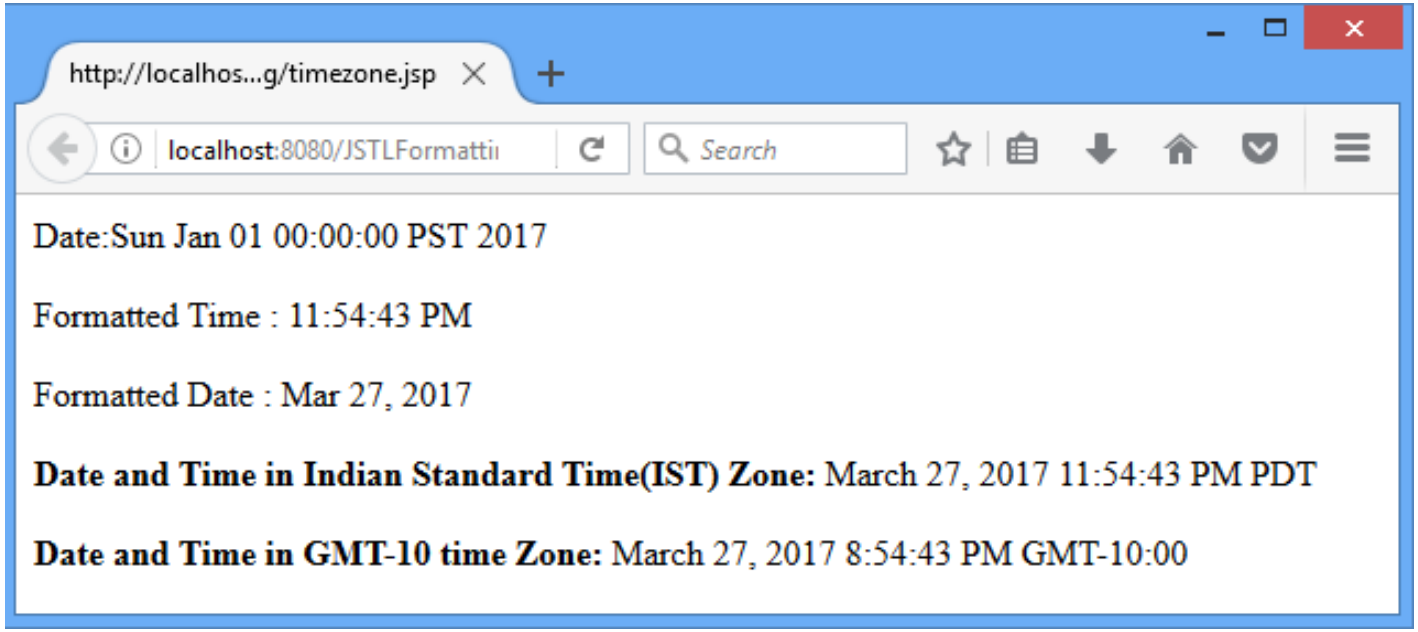
```
1. <%@ taglib prefix="c"
                    uri="http://java.sun.com/jsp/jstl/core" %>
2. <%@ taglib prefix="fmt"
                    uri="http://java.sun.com/jsp/jstl/fmt" %>
3. <c:set var="date" value="01-01-2019" />
4. <fmt:parseDate value="${date}" var="parsedDate"
                                        pattern="dd-MM-yyyy" />
5. <p>Date:<c:out value="${parsedDate}" /></p>
6. <c:set var="Date" value="<%=new java.util.Date()%>" />
7. <p> Formatted Time :
8. <fmt:formatDate type="time" value="${Date}" /> </p>
9. <p> Formatted Date :
10.<fmt:formatDate type="date" value="${Date}" />  </p>
```

JSTL-Formatting tags

TimeZone.jsp

```
<%--   for setting time zone --%>
<fmt:setTimeZone value="IST" />
<c:set var="date" value="<%=new java.util.Date()%>" />
<p><b>Date and Time in Indian Standard Time(IST) Zone:</b>
<fmt:formatDate value="${date}"
                type="both" timeStyle="long" dateStyle="long"
/> </p>
<fmt:setTimeZone value="GMT-10" />
<p><b>Date and Time in GMT-10 time Zone:
</b><fmt:formatDate value="${date}"
                type="both" timeStyle="long" dateStyle="long"
/> </p>
```

JSTL-Formatting tags



JSP - Standard Tag Library (JSTL)

Tag Library	Function	URI	prefix
Core tag library	Variable support Flow Control Iterator URL management Miscellaneous	http://java.sun.com/jsp/jstl/core	c
Functions Library	Collection length String manipulation	http://java.sun.com/jsp/jstl/functions	fn
Internationalization tag library	Message formatting Number and date formatting	http://java.sun.com/jsp/jstl/fmt	fmt
SQL tag library	Database manipulation	http://java.sun.com/jsp/jstl/sql	sql
XML tag library	Flow control Transformation	http://java.sun.com/jsp/jstl/xml	x

JSTL SQL Tags List

The JSTL sql tags provide SQL support.

Syntax:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql"
        prefix="sql"
%>
```

JSTL SQL Tags List

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

JSTL SQL Tags List

Sql.jsp

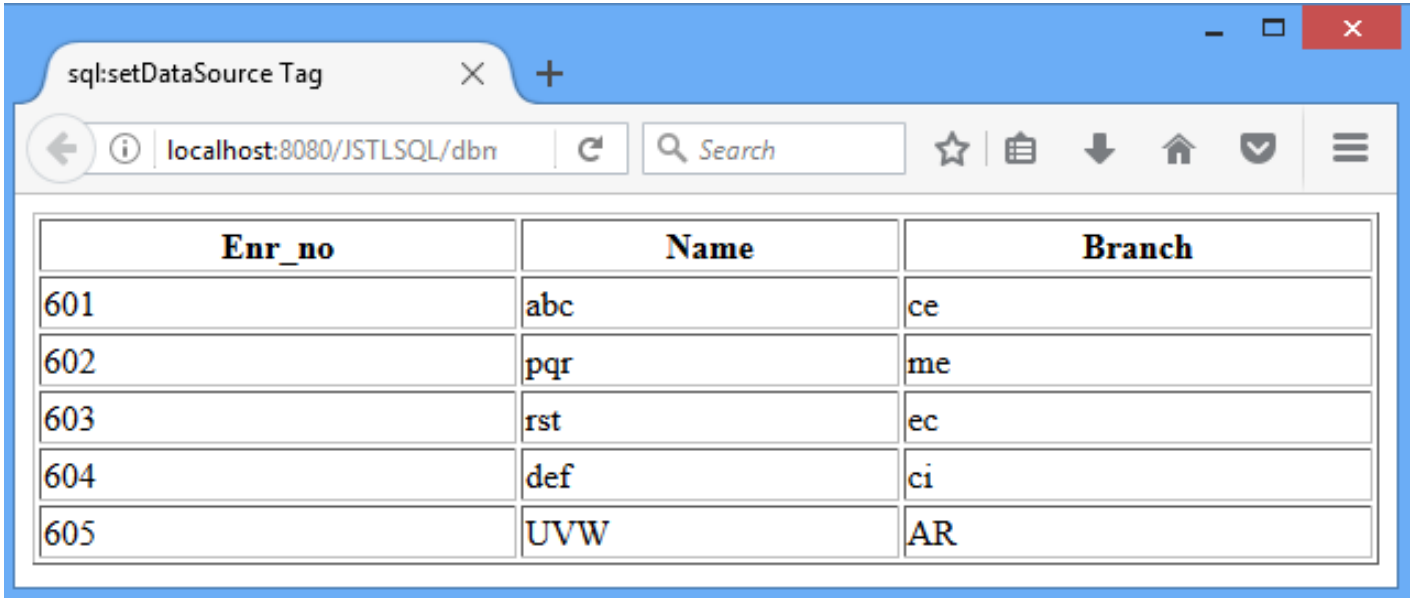
```
1. <%@ taglib uri="http://java.sun.com/jsp/jstl/core"
   prefix="c" %>
2. <%@ taglib uri="http://java.sun.com/jsp/jstl/sql"
   prefix="sql"%>
3. <sql:setDataSource var="db"
      driver="com.mysql.jdbc.Driver"
      url="jdbc:mysql://localhost:3306/gtu"
      user="root"  password="root"/>
4. <sql:query dataSource="${db}" var="rs">
5.    SELECT * from diet;
6. </sql:query>
```

JSTL SQL Tags List

Sql.jsp

```
7. <table border="1" width="100%">
8. <tr>
9.     <td>Enr_no</td>
10.    <td>Name</td>
11.    <td>Branch</td>
12.</tr>
13.<c:forEach var="table" items="${rs.rows}">
14.    <tr>
15.        <td><c:out value="${table.Enr_no}"/></td>
16.        <td><c:out value="${table.Name}"/></td>
17.        <td><c:out value="${table.Branch}"/></td>
18.    </tr>
19.</c:forEach>
20.</table>
```

JSTL SQL Tags List



Enr_no	Name	Branch
601	abc	ce
602	pqr	me
603	rst	ec
604	def	ci
605	UVW	AR

JSP - Standard Tag Library (JSTL)

Tag Library	Function	URI	prefix
Core tag library	Variable support Flow Control Iterator URL management Miscellaneous	http://java.sun.com/jsp/jstl/core	c
Functions Library	Collection length String manipulation	http://java.sun.com/jsp/jstl/functions	fn
Internationalization tag library	Message formatting Number and date formatting	http://java.sun.com/jsp/jstl/fmt	fmt
SQL tag library	Database manipulation	http://java.sun.com/jsp/jstl/sql	sql
XML tag library	Flow control Transformation	http://java.sun.com/jsp/jstl/xml	x

JSTL-XML tag library

- The JSTL XML tags are used for providing a JSP-centric way of manipulating and creating XML documents.
- The xml tags provide flow control, transformation etc.

Syntax:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/xml"
                                        prefix="x" %>
```

JSTL-XML tag library

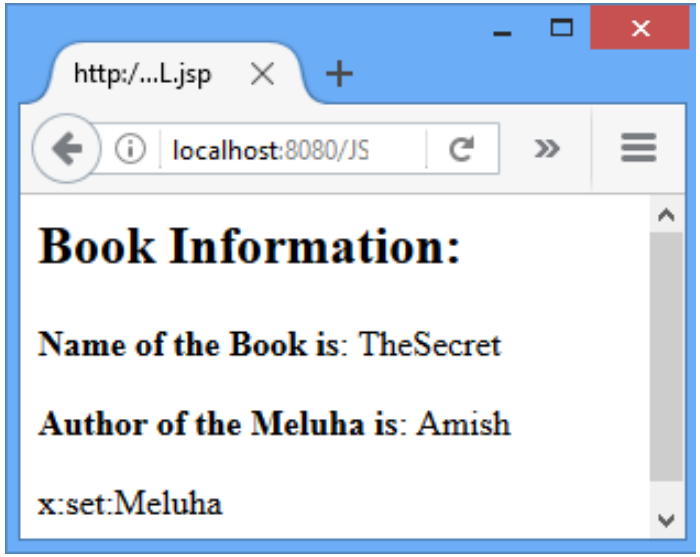
XML Tags	Descriptions
x:out	Similar to <%= ... > tag, but for XPath expressions.
x:parse	It is used for parse the XML data specified either in the tag body or an attribute.
x:set	It is used to sets a variable to the value of an XPath expression.
x:choose	It is a conditional tag that establish a context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

JSTL-XML tag library

```
1.  <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"
    %>
2.  <%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml"
    %>
3.  <c:set var="myBook">
4.  <books>
5.      <myBook>
6.          <title>TheSecret</title>
7.          <author>RhondaByrne</author>
8.      </myBook>
9.      <myBook>
10.         <title>Meluha</title>
11.         <author>Amish</author>
12.     </myBook>
13. </books>
14.  </c:set>
```

JSTL-XML tag library

```
15. <x:parse xml="${myBook}" var="output"/>
16. <b>Name of the Book is</b>:
17. <x:out select="$output/books/myBook[1]/title" />
18. <p><b>Author of the Meluha is</b>:
19. <x:out select="$output/books/myBook[2]/author" />  </p>
20. <x:set var="myTitle"
          select="$output/books/myBook[2]/title"/>
21. <p>x:set:<x:out select="$myTitle" /></p>
```



JSP 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 let you create new tags that you can insert directly into a JavaServer Page just as you would the built-in tags

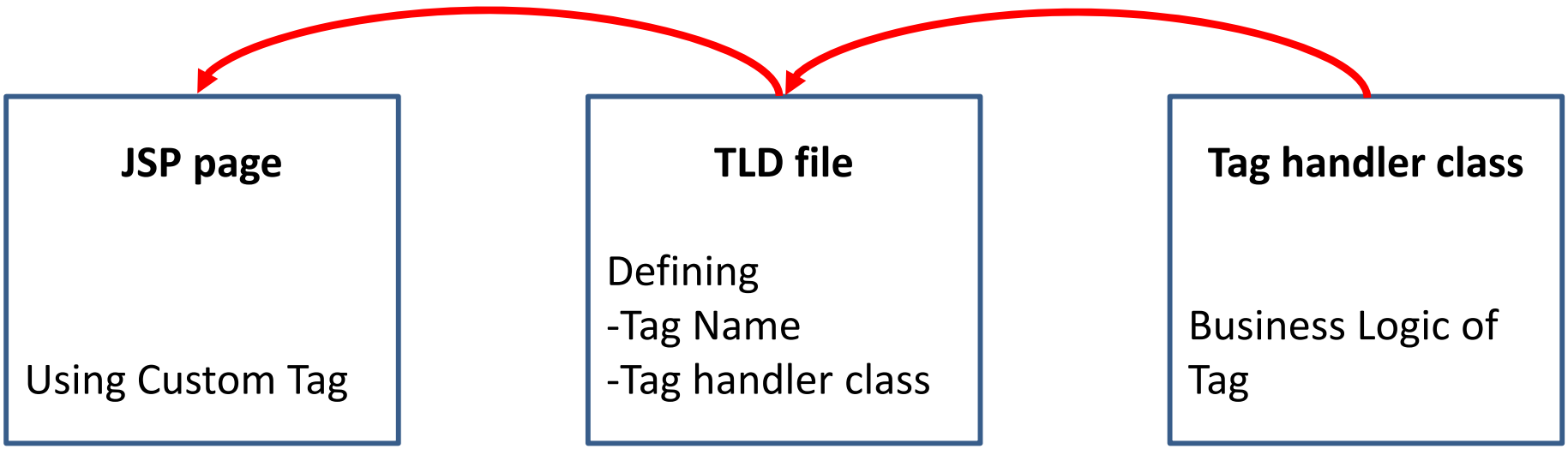
How to create Custom Tag?

To create a custom tag we need three things:

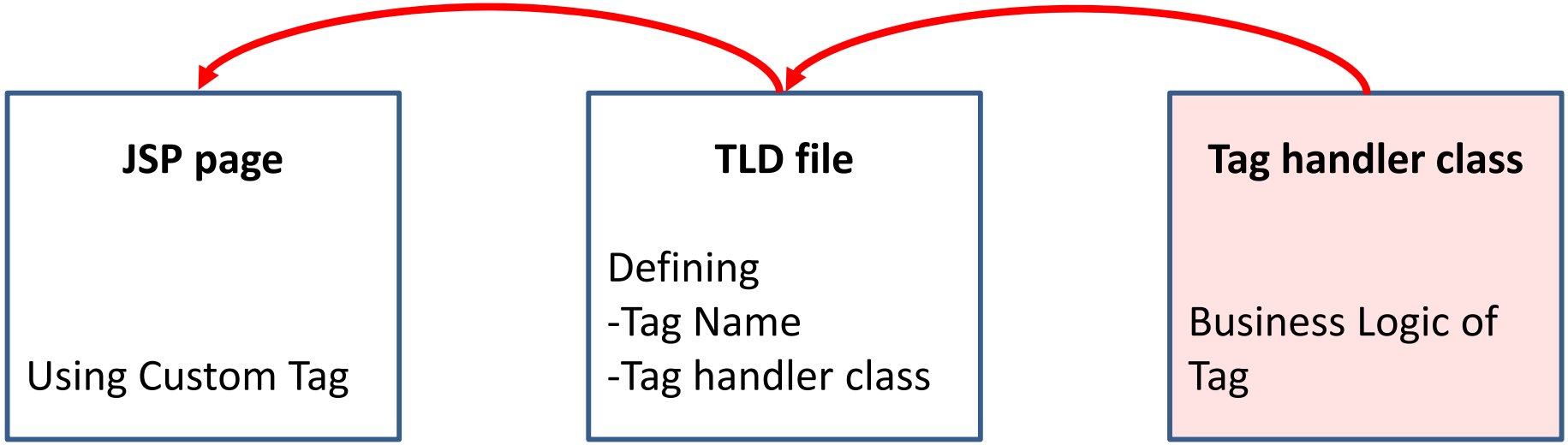
1) **Tag handler class:** In this class we specify what our custom tag will do, when it is used in a JSP page.

2**) TLD file:** Tag descriptor file where we will specify our tag name, tag handler class and tag attributes.

3) **JSP page:** A JSP page where we will be using our custom tag



JSP Custom Tag



Create the Tag handler class

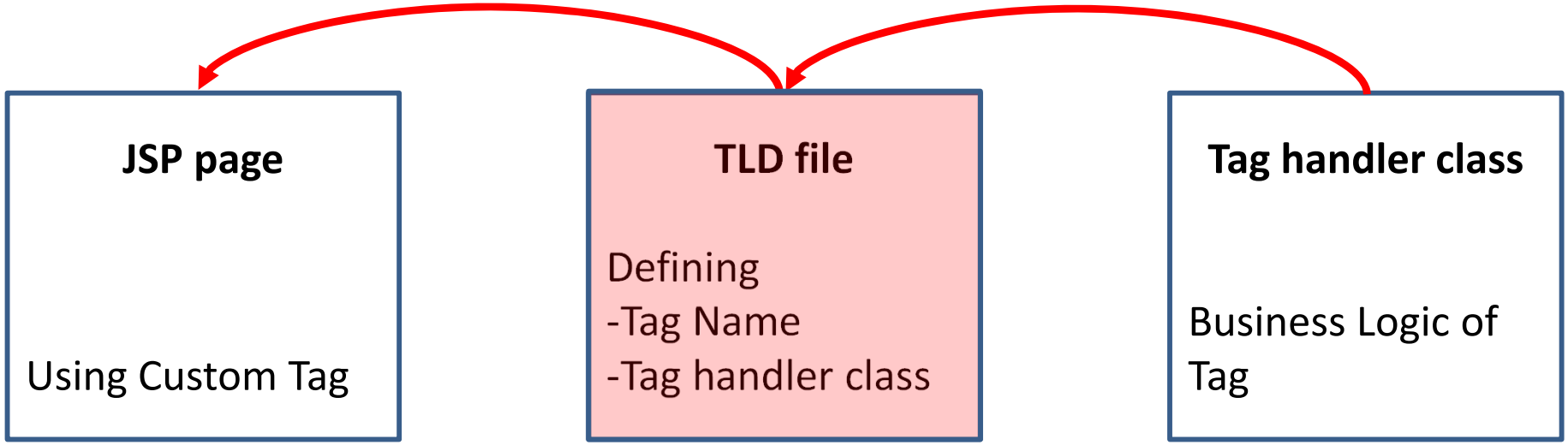
- Define a custom tag named <ex:Hello>
- To create a custom JSP tag, you must first create a Java class that acts as a tag handler.
- So let us create HelloTag class.

Create the Tag handler class

HelloTag.java

```
1. import javax.servlet.jsp.tagext.*;
2. import javax.servlet.jsp.*;
3. import java.io.*;
4. public class HelloTag extends SimpleTagSupport
5. {
6.   public void doTag() throws JspException,
                                          IOException
7.  {
8.     JspWriter out = getJspContext().getOut();
9.     out.println("Hello Custom Tag!");
10.  }
11.}
```

JSP Custom Tag



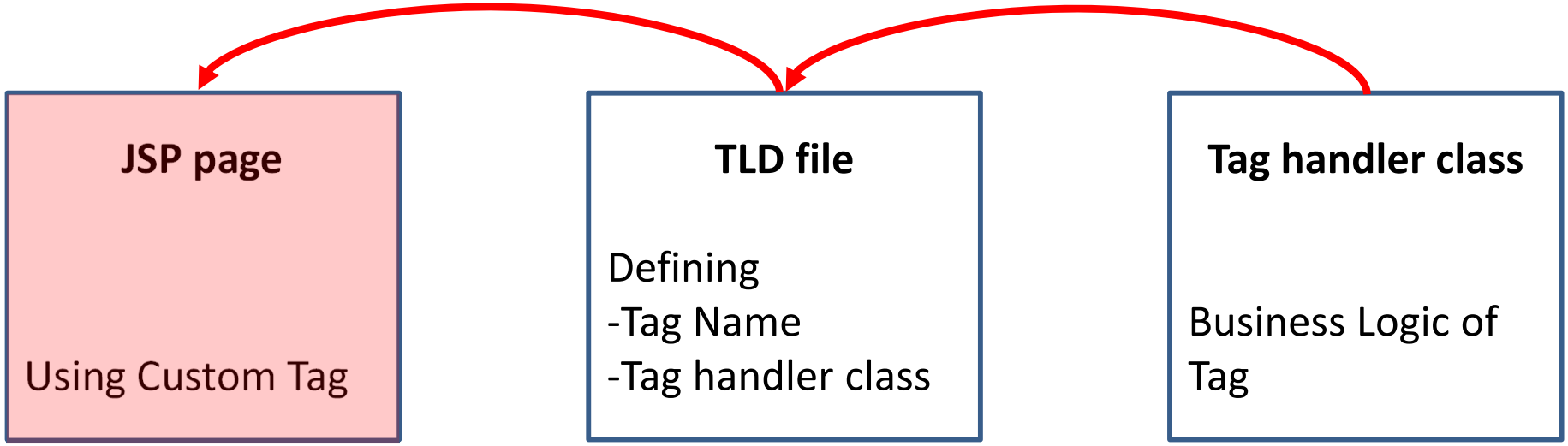
Create TLD file

- **Tag Library Descriptor** (TLD) file contains information of tag and Tag Hander classes.
- It must be contained inside the **WEB-INF** directory.

mytags.tld (Tag Library Descriptor)

```
1. <taglib>
2. <tlib-version>1.0</tlib-version>
3.   <jsp-version>2.0</jsp-version>
4.   <uri>WEB-INF/tlds/mytags.tld</uri>
5.   <tag>
6.     <name>Hello</name>
7.     <tag-class>MyPackage.HelloTag</tag-class>
8.     <body-content>empty</body-content>
9.   </tag>
10.</taglib>
```

JSP Custom Tag



JSP page

```
1. <%@taglib prefix="ex"
                 uri="WEB-INF/tlds/mytags.tld"%>
2. <html>
3.    <body>
4.       <ex:Hello/>
5.    </body>
6. </html>
```