Problem In Servlet
Lets overcome that with Java Server Pages (JSP)

Java Server Pages

What is JSP??

- A Java Server Pages (JSP) is a template for a web page.
- JSP uses Java code to generate an HTML dynamically.
- JSPs are run in a server side component known as a JSP container.
- which translate them into equivalent Java Servlet.
- JSP is a standard extension of the Servlet API.

Servlet Vs. JSP

Servlet	JSP
A Servlet is a Java class implementing the javax.servlet.Servlet interface that runs within a Servlet engine	JSP pages contain a mixture of HTML, Java scripts, JSP elements, and JSP directives, which will compiled by the JSP engine into a servlet
Servlet is dynamic HTML page code using Java Code	JSP is dynamic HTML code having java code when necessary
Servlet file is having extension .java	JSP file is having extension .jsp
Look and feel features are not in servlet	Look and feel features are there in JSP
Servlets is look and act like programs.	JSP is document-centric. (HTML + Java code)
Client side scripting is not possible with Servlet.	But, Some of the JSP functionality can be achieved on the client, using JavaScript.

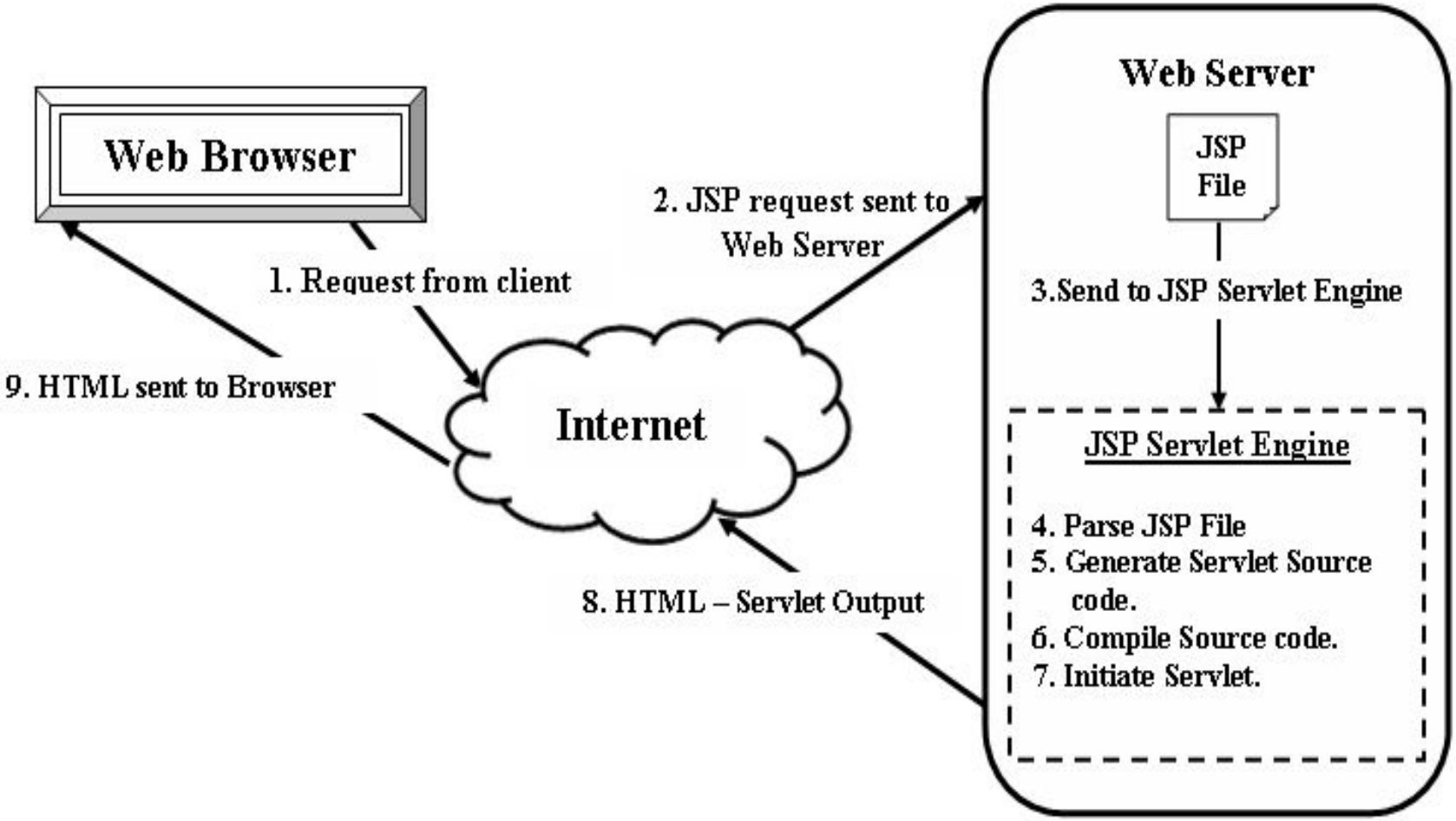
Benefits of JSP

- **Nobody can borrow the code:**
  - The JSP code written runs and remains on the web server. So issue of copy source code does not arise at all. All of JSP's functionality is handled before the page is sent to a browser.
- **Faster Loading of Pages:**
  - With JSP, decision can be made about what user wants to see at Web server prior the pages being dispatched.
- **No browser Compatibility Issues:**
  - JSP pages can run same way in browser like HTML page.
- **JSP support:**
  - JSP is supported by number of Web servers like Apache.
- **Compilation:**
  - Important benefit of JSP is that it is always compiled before the web server processes it. This allows the server to handle JSP pages much faster.
- **JSP elements in HTML/XML pages:**
  - A JSP page looks a lot like an HTML or XML page. It holds text marked with a collection of tags.

Disadvantages of using JSP

- **Attractive Java Code**
  - Putting Java code within web page is really bad design, but JSP makes it tempting to do just that. Avoid this as far as possible. It is done using template using.
- **Java code required**
  - To relatively simple things in JSP can actually demand putting java code in a page.
- **Simple tasks are hard to code**
  - Even including page headers and footers is a bit difficult with JSP.
- **Difficult looping in JSP**
  - In regular JSP pages looping is difficult. In advance JSP we can use some custom tags for looping
- **Occupies a lot of space**
  - Java server pages consume extra hard drive and memory (RAM) space.

JSP Architecture

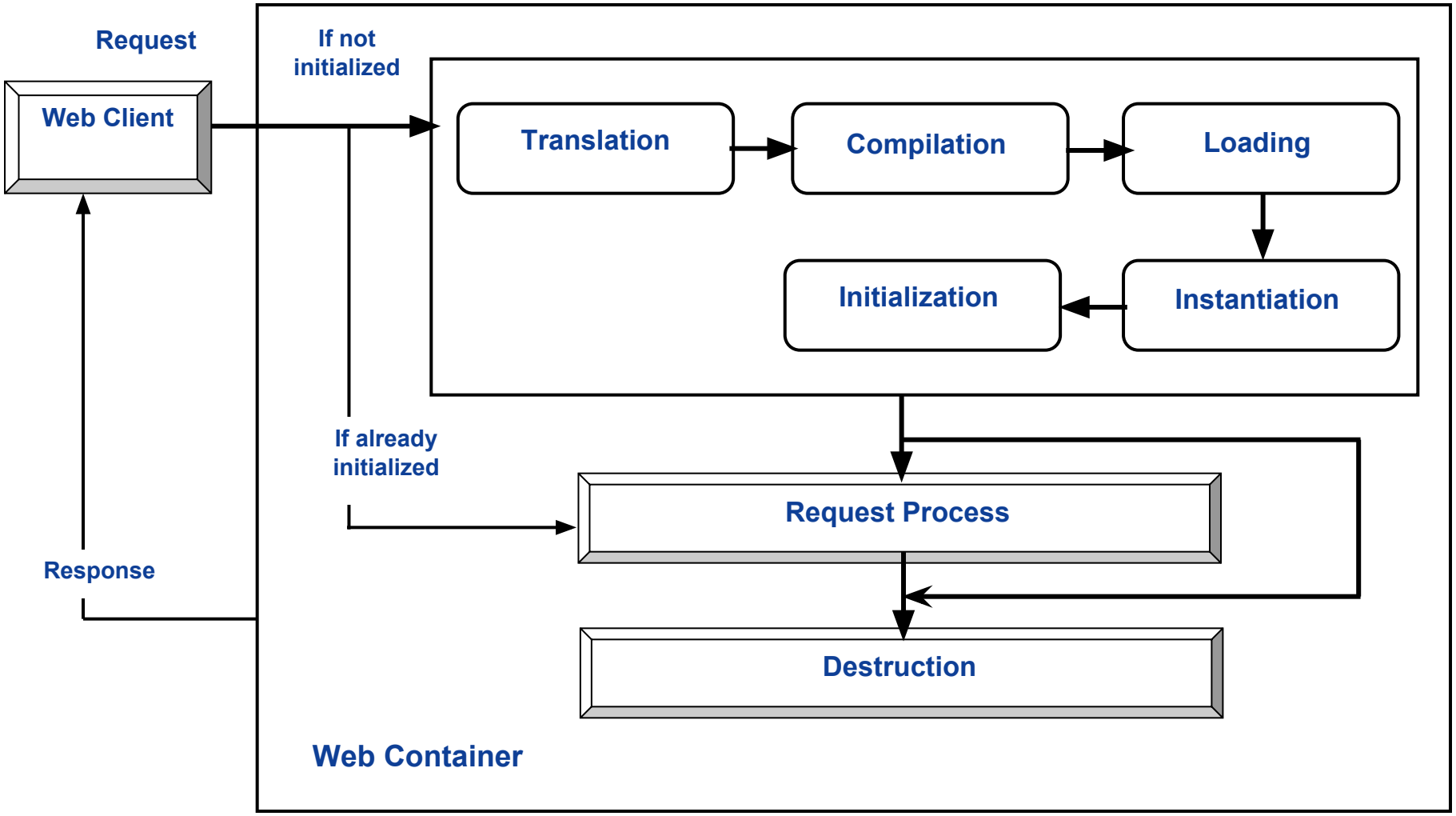


JSP Architecture…

- Steps for JSP request:

1. When the user goes to a JSP page web browser makes the request via internet.
2. JSP request gets sent to the Web server.
3. Web server recognizes the .jsp file and passes the JSP file to the JSP Servlet Engine.
4. If the JSP file has been called the first time, the JSP file is parsed, otherwise Servlet is instantiated.
5. The next step is to generate a special Servlet from the JSP file. The entire HTML required is converted to println statements.
6. The Servlet source code is compiled into a class.
7. The Servlet is instantiated, calling the init and service methods.
8. HTML from the Servlet output is sent via the Internet.
9. HTML results are displayed on the user's web browser

JSP Life Cycle



Life Cycle……

- **jspInit( )**
  - When JSP servlet instance is created, jspInit() method will be called. You can use servletContext object or servletConfig object to get initial parameters. It is similar to init( ) method of servlet.
  - public void jspInit( )
- **_jspService( )**
  - This is similar to service( ) method of Servlet. When JSP servlet instance is called, _jspService( ) method is called where request and response object are sent.
  - public void _jspService(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
- **public void jspDestroy( )**
  - This method is called when the JSP servlet instance is destroyed from the web container. It is also similar to destroy( ) method of Servlet. **Syntax of jspDestroy( ) is as follows:**
  - public void jspDestroy( )

JSP Elements

- JSP Elements are instruction to JSP container about what code to generate and how it should operate.
- Jsp Elements have a special identity to jsp compiler because it starts and ends with special kind of tags.
- HTML code is not compiled by the jsp compiler and also not recognized by the JSP container.
- There are basically three types of JSP elements which are as follow:
  - Directive Elements
  - Scripting elements
  - Action Elements

Directive Elements

- The role of the directives is to pass information to the JSP container.
- Following is the general syntax of Directive Element.

  <%@ directive {attribute name=”value”} %>
- There are 3 types of directives elements in JSP. They are as follow:
  - page Directive
  - include Directive
  - taglib Directive

Page Directive

Attribute name	Use (Description)	Default Value
Language	It is used to define the language which is used with the scriplet elemnts.Most probably its valid value is java only.	Java.
extends	It defines the fully qualified name of the super class of the jsp page..	
import	It defines the list of packages which are imported with the JSP page with its fully qualified name.	Following packages are automatically imported with JSP page: java.servlet.*, java.servlet,.http.*, java.servlet.jsp.*and java.lang.*.
Session	It defines the boolean indicating value true or false whether the jsp page requires and HTTP session or not. If the value is true , then the generated servlet will contains the which caused the Http Session to be created.	True

Page Directive……

Buffer	It specifies the size of output buffer.	8 kb
autoFlush	It also defines the boolean indicating value if its value is true it automatically flushed the buffer and if false then it throws an exception buffer overflow.	True
isThreadSafe	It also defines the boolean indicating value if its value is true its value is true jsp page handles all the requests simultaneously from the multiple threads and if false generates servlet declares that it implements the SingleThreadModel interface.	true
Info	It returns the string message information related to the jsp page using getServletInfo( ) method.	----
isErrorPage	It also defines the boolean indicating value if its value is true its value is true jsp page considered as a error page and if false it is a normal jsp page. In jsp exception is implicit variable so it is to create an error page and implements it with other jsp page to handle the exception.	false

Page Directive……

errorPage	It defines the URL of the error page if we want to implement the error page with the other jsp pages we can use this directive.	
contentType	It specifies the MIME type and character encoding which used with generated servlet.	<%@page contentType=”text/html” %>
PageEncoding	It defines the character encoding of jsp page it self.	ISO-8859-1 (Latin script) for JSP-style and UTF-8 (an 8-bit Unicode encoding) for XML-style tags.

Syntax:

<%@ page [attribute1 = “value1” attribute2 =”value2” attributen= valuen”] %>

Example: <%@ page import=“java.util.*” %>

Include Directive

- Include directive is used to include the static HTML pages (template) and dynamic jsp pages with the other jsp pages.
- For example if we want to set the same header and footer for all the jsp pages, we can create header.html and footer.html and can include it with necessary jsp pages.
- In short the include directive can be used to insert a part of the code that is common to multiple pages.
- Syntax: <%@ include file=”relative path” %>
- The file attribute is used to specify the name of the file to be included.

Scripting elements

- Scripting elements are used to write java code with the jsp file.
- As you know in JSP java code is embedded within HTML code. You need to write some Java language statements or use java features within the JSP page.
- HTML code and Java code is differentiated within JSP file using Scripting Elements.
- There are three types of scripting elements which are as follows:
  - **Scriptlets**
  - **Declaration**
  - **Expression**

Sciptlets

- Scriplets are block of java code for the jsp page. Scriplets starts with <% tag and ending with %> closing tag. As you know JSP is finally converted into servlet code and then JSP engine adds all processing statements of JSP and processes them under _jspService( ) method.
- The syntax of scriplet is:

```
<%
    Statement 1;
    Statement 2;
    …….
    Statement n;
%>
```

- **For example:**

```
<%
    if(age>=18)
        out.println("<p> You are eligible </p>");
    else
        out.println("<p> Sorry , you are not eligible </p>");
%>
```

Declarations

- Declaration tags are used to declare the variables, methods and instances of the classes within the jsp page.
- we can declare these all things with the scriptlets also but sciptlet code becomes the part of the _jspService( ) method , whereas declaration code is incorporates into generated source file outside the _jspService( ). Declarations starts with <%! tag and ending with %> closing tag.
- The syntax of declarations is:

```
<%!  Statement 1;
      Statement 2;
     ……..
     Statement n; %>
```

- Example:

```
<%!
        int count;
      public void jspInit()
      {
          ServletContext c= getServletContext();
          count=Integer.parseInt(c.getInitParameter("counter"));
      }
 %>
```

Expressions

- Expression Element is used to print value of any variable or any valid expression when the jsp page is requested.
- All the expressions are printed automatically by converting values into string values. If the result cannot be converted into a string, an error will be raised at translation time.
- An expression starts with <%= and ends with %>.
- Syntax: <%= expressions %>
- Example: Addition:<%= a+b %>

Action Elements

- Action elements are high level jsp elements which are used to create, modify and use other objects.
- Some standard action elements in JSP page are as follows:
  - <jsp:param>
  - <jsp:include>
  - <jsp:forward>
  - <jsp:plugin>

<jsp:param>

- This element is used to provide the tag/value pairs of information, by including these as sub-attributes of the <jsp:include>, <jsp:forward> and the <jsp:plugin> actions.

<jsp:param name=”pname” value=”pvalue”/>

OR

<jsp: param name=”pname” value=”pvalue”>

</jsp:param>

<jsp:include>

- This element is used to include static and dynamic resources of current JSP page. This object is just used to include resources on current JSP page.
- It can not be used to send response.
- Syntax:

```
<jsp: include page=”jsp page” flush=”true/false”>
    <jsp:param name=”pname” value=”pvalue” />
</jsp:include>
```

<jsp:forward>

- It is used to transfer control from current JSP file to another source of application.
- Whenever this action element is called, execution of current JSP page is stopped and control is transferred to another URL.
- **Syntax:**

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

- <jsp:forward> action element is same as forward( ) method of RequestDispatcher object in servlet programming.

<jsp:plugin>

- It is used to embed an applet and java beans in a with the jsp page.
- The tag automatically detects the browser type and inserts the appropriate HTML tag either <embed> or <object> in the output.
- Syntax:

<jsp: plugin type=”plugintype” code=”class filename” codebase=”url”>

</jsp: plugin>

Scope of the JSP Objects

- The lifetime and accessibility of an object is known as scope.
- In some cases, such as with the implicit objects, the scope is set and cannot be changed.
- With other objects, you can set the scope of the object. There are four valid scopes:
  - **Page Scope**
  - **Request Scope**
  - **Session Scope**
  - **Application Scope**

Scope.....

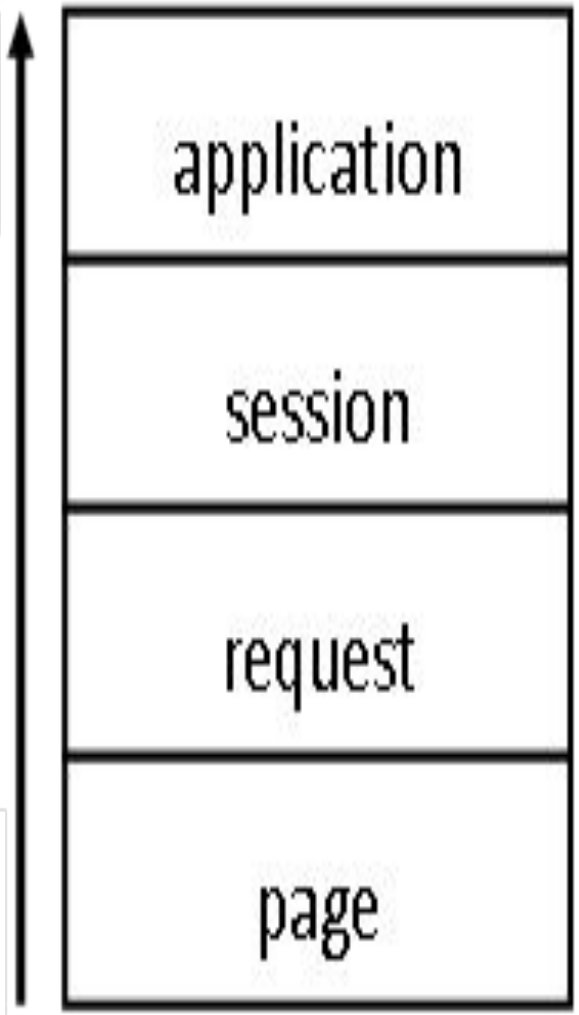
- **Page Scope:**
  - This scope is the most restrictive. With page scope, the object is accessible only within the current jsp page in which it is defined.
  - JSP implicit objects out, exception, response, config, pageContext, and page have ‘page’ scope.
- **Request Scope:**
  - JSP object created using the ‘request’ scope can be accessed from any pages that serve that request.
  - This means that the object is available within the page in which it is created, and within pages to which the request is forwarded or included.
  - Implicit object request has the ‘request’ scope.

Scope.....

- **Session Scope:**
  - Objects with session scope are available to all application components that participate in the client's session.
  - The JSP object that is created using the session scope is bound to the session object.
  - Implicit object session has the 'session' scope.
- **Application scope:**
  - This is the least restrictive scope. Objects that are created with application scope are available to the entire application for the life of the application.
  - Implicit object application has the 'application' scope

Scope.....

Least restrictive & Most Visible



Most restrictive & Least Visible

Object are accessible from pages that belongs to the same application

Object are accessible from pages that belongs to the same session

Object are accessible from pages processing the request where they were created

Object are accessible only within the pages where they were created

Implicit objects of JSP

- Objects that are frequently used in servlet are provided implicitly by JSP are known as Implicit Objects.
- The reason behind implicit object is, JSP page is also converted into servlet at last, so some of objects should be provided by JSP also for programming similar to servlet.
- Implicit objects are used within scriptlet and expression elements.
- Following are Implicit Objects:
  - request, response, out ,session, config, exception, & application

Implicit objects….

- **request:**
  - request object in JSP is also belongs to **javax.Servlet.HttpServletRequest** class.
  - The request object has request scope. That means that the implicit request object is in scope until the response to the client is complete.
- **response:**
  - response object in JSP is belongs to **javax.Servlet.HttpServletResponse** class.
  - The scope of response object is page only because it is used to generate output of a particular page. If you want to transfer control from one page to another you can use **sendRedirect( )** method of response.

Implicit objects…

- **session**
  - It is belongs to **javax.Servlet.http.HttpSession** class.
  - Session object for the requesting client is created under session implicit object.
- **application**
  - It is used to set values and attributes at application level.
  - Application object of JSP is similar to **ServletContext** object of servlet programming.
- **out**
  - It is used to write output into the output stream of client.
  - The scope of out is current page.
  - It is created using **javax.Servlet.jsp.JspWriter** class.

Implicit objects…

- **config**
  - This implicit object is similar to ServletConfig object of Servlet.
  - config is created by **javax.Servlet.ServletConfig** class.
- **page**
  - As you know JSP page is converted into Servlet Class at last.
  - Then servlet instance is created of that particular JSP servlet class.
  - Page is the instance of JSP servlet class created by Web Container for the current request.
  - It is similar with “this” keyworkd in Java

Error Handling and Exception

- The Exception refers to the error which occurred at runtime.
- As we have seen before that in java we can handle exception very easily through the Exception object.
- In JSP exception is an implicit object with page scope.
- It is an instance of java.lang.Throwable, as you know that Throwable class is the super class of all the exception and error classes in the java.
- Here with the JSP we can handle exception in three way
  - With scriplets (Try…Catch).
  - By creating an error page.
  - With deployment descriptor(Web.xml).

Exception handling…..

- Using PageDirective:
  - To create an error page with JSP we can use **isErrrorpage =true** attribute of page directive.
  - After creating an error page we can handle the exception with any JSP page by invoking this error page with it using **errorPage=" URL of the error page"** attribute of the page directive
- Using Web.xml:
  - Deployment Descriptor is a web.xml file which defines the classes, resources and configuration of the application and how the web server uses them to serve web requests.
  - If an error page is defined for handling an exception, the request is directed to the error page's URL.
  - The web application **deployment descriptor uses the <error-page> tag** to define web components that handle errors.
  - We can set deployment descriptor for error handling in two ways, either using exception type or using error code.

Expression Language (EL)

- Expression Language was first introduced in JSTL 1.0 (JSP Standard Tag Library).
- Before the introduction of JSTL, scriptlets were used to manipulate application data.
- JSTL introduced the concept of an expression language (EL) which simplified the page development by providing standard tag libraries.
- These tag libraries provide support for common, structural tasks, such as: iteration and conditionals, processing XML documents, internationalization and database access using the Structured Query Language (SQL).
- The Expression Language introduced in JSTL 1.0 is now incorporated in Java Server Pages specification (JSP 2.0).
- It eliminates the need to use JSP scriptlets & expressions and uses a higher-level syntax for expressions.

EL....

- The JSP expression language allows a page author to access a bean using a simple syntax such as:
  - ${expr}
- In the above syntax, expr stands for a valid expression.
- It must be noted that this expression can be mixed with static text, and may also be combined with other expressions to form larger expressions.
- To deactivate the evaluation of EL expressions, we specify the isELIgnored="true" which is an attribute of page directive. (Default value false)
- We can also enforce that all jsp pages within application have to use only EL and no scripting is allowed.
- To enforce EL-only with no scripting, we can use scripting-invalid in web.xml file.

EL implicit Objects

- **pageContext:**
  - It can be used to access the JSP implicit objects such as request, response, session, out, servletContext etc.
  - For ex. ${pageContext.response} evaluates to the response object for the page.
- **param:**
  - It is used to map a request parameter name to a single String parameter value obtained by calling ServletRequest.getParameter (String name).
  - $(param.name) = request.getParameter (name).
- **paramValues:**
  - Retrieves multiple parameter's values for single param
  - ${paramvalues.name)=request.getParamterValues(name)

EL implicit Objects ….

- **header:**
  - ${header.name} = request.getHeader(name).
- **headerValues:**
  - ${headerValues.name} = request.getHeaderValues(name).
- **cookie:**
  - It is used to map cookie names to a single cookie object.
  - A client request to the server can contain one or more cookies. The expression ${cookie.name.value} returns the value of the first cookie with the given name.
- **initParam:**
  - It is used to map a context initialization parameter name to a single value obtained by calling ServletContext.getInitparameter(String name).

Scope of Objects

- EL impilcit objects are not as same as the implicit objects available for JSP scripting except for pageContext.
- JSP and EL implicit objects have only one object in common (pageContext) and pageContext has properties for accessing all of the other JSP implicit objects.

EL Functions

- **Defining Functions:**
  The JSP expression language allows you to define a function that can be invoked in an expression.
  - For that first we have to create a Java class with function declaration and definition.
  - Then after we have to create a TLD file to register that function.
  - Create a JSP file and provide taglib directive with URI of TLD file and prefix to use that function.

Taglib directive

- Taglib directive declares that the JSP file uses custom tags, names the tag library that defines them, and specifies their tag prefix.
- **Syntax:**

```
<%@ taglib uri="URIToTagLibrary" prefix="tagPrefix" %>
```

Java Standard Tag Library

- Java Server Page's custom tags and tag handlers are designed to help you invent your own tags.
- This tag collection is called the Java Server Pages Standard Tag Library (JSTL).
- The main JSTL package names start with javax.servlet.jsp.jstl, so JSTL should henceforth be considered part of JSP.
- If you are working with JDK 1.4 or later, you probably need only jstl.jar and standard.jar files.
- If you are using Netbeans editor than you have to add JSTL 1.1 library into the library folder. But if you are doing program manually using tomcat than you have to add these two jar files into the WEB-INF\lib folder.

JSTL Tags

- In JSTL all tags are classified in to four families of tags:

1. **core:** It includes General purpose programming tags that let you get and set variables, loop through a collection, and write to the output stream.
2. **xml:** It includes tags that parse, generate, and transform XML.
3. **sql:** It includes Database access tags, including connection support, transactions, queries, and updates.
4. **fmt:** It includes tags for formatting numbers and dates and for localizing international text.

JSTL Tags…

- To use any of these libraries in a JSP, need to declare using the taglib directive in the JSP page, specifying the URI and the Prefix.

Library	Prefix	URI
Core	c	http://java.sun.com/jsp/jstl/core
XML	xml	http://java.sun.com/jsp/jstl/xml
SQL	Sql	http://java.sun.com/jsp/jstl/sql
Formatting	fmt	http://java.sun.com/jsp/jstl/fmt

Core Tags

- The base of JSTL is the core taglib, which provides access to application objects and programming logic.
- There are basic three groups of actions in this library 1) General purpose tags 2) Condition tags and 3) URL-related tags.
- To use Core tags we need to add following taglib directive.

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

< c:out >

- The <c:out> tag is a General purpose tag. It evaluates an expression which may be supplied in an attribute value or contained in the tag body. The resulting value is written to the current JSP output stream.
- Syntax of this tag is as below:

```
<c:out value="value" [escapeXml="{true|false}"] [default="defaultValue"] />
                                    OR
<c:out value=”value” [escapeXml=”{true|false}]>
    Default value
</c:out>
```

Attribute	Description	Required	Default
Value	Data to output	Yes	None
Default	Fallback data to output if value is empty	No	Body
escapeXml	true to escape special characters	No	True

<c:set>

- The set tag sets the value of an EL variable or the property of an EL variable in any of the JSP scopes (page, request, session, or application).
- If the variable does not already exist, it is created.
- **Syntax to set EL variable:**

```
<c:set value=”value” var =”varname”
                              scope=” {page | request | session |
application}”] />
```

< c:if >

- This is condition tag which is useful for conditional actions.
- Syntax of this tag is as given below:

```
<c:if test =”testcondition” var =”varname”
                        scope=” {page | request | session |
application}”] />
                    OR
<c:if test =”testcondition” var =”varname”
                        scope=” {page | request |  session |
application}”] >
                Body content
</c:if>
```

< c:if >.....

Attribute	Description	Required	Default
Test	Condition to evaluate	Yes	None
Var	Name of variable to store test condition's result	No	None
Scope	Scope of variable	No	Page

SQL Tags

- The SQL tag library provides access to relational database systems through JDBC.
- It provides actions to perform transactional database queries and updates and easily access query results.
- To use SQL tags we need to add following taglib directive.

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

<sql:setDataSource>

- It creates and stores in a scoped variable an SQL data source.
- This tag cannot have a body.
- Either the dataSource or url attribute must be specified.

```
<sql: setDataSource
  dataSource=”dataSource” |
  url = “jdbcurl”
  [driver=”driverClassName]
  [user=”username”] [password=”password”]
  [var=”varname]
  [scope= “{page| request | session | application}”]
/>
```

<sql: query>

- It performs a database query like select statement.
- The query should be expected to return a ResultSet.

```
<sql: query sql=”sqlQuery” var = “varname”
    [scope=” {page | request | session | application}”]
    [datasource = “dataSource”]
    [maxRows= “maxRows”]
    [startRow= “startRow”] >
    <sql: param> actions (optional)
</sql: query>
```

<sql: update>

- Performs a database insert, update, delete or other DDL statement, that does not return any results.

```
<sql:update sql = ”sqlUpdate”
        [dataSource = “dataSource”]
        [var = “varname”]
      [scope=” {page | request | session |
    application}”] >
      <sql: param> actions (optional)
</sql: query>
```

Custom Tags in JSP

- **Custom tags** are user-defined tags. They eliminates the possibility of scriptlet tag and separates the business logic from the JSP page.
- The same business logic can be used many times by the use of custom tag.

Advantages of Custom Tags

- **Eliminates the need of scriptlet tag** The custom tags eliminates the need of scriptlet tag which is considered bad programming approach in JSP.
- **Separation of business logic from JSP** The custom tags separate the business logic from the JSP page so that it may be easy to maintain.
- **Re-usability** The custom tags makes the possibility to reuse the same business logic again and again.

Syntax to use custom tag

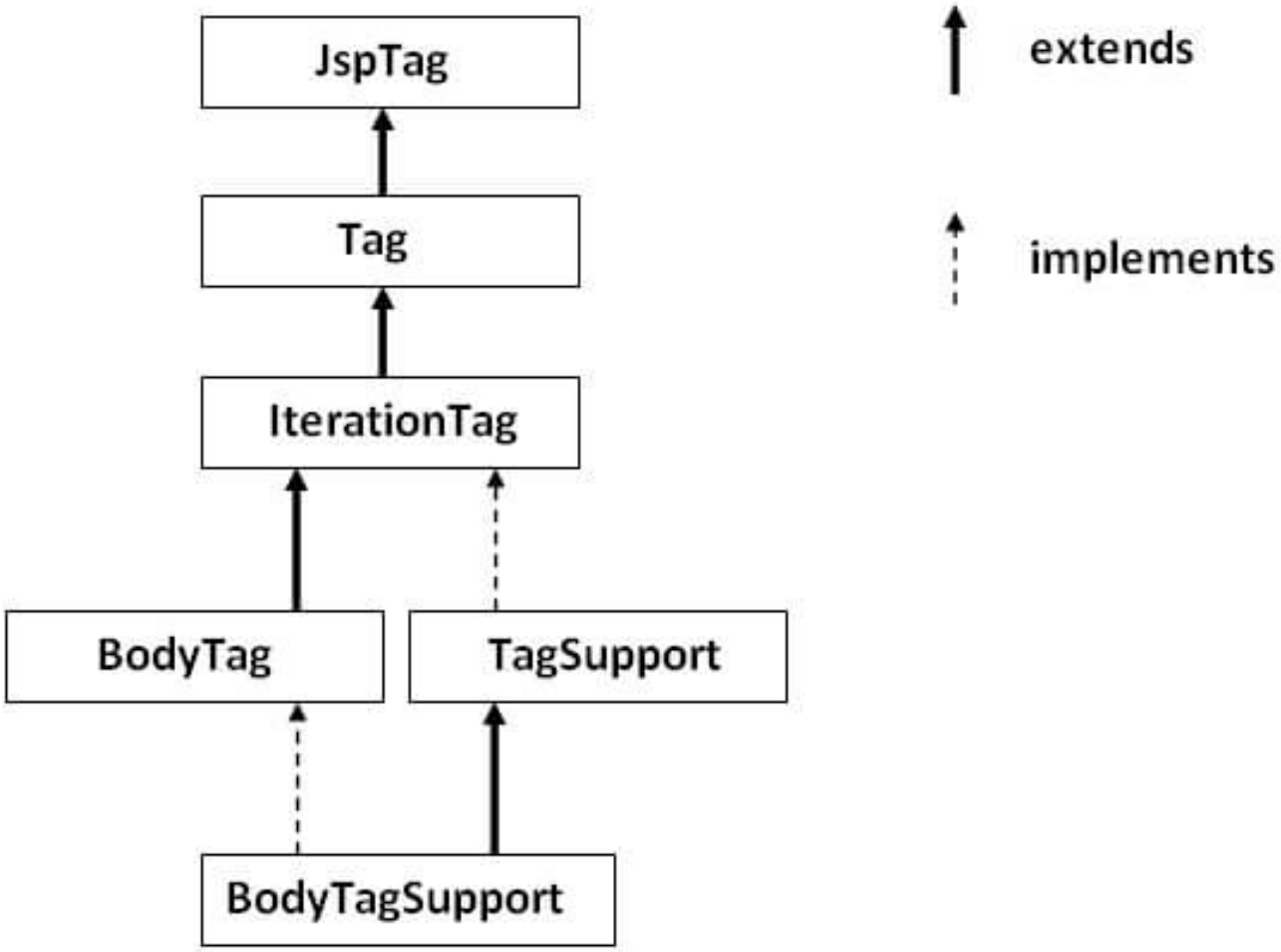
- There are two ways to use the custom tag. They are given below:
- **<prefix:tagname** attr1=value1...attrn=valuen **/>**

- **<prefix:tagname** attr1=value1...attrn=valuen **>**

  body code

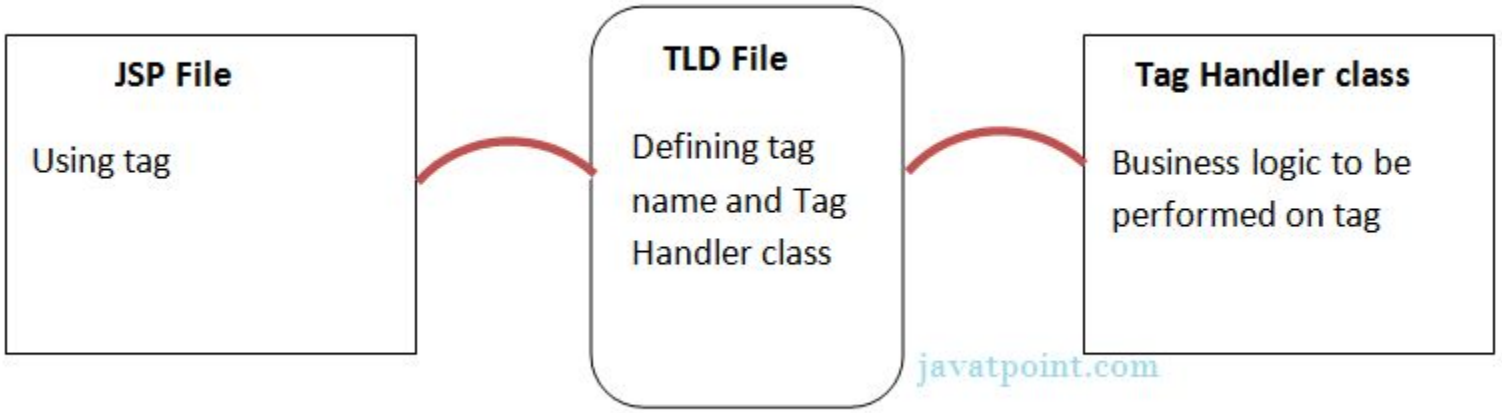
</prefix:tagname>

JSP Custom Tag API



JSP Custom Tag

1) Create the Tag Library Descriptor (TLD) file and define tags.
2) Create the Tag handler class and perform action at the start or at the end of the tag.
3)Create the JSP file that uses the Custom tag defined in the TLD file



1) Create the TLD file

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

```
<taglib>
    <tlib-version>1.0</tlib-version>
    <jsp-version>1.2</jsp-version>
    <short-name>simple</short-name>
    <uri>http://tomcat.apache.org/example-taglib</uri>
    <tag> <name>today</name>
<tag-class>com.javatpoint.sonoo.MyTagHandler</tag-class> </tag>
</taglib>
```

2) Create the Tag handler class

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

Tags with Attributes

- There can be defined too many attributes for any custom tag. To define the attribute, you need to perform two tasks:
- Define the property in the TagHandler class with the attribute name and define the setter method
- define the attribute element inside the tag element in the TLD file

Tags with Bodies

- You can include a message in the body of the tag as you have seen with standard tags. Consider you want to define a custom tag named **<ex:Hello>** and you want to use it in the following fashion with a body –
- <ex:Hello> This is message body </ex:Hello>
- getJspBody().invoke(out); or
  - StringWriter sw = new StringWriter();
  - getJspBody().invoke(sw);

Communication between Tags

- Custom tags communicate with each other through shared objects. There are two types of shared objects: public and private.

- In the following example, the c:set tag creates a public EL variable called aVariable, which is then reused by anotherTag.

- <c:set var="aVariable" value="aValue" />
- <tt:anotherTag attr1="${aVariable}" />

- Nested tags can share private objects. In the next example, an object created by outerTag is available to innerTag. The inner tag retrieves its parent tag and then retrieves an object from the parent. Because the object is not named, the potential for naming conflicts is reduced.

```
<tt:outerTag>
    <tt:innerTag />
</tt:outerTag>
```

XML Tags

- The XML taglib can be used for parsing, writing, or transforming XML. A common pattern for using the XML tags is as follows:
- XML tags have also basic three groups of actions in this library 1) Core Actions 2) Condition tag or Flow control action and 3) Transform Actions.

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

< x:parse >

- This is a Core action tag of XML tag library.
- It parses XML content, provided by the value attribute or the tags body, into a scoped variable(s).
- This variable can then be used for subsequent processing by other XML tags.

<x:parse doc =”XMLdocument” var =”varname”/>

< x:out >

- This xml tag is also core action tag.
- It prints the result of the XPath expression as a string.
- It evaluates XPath expression in the select attribute and writes the result to the servlet output stream.
- This is the similar of <c:out> tag, but it uses XPath instead of EL as the expression language.
- <x:out select = “XPathExpression” [escapeXml = {true|false}] />

< x:set >

- This xml tag is also core action tag.
- It saves the result of the select XPath expression to a scoped variable.
- Returned value may be a node set (XML fragment), boolean, string, or number. It is same as <x:out> but stores its results in a variable.
- <x:set select = “XPathExpression” var =”variablename” scope=” {page | request | session | application}”] />

< x:if >

- This is a conditional tag or Flow control action tag of xml tag library.

<x:if select = “XPathExpression” var =”variablename”
scope=” {page | request | session | application}”] />

OR

<x: if select = “XPathExpression” [var =”variablename”]
scope=” {page | request | session | application}”] >

Body Content

</x:if>

Format Tags

- This library consists of tags that handle internationalization, locale, time zone, number formatting and date formatting.
- These tags are used to format and parse numbers, currencies, percentages, dates and times.
- To use format tags we need to add following taglib directive.

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

<fmt:formatNumber>

- It is a format action tag.
- It formats a number, currency, or percentage in a locale-sensitive manner.
- The formatted value is printed or stored in a scoped variable.
- It performs number formatting based on java.text.Format.
- Syntax and attribute description of this tag is as below:

```
<fmt: formatNumber value=”numericValue”
                        [type=” {number | currency|
    percent}”] ……. />
```

<fmt:formatDate>

- It formats a date and/or time in a locale-sensitive manner.
- The formatted value is printed or stored in a scoped variable.
- It formats a date in the manner used by java.text.DateFormat.
- Syntax of this tag is as given below:

```
<fmt:formatDate value=”date” [type=” {time
    |date | both}]
    [pattern=”customPattern”] …… />
```

J2EE

Thank You