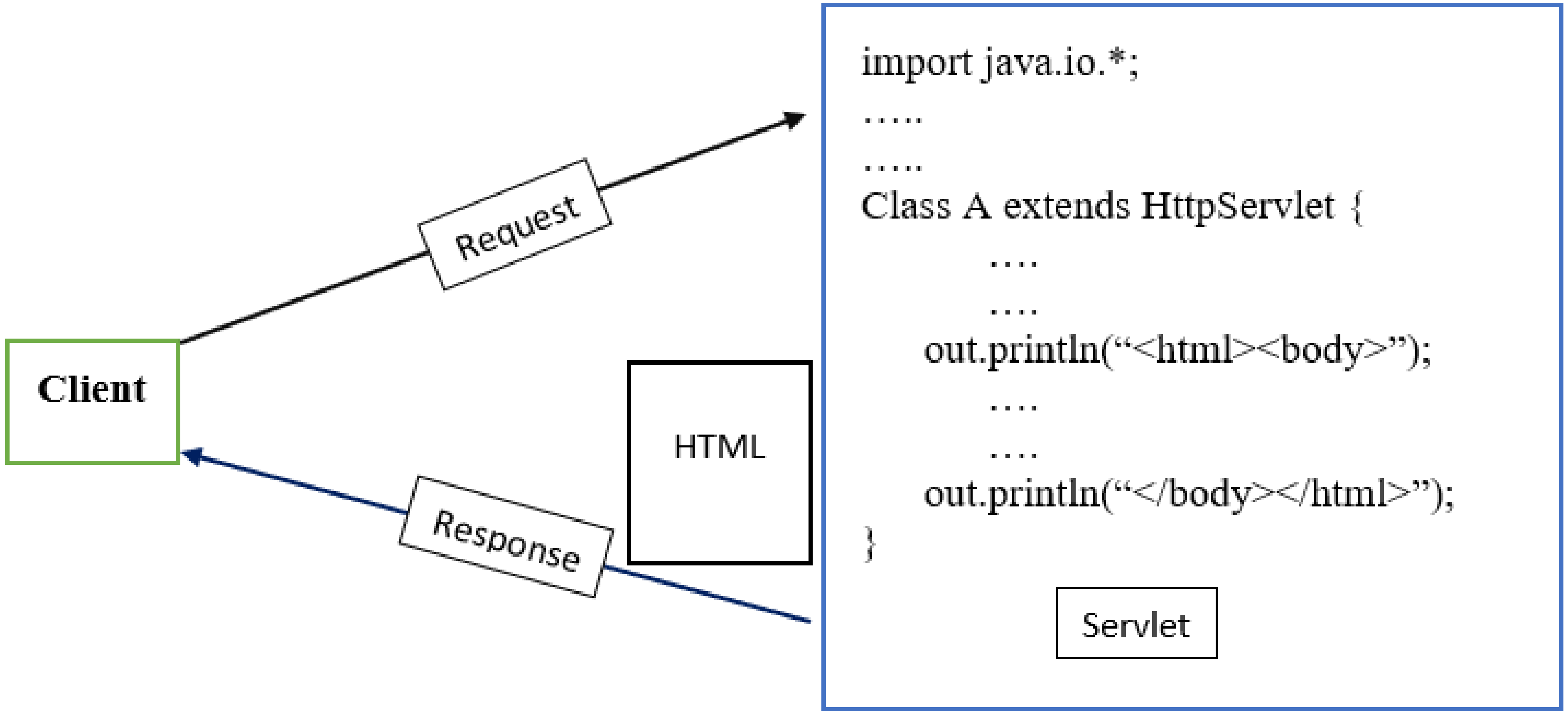
JSP

Lecture 3

Introduction to JSP (Java Server Pages)

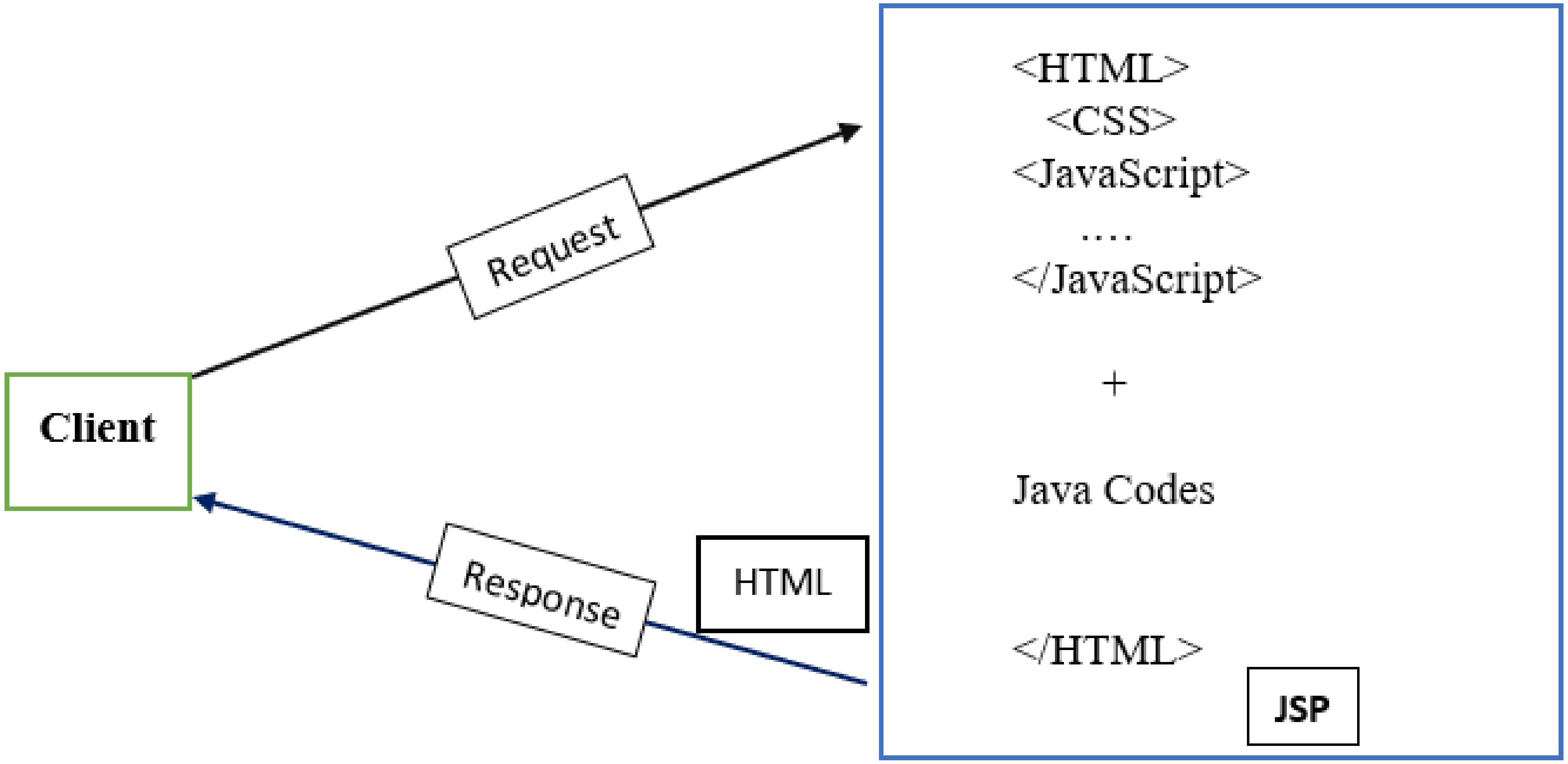
- JSP is the extension of the Servlet technology, but JSP provides more functionality than servlet.

Why JSP?



```
import java.io.*;
…..
…..
Class A extends HttpServlet {
        ….
        ….
    out.println("<html><body>");
        ….
        ….
    out.println("</body></html>");
}
```

Why JSP?



Disadvantages of Servlet

- Static content are generated by java from inside Servlet. That’s why, designing in Servlet is difficult.
- For every request in servlet you have to write service method which is very tiresome process.
- Whenever modification is made in static content (presentation logic) then servlet need to recompiled and redeployed.

How JSP translated into Servlets?

- **JSP**

```
<%@page import = “java.io.*, ... ”  %>
….
<html>
<%= Welcome to JSP %>
<%!
   int a, b;
%>
<%
     S = a * b;
     Out.println(“Sum = ” + S);
%>
</html>
```

- **Servlet**

```
import java.io.*;
Public class A extends HttpServlet {
….
int a, b;
….
…. service(request, response) {
Out.println(“<html>”);

Out.println(“Welcome to JSP ”);
S = a * b;
Out.println(“Sum = ” + S);


Out.println(“</html>”);
}
```

JSP Scripting Tag / Elements

- In JSP, java code can be written inside the JSP page using the scripting elements.
- The scripting elements provides the ability to insert java code inside the JSP.
- There are four types of scripting elements:
  - Declaration tag
  - Scriptlet tag
  - Expression tag
  - Directives tag
    - ✓ Page directive tag
    - ✓ Include directive tag
    - ✓ Taglib directive tag

Declaration Tag (<%! … %>)

- The JSP declaration tag is used to declare fields and methods.
- The code written inside the jsp declaration tag is placed outside the service() method of auto generated servlet.

```
<html>
<%!
        int a = 5; //Variables declaration
        int b = 10;

        public int product (int x, int y) // Method declaration
        {
                return (x * y);
        }
%>
</html>
```

Expression Tag (<%= … %>)

- 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.

```
<html>
<%!
        int a = 5, b = 10;
        public int product() { return (a * b); }
%>

Product of <%= a %> and <%= b %> is: <%= product() %>

</html>
```

Scriplet Tag (<% … %>)

- A scriptlet tag is used to execute java source code in JSP.

```
<html>
<%
            String name=request.getParameter("name");
            out.print("welcome "+name);
%>

</html>
```

Page Directives Tag (<%@ page … %>)

- The page directive defines attributes that apply to an entire JSP page.
- <%@ page attribute="value" %>
- Attributes of JSP page directive are
- import, contentType, extends, info, buffer, language, isELIgnored, isThreadSafe, autoFlush, session, pageEncoding, errorPage, isErrorPage

```
<%@page import = “java.io.*, java.sql.*” %>

<html>
……
……
</html>
```

Include Directives Tag (<%@ include … %>)

- The include directive is used to include the contents of any resource it may be jsp file, html file or text file.
- <%@ include file="filename" %>

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

<html>
……
……
</html>
```

Taglib Directives Tag (<%@ taglib … %>)

- The JSP taglib directive is used to define a tag library that defines many tags. We use the TLD (Tag Library Descriptor) file to define the tags. In the custom tag section we will use this tag so it will be better to learn it in custom tag.
- <%@ taglib file="fileName" %>

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<html>
……
<c:set var="name" value="Debarshi Mazumder"></c:set>
        My name is <c:out value="${name}"></c:out>
        <c:if test="${5>2}"> <br><br> 5 > 2 is true</c:if>
……
</html>
```

JSP Implicit Objects

- These Objects are the Java objects that the JSP Container makes available to the developers in each page and the developer can call them directly without being explicitly declared.
- JSP Implicit Objects are also called pre-defined variables.
- There are 9 JSP implicit objects.
- These are
    - Request, response, out, session, application, config, pageContext, page, Exception.

JSP Implicit Objects

- 1. Request: This is the HttpServletRequest object associated with the request.
- 2. Response: This is the HttpServletResponse object associated with the response to the client.
- 3. Out: This is the PrintWriter object used to send output to the client.
- 4. Session: This is the HttpSession object associated with the request.
- 5. Application: This is the ServletContext object associated with the application context.
- 6. Config: This is the ServletConfig object associated with the page.
- 7. PageContext: This encapsulates use of server-specific features like higher performance JspWriters.

JSP Implicit Objects

- 8. Page: This is simply a synonym for this and is used to call the methods defined by the translated servlet class.
- 9. Exception: The Exception object allows the exception data to be accessed by designated JSP.

Request Implicit Object

- The JSP request is an implicit object of type HttpServletRequest i.e., created for each jsp request by the web container.
- It can be used to get request information such as parameter, header information, remote address, server name, server port, content type, character encoding etc.

```
<%
        String name=request.getParameter(“user_name");
        out.print("welcome "+name);
%>
```

Response Implicit Object

- In JSP, response is an implicit object of type HttpServletResponse. The instance of HttpServletResponse is created by the web container for each jsp request.
- It can be used to add or manipulate response such as redirect response to another resource, send error etc.

```
<%
        response.sendRedirect ("http://www.google.com");
%>
```

Out Implicit Object

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

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

Config Implicit Object

- In JSP, config is an implicit object of type ServletConfig.
- The config object is created by the web container for each jsp page.
- This object can be used to get initialization parameter for a particular JSP page from the web.xml file.

```
<%
        String driver=config.getInitParameter("dname");
        out.print("driver name is="+driver);
%>
```

Application Implicit Object

- 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 configuaration file (web.xml). It can also be used to get, set or remove attribute from the application scope.

```
<%
        String driver=application.getInitParameter("dname");
        out.print("driver name is="+driver);
%>
```

Session Implicit Object

- 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.

```
<%
        session.isNew();
        session.setAttribute("user",name);
        String name=(String)session.getAttribute("user");
        out.print("Hello "+name);
%>
```

Page Implicit Object

- page is an implicit object of type Object class.
- This object is assigned to the reference of auto generated servlet class. It is written as:

Object page=this;

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

Since, it is of type Object it is less used because you can use this object directly in jsp.For example:

```
<% this.log("message"); %>
```

Exception Implicit Object

- 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" %>


<%= exception %>
```

Exception Handling in JSP

- The exception is normally an object that is thrown at runtime.
- Exception Handling is the process to handle the runtime errors.
- In JSP, there are two ways to perform exception handling:
- By errorPage and isErrorPage attributes of page directive
- By <error-page> element in web.xml file

Example using isErrorPage

- First.jsp

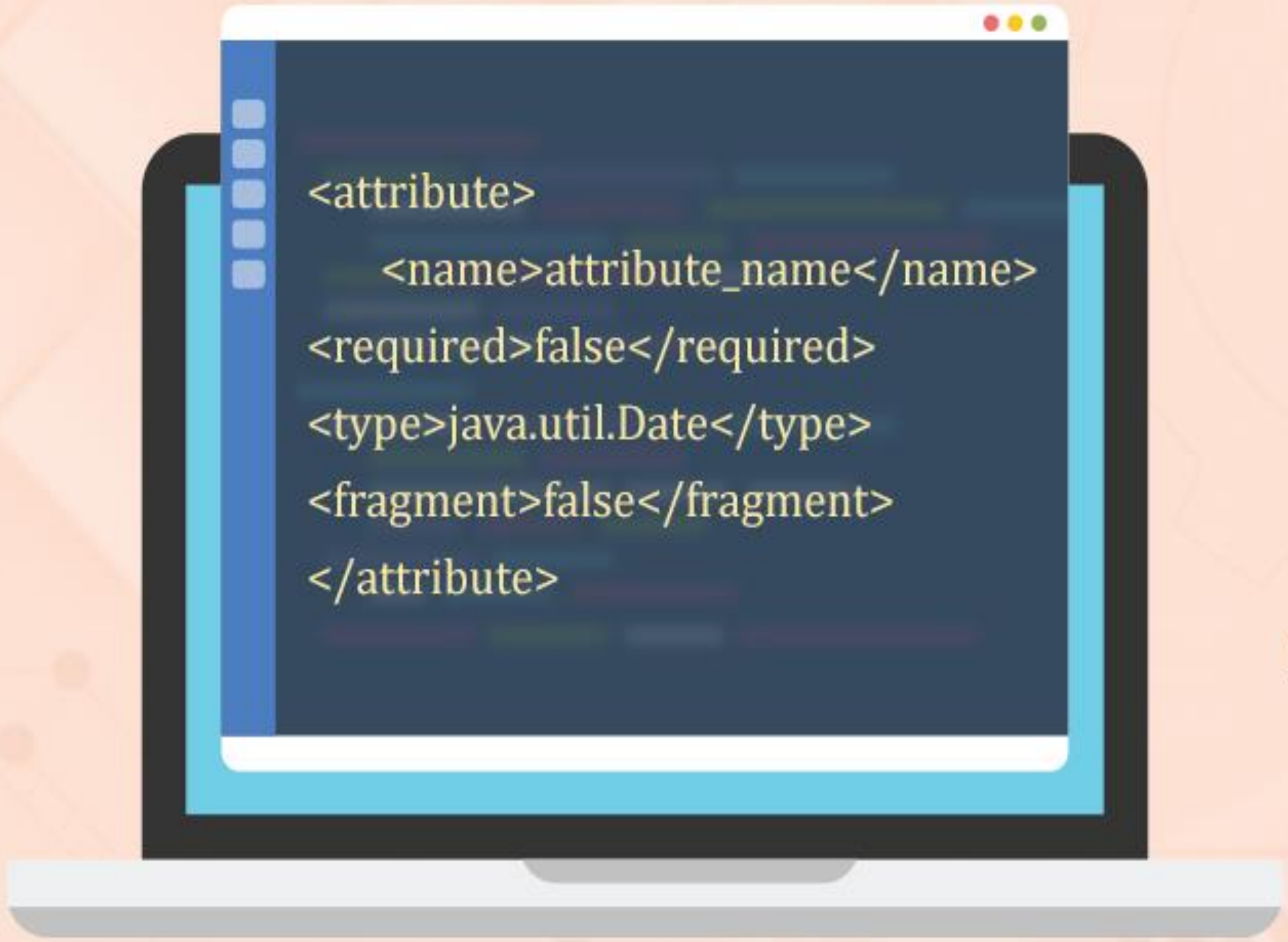
```
<%@ page errorPage="error.jsp" %>
<%
      int c=a/b;
      out.print("division of numbers is: "+c);
%>
```

- Error.jsp

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

<h3>Sorry an exception occured!</h3>

Exception is: <%= exception %>
```



```
<attribute>
    <name>attribute_name</name>
<required>false</required>
<type>java.util.Date</type>
<fragment>false</fragment>
</attribute>
```

Custom Tags in JSP

Custom Tags

- Custom tags are user-defined tags.
- They eliminates the possibility of scriptlet tag and separates the business logic from the JSP page.
- 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 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.

JSP Custom Tag API

- There are two ways to use the custom tag. They are given below:

  - Way 1:

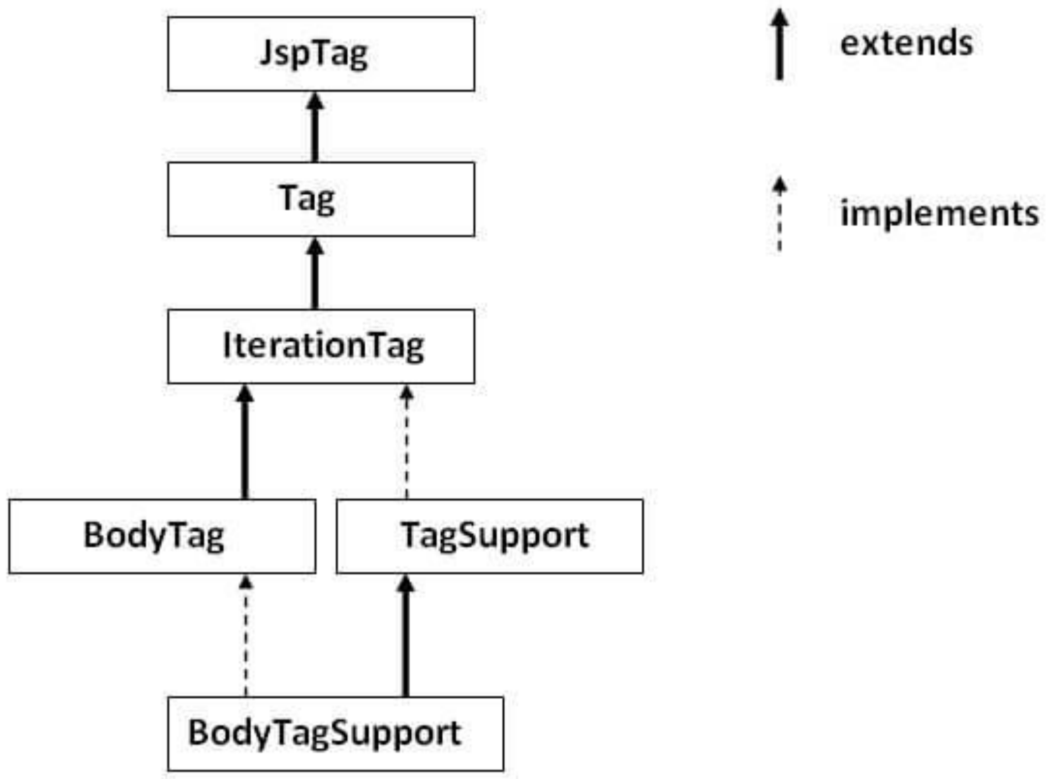
```
<prefix:tagname attr1= value1....attrn = valuen />
```

  - Way 2:

```
<prefix:tagname attr1=value1....attrn = valuen >
        body code
</prefix:tagname>
```

Custom Tag Syntax

- The javax.servlet.jsp.tagext package contains classes and interfaces for JSP custom tag API. The JspTag is the root interface in the Custom Tag hierarchy.



Fields of Tag interface

- public static int EVAL_BODY_INCLUDE
  - it evaluates the body content.
- public static int EVAL_PAGE
  - it evaluates the JSP page content after the custom tag.
- public static int SKIP_BODY
  - it skips the body content of the tag.
- public static int SKIP_PAGE
  - it skips the JSP page content after the custom tag.

Methods of Tag interface

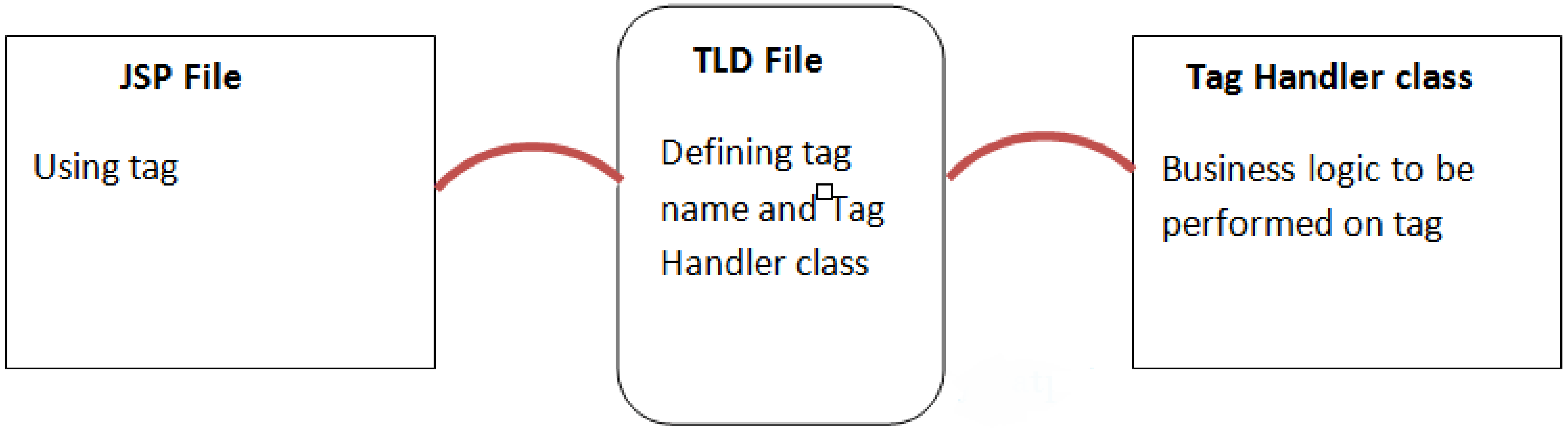
- public void setPageContext(PageContext pc)
  - it sets the given PageContext object.
- public void setParent(Tag t)
  - it sets the parent of the tag handler.
- public Tag getParent()
  - it returns the parent tag handler object.
- public int doStartTag()throws JspException
  - it is invoked by the JSP page implementation object. The JSP programmer should override this method and define the business logic to be performed at the start of the tag.

Methods of Tag interface

- public int doEndTag()throws JspException
  - it is invoked by the JSP page implementation object. The JSP programmer should override this method and define the business logic to be performed at the end of the tag.
- public void release()
  - it is invoked by the JSP page implementation object to release the state.

Understanding Flow of Custom Tag in JSP

- For creating any custom tag, we need to follow following steps:
  - Create the Tag handler class and perform action at the start or at the end of the tag.
  - Create the Tag Library Descriptor (TLD) file and define tags
  - Create the JSP file that uses the Custom tag defined in the TLD file


JSP File
Using tag
TLD File
Defining tag name and Tag Handler class
Tag Handler class
Business logic to be performed on tag

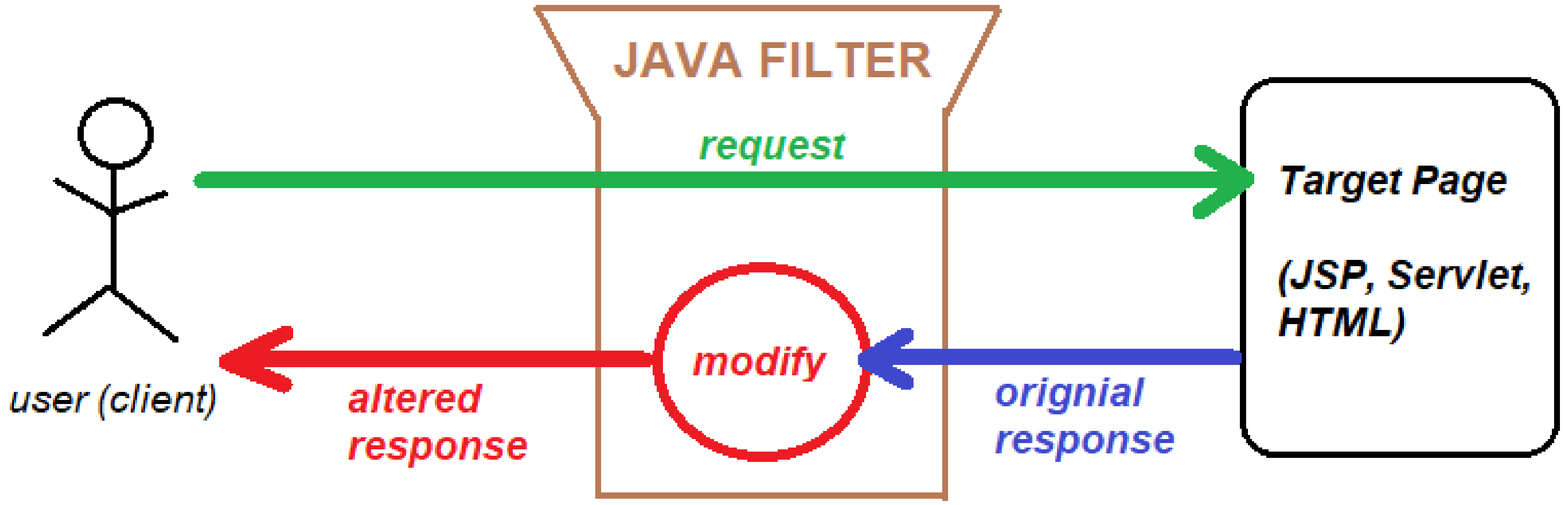
Redirect From One Page To Another In JSP

- By using this method, the server return back the response to the client, from where next request comes and it displays that url.

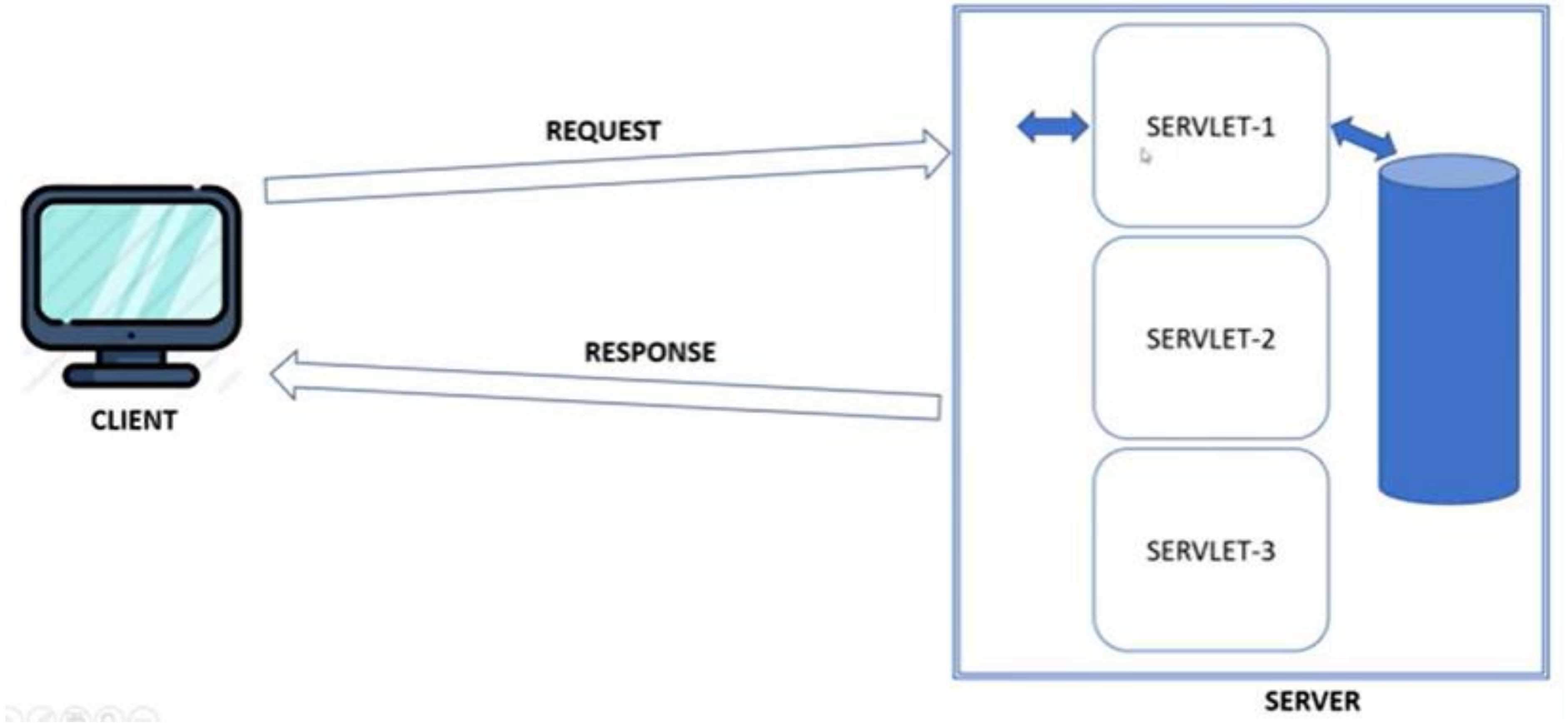
```
<%
    String redirectURL = "http://www.eiu.edu.vn/";
    response.sendRedirect(redirectURL);
%>
```

Servlet Filter

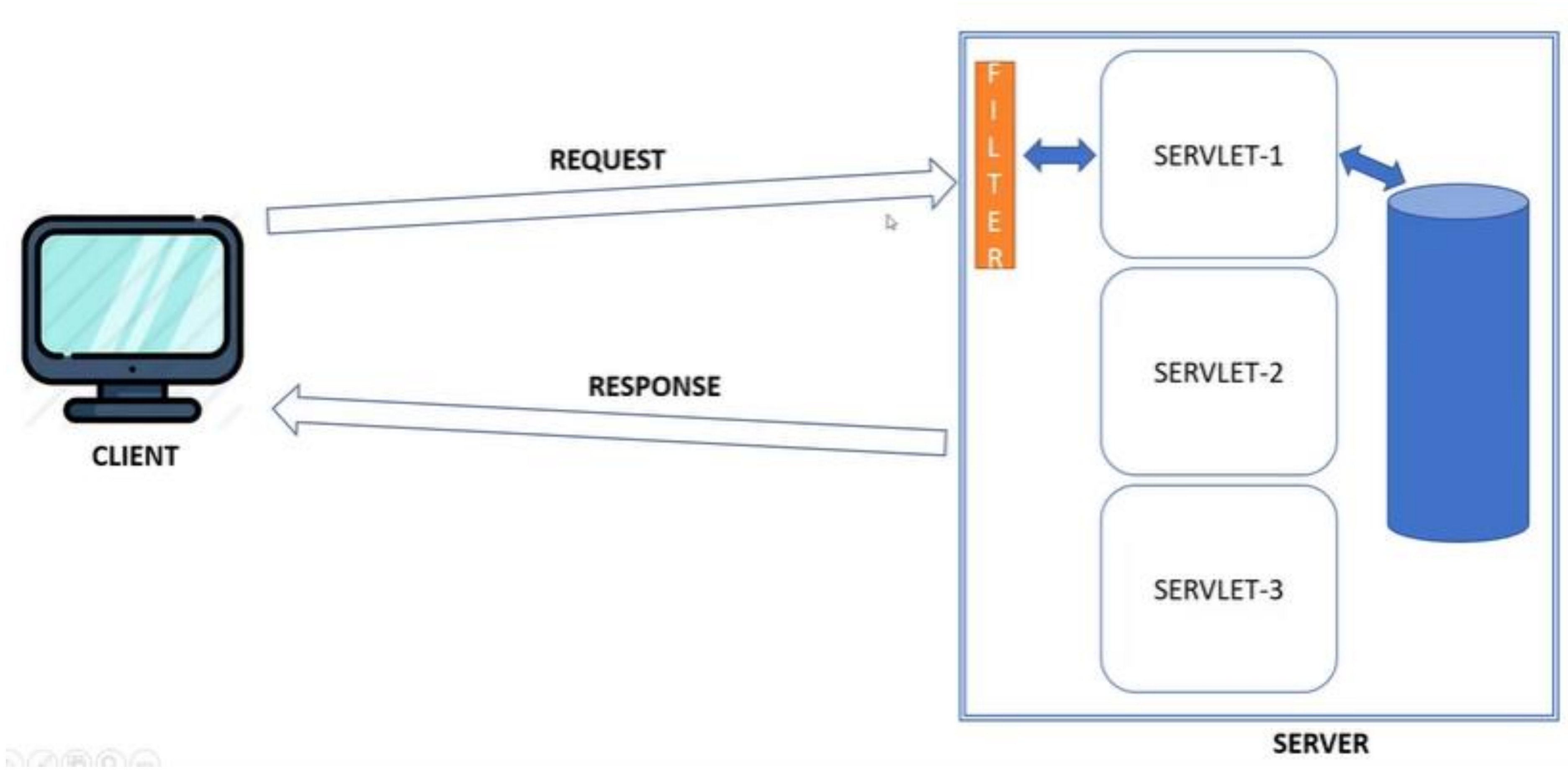
- A filter is an object that is invoked at the preprocessing and postprocessing of a request.
- It is mainly used to perform filtering tasks such as conversion, logging, compression, encryption and decryption, input validation etc.



Why Servlet Filter is required?



Why Servlet Filter is required?



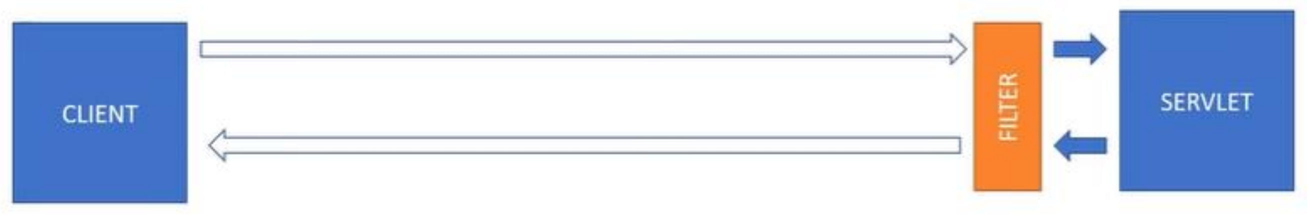
Servlet Filter

- The servlet filter is pluggable
- Its entry is defined in the web.xml file
- If we remove the entry of filter from the web.xml file, filter will be removed automatically and we don't need to change the servlet.

- ***Advantage of Filter***
  - ✓ Filter is pluggable.
  - ✓ One filter don't have dependency onto another resource.
  - ✓ Less Maintenance

Servlet Filter

- ***Usage of Filter***
  - ✓ Recording all incoming requests
  - ✓ Logs the IP addresses of the computers from which the requests originate
  - ✓ Conversion
  - ✓ Data compression
  - ✓ Encryption and decryption
  - ✓ Input validation etc.



- Authentication and authorization of request for resources.
- Formatting of request body or header before sending it to servlet.
- Compressing the response data sent to the client.
- Alter response by adding some cookies, header information etc.
- input validation etc.

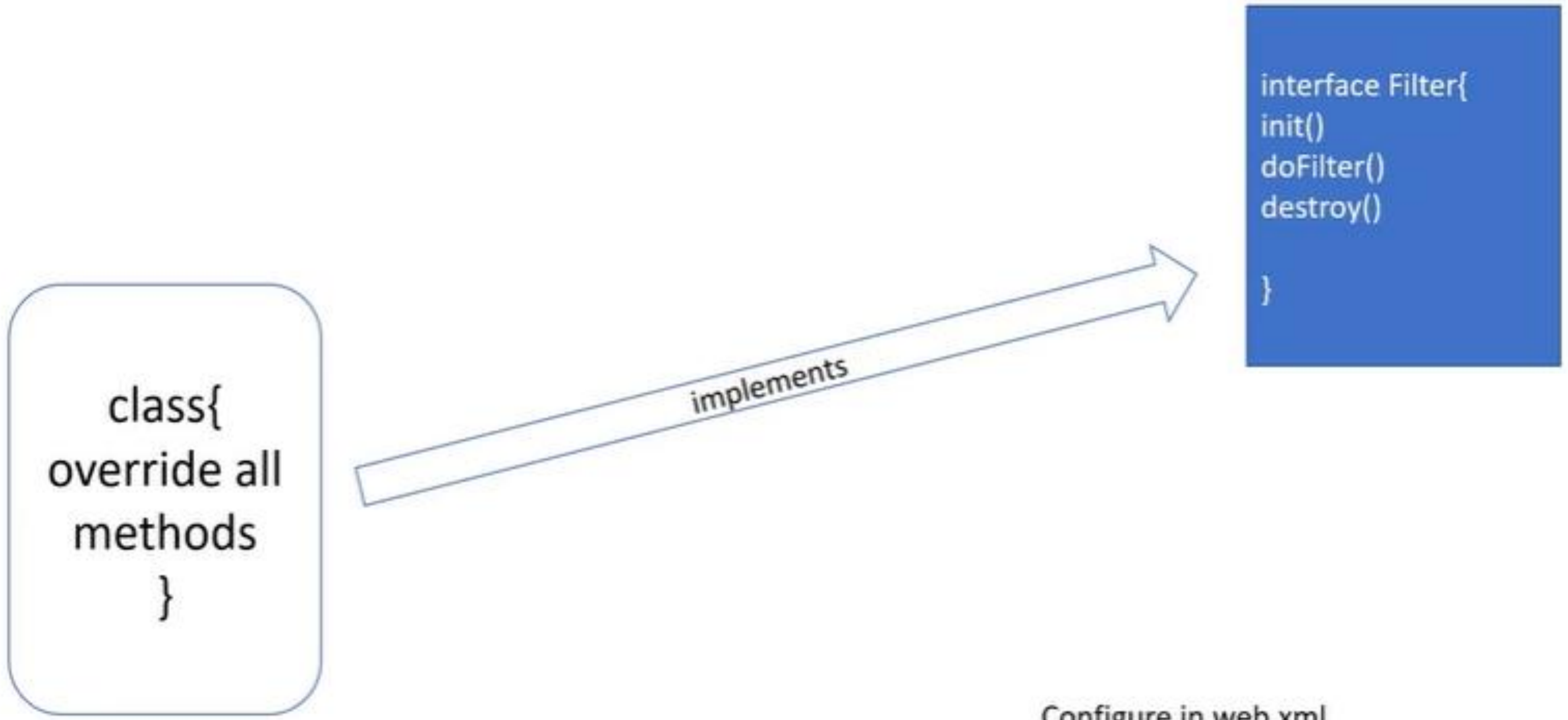
Filter API

- Like servlet filter have its own API. The javax.servlet package contains the three interfaces of Filter API.
  - Filter
    - ✓ Create a new filter by implementing this interface.
  - FilterChain
    - ✓ is responsible to invoke the next filter or resource in the chain.
  - FilterConfig

Life Cycle of a Filter

- public void init (FilterConfig config)
  - is invoked only once. It is used to initialize the filter.
- public void doFilter (HttpServletRequest request, HttpServletResponse response, FilterChain chain)
  - is invoked every time when user request to any resource, to which the filter is mapped. It is used to perform filtering tasks.
- public void destroy()
  - This is invoked only once when filter is taken out of the service.

How to Create and Configure Filter?



Configure in web.xml

How to Create and Configure Filter?

```
import javax.servlet.Filter;
import javax.servlet.FilterChain;
public class MyFilter implements Filter {
        public void doFilter(ServletRequest arg0, ServletResponse arg1, FilterChain arg2)
        throws IOException, ServletException {
        System.out.println("Before filter");
                //....Before Task
        arg2.doFilter(arg0, arg1);
        System.out.println("After filter");
                //....After Task
} }
```

How to define Filter in web.xml?

```
<web-app>
        <filter>
                <filter-name>...</filter-name>
                <filter-class>...</filter-class>
        </filter>
        <filter-mapping>
                <filter-name>...</filter-name>
                <url-pattern>...</url-pattern>
        </filter-mapping>
 </web-app>
```

JSTL

Java

www.educba.com

JSTL

- The JSP Standard Tag Library (JSTL) represents a set of tags to simplify the JSP development.
- The JavaServer Pages Standard Tag Library (JSTL) is a collection of useful JSP tags which encapsulates the core functionality common to many JSP applications.
- JSTL has support for common, structural tasks such as iteration and conditionals, tags for manipulating XML documents, internationalization tags, and SQL tags.
- It also provides a framework for integrating the existing custom tags with the JSTL tags.

Advantage of JSTL

- Fast Development JSTL provides many tags that simplify the JSP.
- Code Reusability We can use the JSTL tags on various pages.
- No need to use scriptlet tag It avoids the use of scriptlet tag.

Install JSTL Library

- To begin working with JSP tags you need to first install the JSTL library.
- If you are using the Apache Tomcat container, then follow these two steps −
  - Step 1: Download the binary distribution from Apache Standard Taglib and unpack the compressed file.
  - Step 2 : To use the Standard Taglib, simply copy the JAR files in the distribution's 'lib' directory to your application's webapps\ROOT\WEB-INF\lib directory.
- To use any of the libraries, you must include a <taglib> directive at the top of each JSP that uses the library.

Classification of The JSTL Tags

- The JSTL tags can be classified, according to their functions, into the following JSTL tag library groups that can be used when creating a JSP page −
  - Core Tags: The JSTL core tag provide variable support, URL management, flow control, etc.
  - Formatting Tags: The Formatting tags provide support for message formatting, number and date formatting, etc.
  - SQL Tags: The JSTL SQL tags provide SQL support.
  - XML Tags: The XML tags provide flow control, transformation, etc.
  - JSTL Functions Tags: The functions tags provide support for string manipulation and string length.

JSTL Core Tags

- The JSTL core tag provides variable support, URL management, flow control etc.
- Following is the syntax to include the JSTL Core library in your JSP –

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

- JSTL Core Tags List
  - c:out, c:import, c:set
  - c:remove, c:catch, c:if, c:choose, c:when, c:otherwise
  - c:forEach, c:forTokens, c:param, c:redirect, c:url

JSTL Core <c:out> Tag

- The <c:out> tag is similar to JSP expression tag, but it can only be used with expression.
- It will display the result of an expression, similar to the way < %=...% > work.
- The < c:out > tag automatically escape the XML tags.

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

JSTL Core <c:set> Tag

- It is used to set the result of an expression evaluated in a 'scope’.
- The <c:set> tag is helpful because it evaluates the expression and use the result to set a value of java.util.Map or JavaBean.
- This tag is similar to jsp:setProperty action tag.

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

JSTL Core <c:import> Tag

- The <c:import> is similar to jsp 'include', with an additional feature of including the content of any resource either within server or outside the server.
- This tag provides all the functionality of the <include > action and it also allows the inclusion of absolute URLs.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
  <body>
      <c:import var="data" url="http://www.eiu.edu.vn"/>
      <c:out value="${data}"/>
  </body>
</html>
```

JSTL Core <c:remove> Tag

- It is used for removing the specified variable from a particular scope.
- This action is not particularly helpful, but it can be used for ensuring that a JSP can also clean up any scope resources

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

JSTL Core <c:if> Tag

- The < c:if > tag is used for testing the condition and it display the body content, if the expression evaluated is true.
- It is a simple conditional tag which is used for evaluating the body content, if the supplied condition is true.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html> <body>
   <c:set var="income" scope="session" value="${1250*8}"/>
   <c:if test="${income > 8000}">
         <p>My income is: <c:out value="${income}"/><p>
   </c:if>
</body> </html>
```

JSTL Core <c:catch> Tag

- It is used for Catches any Throwable exceptions that occurs in the body and optionally exposes it.
- In general it is used for error handling and catches any exceptions that occurs in a program body.

```
<c:catch var ="catchtheException">
        <% int x = a/0;%>
</c:catch>
```

JSTL Core <c:catch> Tag

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html> <body>
    <c:catch var ="catchtheException">
        <% int x = 5/0;%>
    </c:catch>

   <c:if test = "${catchtheException != null}">
   <p>The type of exception is : ${catchtheException} <br />
          There is an exception: ${catchtheException.message}</p>
   </c:if>
</body> </html>
```

<c:choose>, <c:when>, <c:otherwise> Tags

- The c:when and c:otherwise works like if-else statement. But it must be placed inside c:choose tag
- < c:choose >: 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 >: is subtag of <choose > that will include its body if the condition evaluated be 'true’.
- < c:otherwise >: is also subtag of < choose > it follows &l;twhen > tags and runs only if all the prior condition evaluated is 'false'.

<c:choose>, <c:when>, <c:otherwise> Example

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html> body>
    <c:set var="income" scope="session" value="${4000*4}"/>
    <p>Your income is : <c:out value="${income}"/></p>
    <c:choose>
            <c:when test="${income <= 1000}">  Income is not good. </c:when>
            <c:when test="${income > 10000}">  Income is very good. </c:when>
             <c:otherwise> Income is undetermined...  </c:otherwise>
    </c:choose>
</body> </html>
```

JSTL Core <c:forEach>Tag

- The is an iteration tag used for repeating the nested body content for fixed number of times or over the collection.
- These tag used as a good alternative for embedding a Java while, do-while, or for loop via a scriptlet.

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

JSTL Core <c:forTokens>Tag

- The < c:forTokens > tag iterates over tokens which is separated by the supplied delimiters.
- It is used for break a string into tokens and iterate through each of the tokens to generate output.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html> <body>
            <c:forTokens items="Rahul-Nakul-Rajesh“ delims="-"
           var="name">
                        <c:out value="${name}"/><p>
            </c:forTokens>
</body> </html>
```

JSTL Core <c:url> Tag

- The <c:url> 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.
- The JSTL url tag is used as an alternative method of writing call to the response.encodeURL() method.

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

JSTL Core <c:param> Tag

- The < c:param > tag allow the proper URL request parameter to be specified within URL and it automatically perform any necessary URL encoding.
- Inside < c:param > tag, the value attribute indicates the parameter value and name attribute indicates the parameter name

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html> <body>
        <c:url value="/index1.jsp" var="completeURL"/>
                <c:param name="trackingId" value="786"/>
                <c:param name="user" value="Nakul"/>
        </c:url>
        ${completeURL}
</body> </html>
```

JSTL Core <c:redirect>Tag

- The < c:redirect > tag redirects the browser to a new URL. It supports the context-relative URLs, and the < c:param > tag.
- It is used for redirecting the browser to an alternate URL by using automatic URL rewriting

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html> <body>
            <c:set var="url" value="0" scope="request"/>
            <c:if test="${url= = 0}">
                        <c:redirect url="http://eiu.edu.vn"/>
            </c:if>
 </body> </html>
```

JSTL Function Tags

- The JSTL function provides a number of standard functions, most of these functions are common string manipulation functions.
- The syntax used for including JSTL function library in your JSP is:

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

- JSTL Function Tags List
  - fn:contains(), fn:containsIgnoreCase(), fn:endsWith(), fn:escapeXml(), fn:indexOf()
  - fn:trim(), fn:startsWith(), fn:split(), fn:toLowerCase(), fn:toUpperCase()
  - fn:substring(), fn:substringAfter(), fn:substringBefore(), fn:length(), fn:replace().

fn:containsIgnoreCase()

- This function is used to test if an input string contains the specified substring as a case insensitive way.
- During searching the specified substring it ignores the case

```
<html> <body>
    <c:set var="String" value="Welcome to java Course"/>
    <c:if test="${fn:containsIgnoreCase(String, 'java')}">
            <p> java string  is found <p>
    </c:if>
    <c:if test="${fn:containsIgnoreCase(String, ‘JAVA')}">
            <p> JAVA string  is found <p>
    </c:if>
</body> </html>
```

fn:endsWith()

- This function is used for testing if an input string ends with the specified suffix.
- If the string ends with a specified suffix, it returns true otherwise false.

```
<html> <body>
            <c:set var="String" value="Welcome to java course"/>
            <c:if test="${fn:endsWith (String, ‘course')}">
                        <p> string  is ends with course <p>
            </c:if>
            <c:if test="${fn:endsWith (String, ‘java')}">
            <p> string  is ends with java <p>
    </c:if>
</body> </html>
```

fn:escapeXml()

- This function escapes the characters that would be interpreted as XML markup.
- It is used for escaping the character in XML markup language.

```
<html> <body>
        <c:set var="string1" value="Welcome to EIU."/>
        <c:set var="string2" value="It is <xyz>java course.</xyz>"/>
        <p>With escapeXml() Function:</p>
        <p>string-1 : ${fn:escapeXml(string1)}</p>
        <p>string-2 : ${fn:escapeXml(string2)}</p>
        <p>Without escapeXml() Function:</p>
        <p>string-1 : ${string1}</p>
        <p>string-2 : ${string2}</p>
</body> </html>
```

fn:indexOf()

- This function return an index of string.
- It is used for determining the index of string specified in substring.

```
<html> <body>
    <c:set var="string1" value="Welcome to java."/>
    <c:set var="string2" value="It is <xyz>java course. </xyz>"/>
      <p>Index-1 : ${fn:indexOf(string1, "java")}</p>
    <p>Index-2 : ${fn:indexOf(string2, "java")}</p>
</body> </html>
```

fn:length()

- This function returns the number of characters inside a string, or the number of items in a collection.
- It is used for calculating the length of string and to find out the number of elements in a collection.

```
<html> <body>
   <c:set var="str1" value="Welcome to java    programming    "/>
   <p>String Length is : ${fn:length(str1)}</p>
</body> </html>
```

fn:trim()

- This function removes the blank spaces from both the ends of a string.
- It mainly used for ignoring the blank spaces from both the ends of string

```
<html> <body>
    <c:set var="str1" value="Welcome to java    programming    "/>
    <p>String-1 Length is : ${fn:length(str1)}</p>


            <c:set var="str2" value="${fn:trim(str1)}" />
            <p>String-2 Length is : ${fn:length(str2)}</p>
            <p>Final value of string is : ${str2}</p>
</body> </html>
```

fn:startsWith()

- This function is used for returning a Boolean value.
- It gives the true result when the string is started with the given prefix otherwise it returns a false result

```
<html>   <body>
    <c:set var="msg" value="The Example of JSTL
                                                  fn:startsWith() Function"/>
    The    string    starts    with    "The":    ${fn:startsWith(msg,    'The')}
<br>
   The string starts with "Example": ${fn:startsWith(msg,  'Example')}
</body> </html>
```

fn:split()

- This function is used for returning a Boolean value.
- It gives the true result when the string is started with the given prefix otherwise it returns a false result.

```
<html>   <body>
        <c:set var="str1" value="Welcome-to-JSP-Programming."/>
        <c:set var="str2" value="${fn:split(str1, '-')}" />


        <p>String-2-0 : ${str2[0]}</p>
        <p>String-2-2 : ${str2[2]}</p>
</body> </html>
```

fn:join()

- This function join the strings into a strings.

```
<html>   <body>
            <c:set var="str1" value="Welcome-to JSP Programming."/>
            <p>String-1 : ${str3}</p>
            <c:set var="str2" value="${fn:split(str1, '-')}" />
            <p>String-2 : ${str3}</p>
            <c:set var="str3" value="${fn:join(str2, ' * ')}" />
            <p>String-3 : ${str3}</p>
            <c:set var="str4" value="${fn:split(str3, ' ')}" />
            <p>String-4 : ${str3}</p>
            <c:set var="str5" value="${fn:join(str4, '--')}" />
            <p>String-5 : ${str5}</p>
</body> </html>
```

fn:toLowerCase()

- The fn:toLowerCase() function converts all the characters of a string to lower case.
- It is used for replacing any upper case character in the input string with the corresponding lowercase character.

```
<html>   <body>
        <c:set var="string“ value="Welcome to Java Programming"/>
         ${fn:toLowerCase("HELLO,")}
         ${fn:toLowerCase(string)}
</body> </html>
```

fn:toUpperCase()

- This function converts all the characters of a string to upper case.
- It is used for replacing any lower case character in the input string with the corresponding upper case character

```
<html>   <body>
        <c:set var="string“ value="Welcome to Java Programming"/>
         ${fn:toUpperCase("Hello,")}
         ${fn:toUpperCase(string)}
</body> </html>
```

fn:substring()

- This function returns the subset of a string.
- It is used to return the substring of given input string according to specified start and end position.

```
<html>   <body>
        <c:set var="string" value="Welcome to Java Programming"/>
        <c:set var="sub" value="${fn:substring(string, 5, 17)}"/>
        <p>Substring is ${sub}</p>
</body> </html>
```

fn:substringAfter()

- This function returns the subset of string followed by a specific substring.
- It returns the part of string which lies after the provided string value.

```
<html>   <body>
        <c:set var="string" value="Welcome to Java Programming"/>
        <c:set var="sub" value="${fn:substringAfter(string, 'to')}"/>
        <p>Substring is ${sub}</p>
</body> </html>
```

fn:substringBefore()

- This function returns the subset of string before a specific substring.
- It is used for returning a part of original string which lies before the specified string value.

```
<html>   <body>
        <c:set var="string" value="Welcome to Java Programming"/>
        <c:set var="sub" value="${fn:substringBefore(string, 'to')}"/>
        <p>Substring is ${sub}</p>
</body> </html>
```

fn:replace()

- This function replaces all the occurrence of a string with another string sequence.
- It search in an input string and replace it with the provided string.

```
<html>   <body>
        <c:set var="string" value="Welcome to Java Programming"/>
             ${fn:replace(string, "Java", ".Net")}
                 <c:set var="sub" value="${fn:replace(string, 'a', 'e')} "/>
             <p>Substring is ${sub}</p>
</body> </html>
```

JSTL Formatting Tags

- It provides support for message formatting, number and date formatting etc.
- The syntax used for including JSTL formatting library in your JSP is:

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

- JSTL Formatting Tags List
  - fmt:parseNumber(), fmt:timeZone(), fmt:formatNumber()
  - fmt:parseDate(), fmt:bundle(), fmt:setTimeZone(), fmt:setBundle(), fmt:message(), fmt:formatDate().

<fmt:parseNumber>

- This tag is used to Parses the string representation of a currency, percentage, or number.
- It is based on the customized formatting pattern.

```
<html>   <body>
       <c:set var="Amount" value="786.970" />
    <fmt:parseNumber var="j" type="number" value="${price}" />
    <p><i>Price is:</i>  <c:out value="${j}" /></p>
     <fmt:parseNumber   var="j"   integerOnly="true"   type="number"
            value="${price}" />
    <p><i> Price is:</i>  <c:out value="${j}" /></p>
</body> </html>
```

<fmt:timeZone>

- It specifies the parsing action nested in its body or the time zone for any time formatting.
- It is used for specify the time zone information used for time formatting operations.

```
<html>   <body>
    <c:set var="str" value="<%=new java.util.Date()%>" />
    <table border="2" width="100%">
    <tr> <td width="100%" colspan="2" bgcolor="#FF7F50">
    <p align="center"> <b> <font color="#000000" size="6">
    Formatting:<fmt:formatDate value="${str}" type="both"
                                timeStyle="long" dateStyle="long" />
     </font> </b</p</td> </tr>
</body> </html>
```

<fmt:formatNumber>

- This tag is used to format the numerical value using the specific format or precision.
- It is used to format percentages, currencies, and numbers according to the customized formatting pattern.

```
<html>   <body>
            <c:set var="Amount" value="9850.14115" />
            <p> Formatted Number-1:
            <fmt:formatNumber value="${Amount}" type="currency" /></p>
            <p>Formatted Number-2:
            <fmt:formatNumber type="number" groupingUsed="true" value="${Amount}" /></p>
            <p>Formatted Number-3:
            <fmt:formatNumber type="number" maxIntegerDigits="3" value="${Amount}" /></p>
```

<fmt:formatNumber>

```
        <p>Formatted Number-4:
        <fmt:formatNumber type="number" maxFractionDigits="6" value="${Amount}" /></p>
        <p>Formatted Number-5:
        <fmt:formatNumber type="percent" maxIntegerDigits="4" value="${Amount}" /></p>
        <p>Formatted Number-6:
        <fmt:formatNumber type="number" pattern="###.###$" value="${Amount}" /></p>
</body></html>
```

<fmt:parseDate>

- This <fmt:parseDate> tag parses the string representation of a time and date.
- It is used to format the time and date according to a customized formatting pattern.

```
<html>   <body>
            c:set var="date" value="12/08/2016" />
            <fmt:parseDate value="${date}" var="parsedDate"  pattern="dd/mm/yyyy" />
            <p><c:out value="${parsedDate}" /></p>
</body> </html>
```

<fmt:message>

- This tag is used for displaying an internationalized message.
- It maps the key of localized message to return the value using a resource bundle specified in the bundle attribute.

```
package com.javatpoint;
import java.util.ListResourceBundle;
public class Message extends ListResourceBundle {
    public Object[][] getContents() {  return contents;  }
    static final Object[][] contents = { { "vegetable.Potato", "Potato" }, { "vegetable.Tomato",
                                                   "Tomato" }, { "vegetable.Carrot", "Carrot" }, };
}
```

<fmt:message>

```
<html> <body>
            <fmt:setBundle basename="com.javatpoint.Message"     var="lang"/>
            <fmt:message key="vegetable.Potato"    bundle="${lang}"/>
            <br/>
            <fmt:message key="vegetable.Tomato“  bundle="${lang}"/>
            <br/>
            <fmt:message key="vegetable.Carrot"    bundle="${lang}"/>
 </body> </html>
```

<fmt:bundle>

- This tag loads the resource bundle which is used by its tag body.
- This tag will make the specified bundle available for all <fmt:message> tags that occurs between the boundary of <fmt:bundle> and </fmt:bundle> tags.
- It is used to create the ResourceBundle objects which will be used by their tag body.
- The syntax used for including the <fmt:bundle> tag is:

```
<fmt:bundle basename="Resource Bundle Name" prefix="msg"> body content </fmt:bundle>
```

<fmt:bundle>

```
// Simple.java
package com.javatpoint;
import java.util.ListResourceBundle;
public class Simple extends ListResourceBundle {
        public Object[][] getContents() {  return contents; }
        static final Object[][] contents = { { "colour.Violet", "Violet" }, {
                        "colour.Indigo", "Indigo" }, { "colour.Blue", "Blue" }, };
}
```

<fmt:bundle>

```
<html> <body>
          <fmt:bundlebasename = "com.javatpoint.Simple"          prefix="colour.">
                    <fmt:message key = "Violet"/><br/>
                    <fmt:message key = "Indigo"/><br/>
                    <fmt:message key = "Blue"/><br/>
          </fmt:bundle>
  </body>  </html>
```

<fmt:setBundle>

- This <fmt:setBundle> tag is used to load the resource bundle and store their value in the bundle configuration variable or the name scope variable.
- It is used for creating the ResourceBundle object which will be used by tag body.

```
package com.javatpoint;
import java.util.ListResourceBundle;
public class Main extends ListResourceBundle {
    public Object[][] getContents() { return contents; }
            static final Object[][] contents = { { "vegetable.Potato", "Potato" }, { "vegetable.Tomato",
                                                                    "Tomato" }, { "vegetable.Carrot", "Carrot" }, };
}
```

<fmt:bundle>

```
<html>
        <body>
                <fmt:setBundle basename="com.javatpoint.Main" var="lang"/>
                <fmt:message key="vegetable.Potato" bundle="${lang}"/> <br/>
                <fmt:message key="vegetable.Tomato" bundle="${lang}"/> <br/>
                <fmt:message key="vegetable.Carrot"    bundle="${lang}"/> <br/>
        </body>
</html>
```

<fmt:setTimeZone>

- This tag store the time zone inside a time zone configuration variable.
- It is used for copy a time zone object inside a specified scope variable.

```
<html> <body>
        <c:set var="date" value="<%= new java.util.Date() %>" />
        <p> <b> Date and Time in Vietnam Standard Time(VST) Zone: </b>
         <fmt:formatDate value="${date}"  type="both"  timeStyle="long“ dateStyle="long" />
        </p>
        <fmt:setTimeZone value="GMT-17" />
        <p> <b> Date and Time in GMT-17 time Zone: </b>
        <fmt:formatDate value="${date}" type="both"   timeStyle="long" dateStyle="long" /></p>
</body> </html>
```

<fmt:formatDate>

- This tag is used for different formats of date and time using the supplied pattern and styles.
- It is used to format the time and date according to the customized formatting pattern

```
<html> <body>
            <c:set var="Date" value="<%=new java.util.Date() %>" />
            <p>  Formatted Time :   <fmt:formatDate type = "time" value = "${Date}" />  </p>
            <p>  Formatted Date :    <fmt:formatDate type = "date" value = "${Date}" />  </p>
            <p>  Formatted Date and Time :  <fmt:formatDate type = "both" value="${Date}" </p>
             <p> Formatted Date and Time in short style :  <fmt:formatDate type = "both”
                                                       dateStyle = "short" timeStyle = "short"  value = "${Date}" />  </p>
```

<fmt:formatDate>

```
        <p>  Formatted Date and Time in medium style : <fmt:formatDate type="both"
                         dateStyle="medium"     timeStyle="medium"  value="${Date}" /> </p>

        <p>  Formatted Date and Time in long style :  <fmt:formatDate type="both"
                                    dateStyle="long" timeStyle="long"  value="${Date}"/>  </p>

</body> </html>
```

JSTL XML tags

- 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.
- The url for the xml tags is http://java.sun.com/jsp/jstl/xml and prefix is x.
- The JSTL XML tag library has custom tags used for interacting with XML data.
- The syntax used for including JSTL XML tags library in your JSP is:

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

JSTL XML tags

- Before you proceed further with the examples, you need to copy the two XML and XPath related libraries into the <Tomcat Installation Directory>\lib:

- Xalan.jar: Download this jar file from the link: http://xml.apache.org/xalan-j/index.html

- XercesImpl.jar: Download this jar file from the link: http://www.apache.org/dist/xerces/j/

<x:out>

- This tag is used for displaying the result of an xml Path expression and writes the result to JSP writer object.
- It is similar to the scriptlet tag <%= %> used in JSP

```
html>  <body>
          <h2>Vegetable Information:</h2>
          <c:set var="vegetable">
          <vegetables>
                    <vegetable>  <name>onion</name>  <price>40/kg</price> </vegetable>
                    <vegetable> <name>Potato</name>  <price>30/kg</price> </vegetable>
                    <vegetable> <name>Tomato</name>     <price>90/kg</price> </vegetable>
          </vegetables>
```

<x:out>

```
</c:set>
            <x:parse xml="${vegetable}" var="output"/>
                        <b>Name of the vegetable is</b>:
            <x:out select="$output/vegetables/vegetable[1]/name" /> <br>
                        <b>Price of the Potato is</b>:
            <x:out select="$output/vegetables/vegetable[2]/price" />
</body>  </html>
```

<x:parse>

- This tag is used for parse the XML data specified either in the tag body or an attribute.
- It is used for parse the xml content and the result will stored inside specified variable.

<x:parse>

novels.xml file:

```
<books>
        <book>
                <name>Three mistakes of my life</name>
                <author>Chetan Bhagat</author>
                <price>200</price>
        </book>
        <book>
                <name>Tomorrow land</name>
                <author>NUHA</author>
                <price>2000</price>
        </book>
</books>
```

<x:parse>

```
<html>  <body>
            <h2>Books Info:</h2>
            <c:import var="bookInfo" url="novels.xml"/>
            <x:parse xml="${bookInfo}" var="output"/>
            <p> First Book title: <x:out
            select="$output/books/book[1]/name" /> </p>
            <p> First Book price: <x:out
            select="$output/books/book[1]/price" /> </p>
            <p> Second Book title: <x:out
            select="$output/books/book[2]/name" /> </p>
            <p> Second Book price: <x:out
            select="$output/books/book[2]/price" /> </p>
  </body> </html>
```

<x:set>

- This tag is used to set a variable with the value of an XPath expression.
- It is used to store the result of xml path expression in a scoped variable.

```
<html>  <body>
            <h2>Books Information:</h2>
            <c:set var="book">
            <books>
                        <book>
                                    <name>Three mistakes of my life</name>
                                    <author>Chetan Bhagat</author>
                                    <price>200</price>
                        </book
```

<x:set>

```
                    <book>
                              <name>Tomorrow land</name>
                              <author>Brad Bird</author>
                              <price>2000</price>
                    </book>
          </books>
</c:set>
<x:parse xml="${book}" var="output"/>
          <x:set var="fragment" select="$output/books/book[2]/price"/>
          <b>The price of the Tomorrow land book</b>:
<x:out select="$fragment" />
</body></html>
```

XML <x:choose>, <x:when>, <x:otherwise>

- <x:choose>:
  - This tag 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.
- <x:when>:
  - is subtag of <x:choose> that will include its body if the condition evaluated be 'true'.
- <x:otherwise>:
  - This is also subtag of <x:choose> it follows <x:when> tags and runs only if all the prior condition evaluated is 'false’.
- The <x:when> and <x:otherwise> works like if-else statement. But it must be placed inside <x:choose> tag.

XML <x:choose>, <x:when>, <x:otherwise>

```
<html>  <body> <h3>Books Information:</h3>

            <c:set var="xmltext">

                        <books>

                                    <book>

                                                <name>Three mistakes of my life</name>
                                                <author>Chetan Bhagat</author>

                                                <price>200</price>

                                    </book>

                                    <book>

                                                <name>Tomorrow land</name>

                                                <author>Brad Bird</author>

                                                <price>2000</price>

                                    </book>
```

XML <x:choose>, <x:when>, <x:otherwise>

```
                </books>
        </c:set>
        <x:parse xml="${xmltext}" var="output"/>
        <x:choose>
                <x:when select="$output//book/author = 'Chetan bhagat’”>
                        Book is written by Chetan bhagat  </x:when>
                <x:when select="$output//book/author = 'Brad Bird’”>
                        Book is written by Brad Bird </x:when>
                <x:otherwise>  The author is unknown...  </x:otherwise>
        </x:choose>
</body> </html>
```

<x:if>

- The <x:if> tag is used for evaluating the test XPath expression.
- It is a simple conditional tag which is used for evaluating its body if the supplied condition is true.

```
<html> <body> <h2>Vegetable Information:</h2>
            <c:set var="vegetables">
                        <vegetables>
                                    <vegetable>
                                                <name>onion</name>
                                                <price>40</price>
                                    </vegetable>
                                    <vegetable>
                                                <name>Potato</name>
```

<x:if>

```
                        <price>30</price>
                </vegetable>
                <vegetable>
                        <name>Tomato</name>
                        <price>90</price>
                </vegetable>
        </vegetables>
    </c:set>
    <x:parse xml="${vegetables}" var="output"/>
    <x:if select="$output/vegetables/vegetable/price < 100">
        Vegetables prices are very low. </x:if>
</body> </html>
```

<x:transform>

- This tag is used in a XML document for providing the XSL (Extensible Stylesheet Language) transformation.
- It is used for transforming xml data based on XSLT script.

<u>transfer.xsl file:</u>

```
<xsl:stylesheet version="1.0"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
        <xsl:param name="doc"/>
                <xsl:template match="/">
                        <html> <body>
                                <h2>Company's Employee detail</h2>
```

<x:transform>

```
                    <table border="2">
                          <tr> <th align="left">Name  </th>
                          <th align="left">Designation  </th>
                          <th align="left">Age  </th>  </tr>
                    <xsl:for-each select="organisation/company/emp">
                          <tr>
                                <td>  <xsl:value-of select="name"/>  </td>
                                <td>  <xsl:value-of select="designation"/>  </td>
                                <td>  <xsl:value-of select="age"/>  </td>
                          </tr>
                    </xsl:for-each>
</table>  </body> </html>  </xsl:template>  </xsl:stylesheet>
```

<x:transform>

transfer.xml:

```
<organization> <company>
            <emp>
                        <name>Rajan Singh</name>
                        <designation>Business Developer</designation>
                        <age>40</age>
            </emp>
            <emp>
                        <name>Supriya Gaur</name>
                        <designation>HR Executive</designation>
                        <age>22</age>
            </emp> </company>
```

<x:transform>

```
        <company>
                <emp>
                        <name>Shashnak Singhal</name>
                        <designation>Sr. Java Programmer</designation>
                        <age>26</age>
                </emp>
                <emp>
                        <name>Hemant Kishor</name>
                        <designation>Sr. PHP Programmer</designation>
                        <age>23</age>
                </emp>
</company> </organization> >
```

<x:transform>

Index.jsp:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<html> <head>
      <title>x:transform Tag</title>
</head>  </html>

<c:import var="xml" url="transfer.xml" />
<c:import var="xsl" url="transfer.xsl" />

<x:transform xml="${xml}" xslt="${xsl}" />
```

<x:param>

- This tag is used to set the parameter in the XSLT style sheet.
- It use along with the transform tag for sending parameter along with the value.

transfer.xsl file:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="html" indent="yes"/>
<xsl:param name="bgColor"/>
<xsl:template match="/">
            <html>  <body>
                        <xsl:apply-templates/>
            </body> </html>
</xsl:template>
```

<x:param>

```
<xsl:template match="books">
            <table border="1" width="60%" bgColor="{$bgColor}">
            <xsl:for-each select="book">
            <tr>
                        <td>  <b><xsl:value-of select="name"/></b>  </td>
                        <td>       <xsl:value-of select="author"/>       </td>
                        <td>       <xsl:value-of select="price"/>        </td>
            </tr>
            </xsl:for-each>
        </table>
    </xsl:template>
</xsl:stylesheet>
```

<x:param>

index.jsp

```
<html>  <body>
            <h3>Novels Information:</h3>
            <c:set var="xmltext">
            <books>
                        <book>
                                    <name>Three mistakes of my life</name>
                                    <author>Chetan Bhagat</author>
                                    <price>200</price>
                        </book>
```

<x:param>

```
                    <book>
                              <name>Tomorrow land</name>
                              <author>Brad Bird</author>
                              <price>1000</price>
                    </book>
                    <book>
                              <name>Wings of fire</name>
                              <author>Dr. APJ Abdul Kalam</author>
                              <price>500</price>
                    </book>
          </books>
</c:set>
```

<x:param>

```
  <c:import url="transfer.xsl" var="xslt"/>
<x:transform xml="${xmltext}" xslt="${xslt}">
            <x:param name="bgColor" value="yellow"/>
</x:transform>
</body> </html>
```

JSTL SQL Tags

- The JSTL sql tags provide SQL support.
- The SQL tag library allows the tag to interact with RDBMSs such as Microsoft SQL Server, mySQL, or Oracle.
- The syntax used for including JSTL SQL tags library in your JSP is:

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

- JSTL Function Tags List
  - sql:setDataSource, sql:query, sql:update, sql:param, sql:dateParam, sql:transaction.

<sql:setDataSource>

- This tag is used for creating a simple data source suitable only for prototyping.
- It is used to create the data source variable directly from JSP and it is stored inside a scoped variable.
- It can be used as input for other database actions.

```
<html>
<body>
        <sql:setDataSource var = "db"   driver = "com.mysql.jdbc.Driver"
                url = "jdbc:mysql://localhost/test“    user="root”     password="1234"/>
</body>
</html>
```

<sql:query>

- This tag is used for executing the SQL query defined in its sql attribute or the body.
- It is used to execute an SQL SELECT statement and saves the result in scoped variable.

```
<html>
<body>
            <sql:setDataSource var="db"   driver="com.mysql.jdbc.Driver"
                        url="jdbc:mysql://localhost/test“    user="root"   password="1234"/>
            <sql:query dataSource="${db}" var="rs">
                        SELECT * from Students;
            </sql:query>
</body>
</html>
```

<sql:update>

- This tag is used for executing the SQL DML query defined in its sql attribute or in the tag body.
- It may be SQL UPDATE, INSERT or DELETE statements.

```
<html>  <body>
            <sql:setDataSource var="db"   driver="com.mysql.jdbc.Driver"
                        url="jdbc:mysql://localhost/test“     user="root"   password="1234"/>
            <sql:update dataSource="${db}" var="count">
                        insert into students values (154,'nasreen', 'jaha', 25);
            </sql:update>
</body>  </html>
```

<sql:param>

- This tag sets the parameter value in SQL statement.
- It is used as nested tag for <sql:update> and <sql:query> to provide the value in SQL query parameter.
- If null value is provided, the value is set at SQL NULL for value attribute.

```
<html>  <body>

            <sql:setDataSource var="db"   driver="com.mysql.jdbc.Driver"
                        url="jdbc:mysql://localhost/test“     user="root“   password="1234"/>

            <c:set var="StudentId" value="152"/>

            <sql:update dataSource="${db}" var="count">  delete from students where id = ?

            <sql:param value="${StudentId}" />  </sql:update>

</body>  </html>
```

<sql:dateParam>

- This is used to set the specified date for SQL query parameter.
- It is used as nested tag for <sql:update> and <sql:query> to provide the date and time value for SQL query parameter.
- If null value is provided, the value is set at SQL NULL

```
<html>  <body>
        <sql:setDataSource var="db"   driver="com.mysql.jdbc.Driver"
                url="jdbc:mysql://localhost/test“    user="root"   password="1234"/>
        <%  Date DoB = new Date("2000/10/16"); int studentId = 151;  %>
        <sql:update dataSource="${db}" var="count">
                update student set DoB = ? where id = ?
                <sql:dateParam value="<%=DoB%>" type="DATE" />
```

<sql:dateParam>

```
                    <sql:param value="<%=studentId%>" />
          </sql:update>
</body>  </html>
```

<sql:transaction>

- This tag is used for transaction management. It is used to group multiple <sql:update> into common transaction.
- If you group multiple SQL queries in a single transaction, database is hit only once.
- It is used for ensuring that the database modifications are performed by the nested actions which can be either rolled back or committed.

<sql:transaction>

```
<sql:transaction dataSource="${db}">
        <sql:update var="count">
                update student set first_name = 'suraj' where id = 150
        </sql:update>
        <sql:update var="count">
                update student set last_name= 'saifi' where id = 153
        </sql:update>
        <sql:update var="count">
                insert into student values (154,'supriya', 'jaiswal', '1995/10/6');
        </sql:update>
</sql:transaction>
```

JDBC in JSP

```
<%@ page import="java.sql.*" %>


        <%      String dbURL = "jdbc:mysql://localhost:3306/ ";
                String dbName = "debarshi";
                String username = "debarshi";
                String password = "root";
                String url = dbURL + dbName;
                String sql = "select * from student";
        %>
```

JDBC in JSP

```
<%
        Class.forName("com.mysql.jdbc.Driver");
        Connection con = DriverManager.getConnection(url, username, password);
        Statement st= con.createStatement();
        ResultSet rs = st.executeQuery(sql);
        //rs.next();
        while(rs.next())
        {
                String roll = rs.getString(1);
                String name =  rs.getString(2)
        }
%>
```