UNIT – IV

"Java Server Pages"

Date
Page

✓ Java Server Pages (JSP) is a Server-Side Programming technology that enable the creation of dynamic, platform-independant method for building Web-based applications.

✓ JSP has access to complete Java APIs, JDBC etc

Introduction to JSP

- JSP is a Server side programing technology for building dynamic web pages.
- It provides more functionality than Servlets.
- A JSP file consist of HTML tags & JSP tags.
- A JSP file is saved with file extension ".jsp".
- Helps developers insert Java code in HTML pages by using Special JSP tags.

The Problems With Servlets.

(1). A Single servlet class has to perform tasks such as
- ✓ Accepting the request
- ✓ Processing of request
- ✓ Handling the logic
- ✓ Generation of (HTML) response.

(2). For developing web based application, The developer must have knowledge of Java as well as HTML code.

(3). If a small change in the look and feel of web based application, the whole application is recompiled, configured and executed.

(4) Servlets do not support the use of web based tools for application development. With this applying complete styles & look & feel is more complex, time consuming & prone to errors.

```
pw.println("<B> welcome </B>");
```
(Macromedia flash | Dream viewer)

All these are problems associated with Servlets because of single servlet to handle all tasks.

These problems are resolved using JSP Technology.

The Anatomy of JSP page (Structue)

JSP page is a simple web page which contains

① Template Text - It can be Script code such as HTML, CSS, XML or a simple text

② JSP elements - Responsible for dynamic Content. The various jsp elements can be action tages, custom tags, directives, JSTL library elements etc. (JSTL - Java Server Pages Standard Tag Library).

first.jsp

```
<%@ page language = "java" contentType = "text/html" %>
<html>
  <head>
    <title> First JSP Page </title>
  </head>
  <body bgcolor = "green" >
    <H1> welcome to the Day </H1>
    <H1> <%= new Date().toString() %> </H1>
  </body>
</html>
```

Template Text

jsp elements

After processing JSP request, the template text and JSP elements are merged together, Sent to browser as response.

"JSP Processing"

- To process JSP pages, a web server needs a JSP engine called JSP Container.
- The JSP Container is responsible for handling JSP pages.
- The JSP container works with the webserver to provide runtime environment and the Services needed by JSP.

The JSP processing is

1. The browser Sends the HTTP requests to the web Server.
2. The web server recognizes the HTTP request is for a JSP page and forwards the request to jsp container. (By viewing .jsp file)
   Eg: "first.jsp"
3. On recieving the request, the jsp container Search & reads the desired page (jsp file).
   The jsp page is converted into corresponding Servlet.
   The jsp page is combination of jsp elements and template text.

The template text is Converted into Corrosponding println statement.

Eg: <html>
<body>
≡ ⟹ pw.println("<html>");
pw.println("<body>");

- Every JSP element is Converted into Corrosponding java code.

This phase is called "Translation phase"

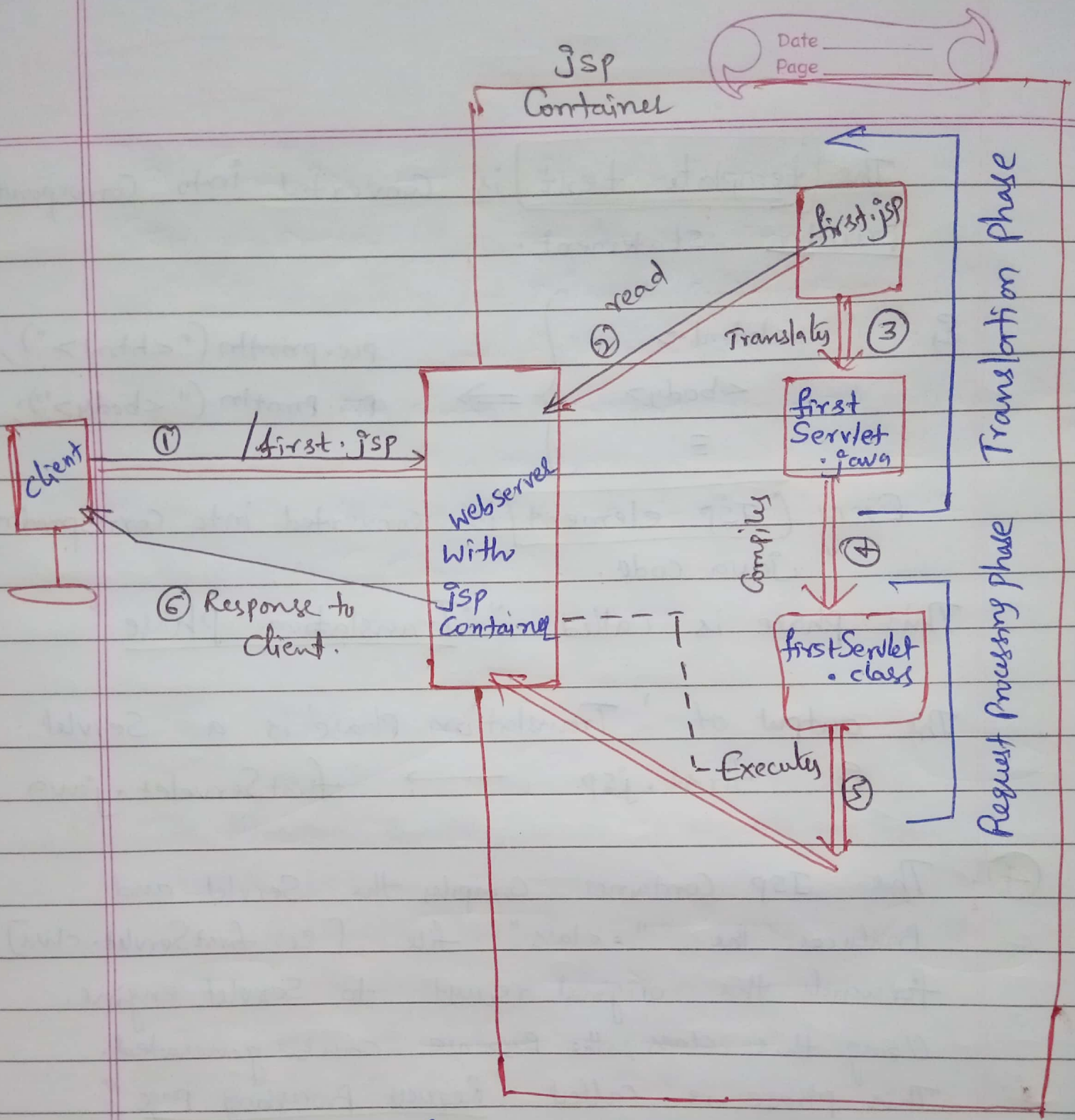
The output of 'Translation Phase' is a Servlet.
Eg: first.jsp ⟶ firstServelet.java.

④ The JSP Container Compiles the Servlet and Produces the ".class" file [Eg: firstServlet.class], forwards the original request to Servlet engine. Using this class, the Response can be generated. This phase is called "Request Procesing Page"

The Servlet Container loads the Servlet and executes it, produces the Output in HTML format

⑤ The Container give response to webserver.

⑥ The webserver forwards the HTTP response to browser.



Only for the first request, the response is slow. for every next request, if it is to the same jsp page, then the Container directly executes and gets response. Thus improves the performance.

JSP Application Design with MVC.

MVC stands for Model-View-Controller Architecture, it is a design pattern for developing web applications.

The basic idea in MVC model is to seperate design logic into three parts - Modelling, viewing and Controlling.

A server Application is classified into 3 parts.

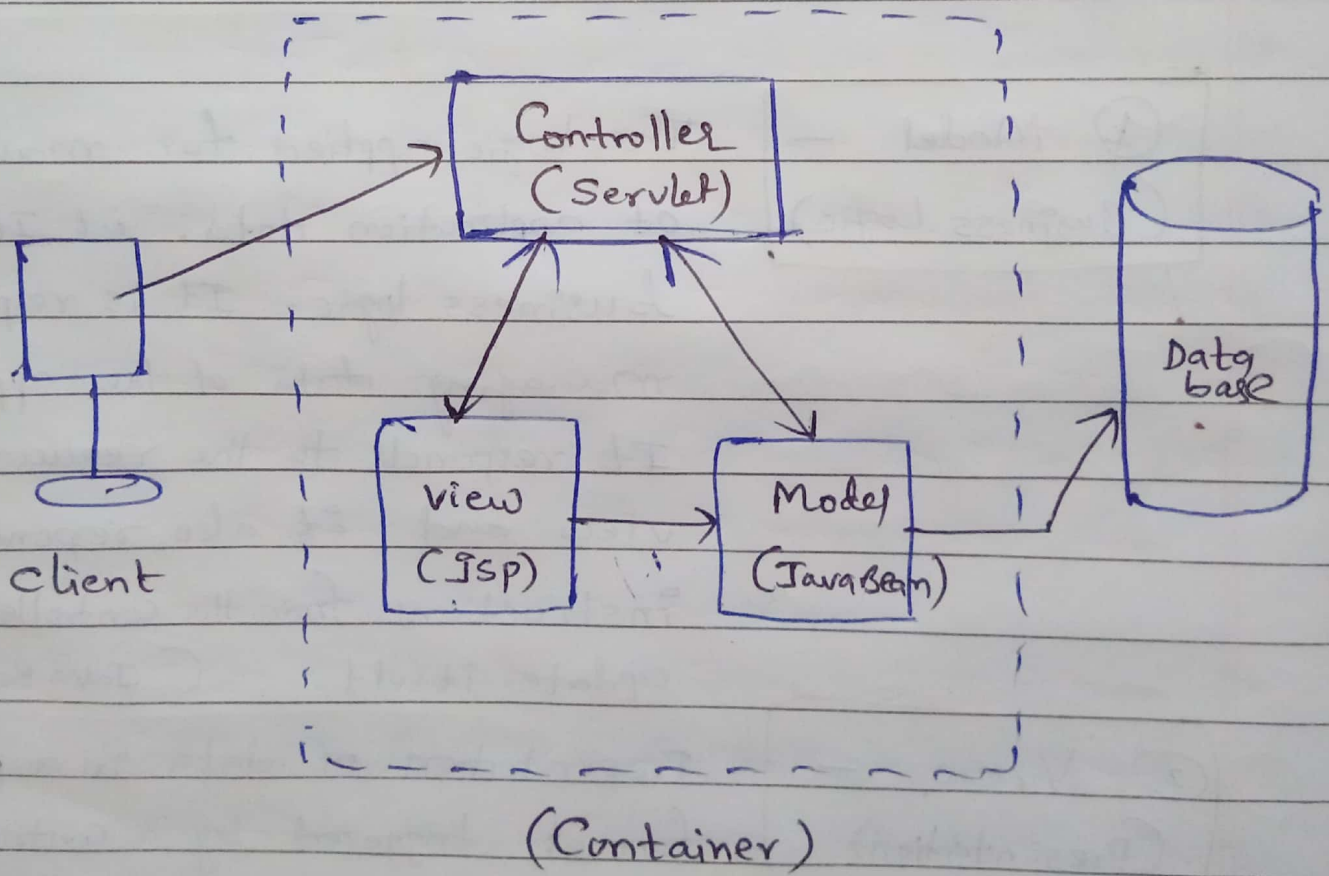
(1) Model - (Business Logic) — The logic applied for manipulation of application data. i.e It can have business logic. It is responsible for managing data of the application. It responds to the request from the view and it also responds to instructions from the controller to update itself. (Java Beans, EJB)

(2) View - (Presentation) logic — Presentation of data in a particular format, triggered by a controller's decision to present the data. It is the code written for look and feel of the webpage. (HTML, JSP, PHP). Eg: Background color, font size, label etc.

③. Controller — (Request Processing) logic

Responsible for responding to the user input and perform interactions on the data mode objects.

The Controller recieves the input, validates and then performs business operations that modifies state of data model. (Servlets).



The various Advantages of MVC Architecture

1. Faster Web Application Development
2. Ideal for developing large size web applications
3. MVC model returns the data without the need of formatting.
4. Allows to use web development tools like Macromedia flash or Dream Viewer.
5. The modifications never effect the entire model.
6. clear seperation between business, presentation & request processing logic.

JSP - Life cycle.

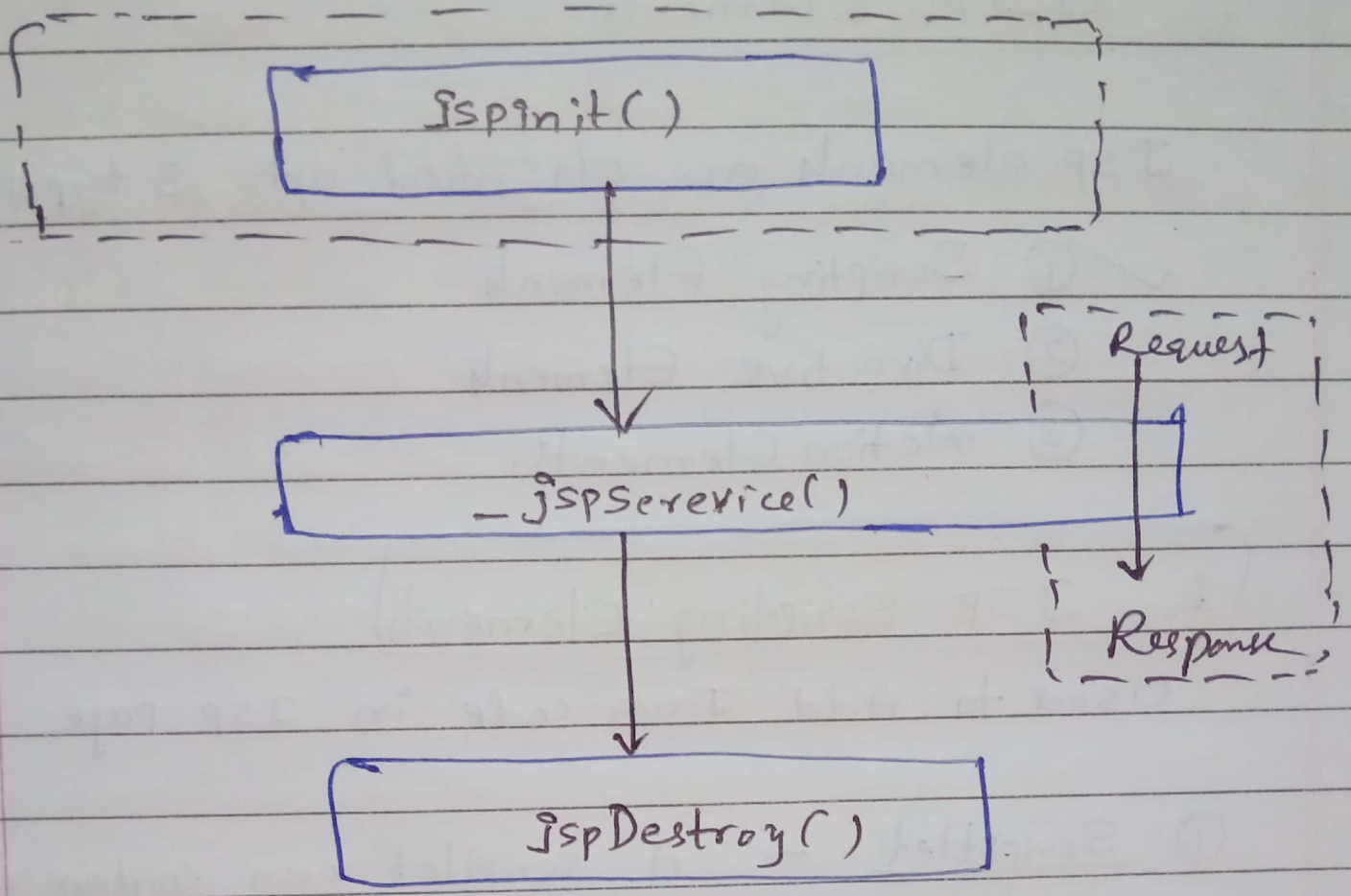
The life cycle of jsp is in 4 phases.

① Compilation – If the browser request a JSP page, first checks is it required to compile | is it already compiled. If Required
- Parsing JSP page
Translate JSP into Servlet
Compiling the Servlet.

② JSP Initialization – The container invokes the jspInit() before servicing any requests. We can initialize database connections, open files, create tables etc.

③ JSP Execution – The container invokes _jspService() method in the jsp. It takes HttpServletRequest & HttpServletResponse as parameters. It is responsible for generating the response.

④ JSP Cleanup – Before a JSP page is unloaded, this method is called to perform any cleanup operation such as closing files or releasing database connections.



JSP Elements

Jsp elements are classified into 3 types

1. Scripting Elements
2. Directive Elements
3. Action Elements.

1. JSP Scripting Elements

Used to Add Java code in JSP page

① Scriptlet — A scriptlet can contain any number of Java language statements, variables or method declarations or expressions.

The syntax of Scriptlet is

```
<% code fragment %>
```

A JSP Page can have any no. of Scriptlet tags.

```
<html>
<body>
Eg: <% out.println("Welcome to JSP"); %>
    <% out.println("Nice day"); %>
    <BR>
    <H1>
        <% out.println("Welcome to World"); %>
    </H1>
</body>
</html>
```

② JSP Declarations — Declares one or more variables or methods that we can use in Java code. The variables must be declared before using them. The Syntax of declaration is

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

It is called declaration tag, used to declare static members, instance variables or methods.

Eg.

```
<html>
  <body>
    <h3> Variables & Method Declaration </h3>
    <B>
    <%! String msg = "Hello world"; %>
    <%! int a=10; int b=20; double c=3.5; %>
    <%! public double mul(double a, double b)
        {
            return (a*b);
        } %>
    <% out.println("The msg is " +msg); %> <br>
    <% out.println("The result is " + (a+b+c)); %> <br>
    <% out.println("The a value is " +a); %> <br>
    <% out.println("The function call " +mul(5.5,3.5)); %>
  </body>
</html>
```

Eg:- <%! int i=20; %>
<%! int a,b,c; %>
<%! Box b1 = new Box(10,20,30); %>

③ JSP Expression — This element contains a scripting language expression that is evaluated, converted to a string, inserted where the expression appears in the jsp file.

The expression element contains any expression that is valid according to the java language specification.

NOTE: No semicolon to end an expression.

The Syntax of JSP expression is

<%= expression %>

Eg:- <!-- Expression element -->
<html>
<body>
<%= (200 % 8) %>
<%= (new java.util.Date()) %>
</body>
</html>

④ JSP Comments - Marks the text or statements that the jsp container should ignore.

The syntax is

<%-- Jsp Comments Here --%>

Eg: <%-- The method to find volume --%>

Note: — HTML Comments are also allowed.

<!-- The method to find volume -->

// Program to demonstrate factorial of a no.

```
<%@ page language = "java" contentType = "text/html" %>
<html>
  <body Style = " color : red " >
    <% out.println (" The factorial of 10 is "); %>
    <%! int fa = 1; %>
    <%
        for ( int i = 1 ; i <= 10 ; i++)
          {
            fa = fa * i;
          }
        out . println (" → " + fa);
    %>
  </body>
</html>
```

2. Action Elements.

→ JSP Action elements are useful to perform a specific task in jsp page.

→ The actions use constructs in xml syntax to control the behaviour of servlet container

→ Actions used to dynamically insert a file, reuse Javabean components, forward user to another page etc.

"Action elements are predefined functions".

The Syntax is

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

① <jsp : include>

- Used to include another jsp file or html file in to jsp page.
- Inserts a file at the time the jsp page is translated into a servlet, this action inserts the file at the time of page is requested.
- The Syntax is

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

clears buffer before loads.

Eg: <html>
<body>
<h1> Finding factorial of a number </h1>
<jsp:include page = "fact.jsp" flush = "true" />
</body>
</html>

" It provides better reusability of the code ".

② <jsp:param >
- It is used to pass parameters to another file.
- It can be used with either <jsp:include> or <jsp:forward>.
- The Syntax is

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

Eg: <jsp:param name = "n" value = "10" />
<jsp:param name = "city" value = "hyd" />

Eg: To find the reverse of a number.

param.jsp

<%@page language = "java" contentType = "text/html" %>
<html>
<body>
<jsp:include page = "reverse.jsp" >
<jsp:param name = "n1" value = "123" />
</body> </jsp:include>

reverse.jsp

```
<%@ page language="java" contentType="text/html" %>
<html>
  <body>
<% int n = Integer.parseInt(request.getParameter("n1"));
    int s=0, r;
    while (n>0)
     {
        r = n%10;
        s = s*10+r;
        n = n/10;
     }
    out.println("The reverse num is " +s);
%>
</body>
</html>
```

Execution : http://localhost:8080/myjsps/param.html.

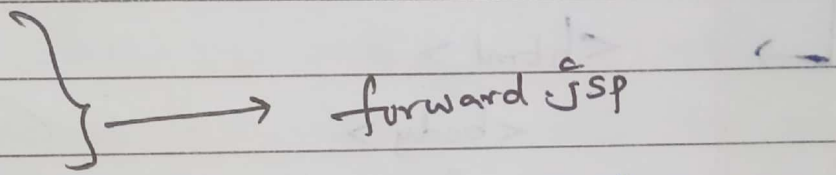
③. <jsp:forward>

- Used to forward the request to a new page.
- Terminates the action of the current page and forwards the request to another resource such as html page, jsp page or java servlet.
- The Syntax is

```
<jsp:forward page = "URL of source" />
```

Eg:

```
<html>
  <body>
    <jsp:forward page = "reverse.jsp" >
    <jsp:param name="n1" value = "123" />
    </jsp:forward >
  </body>
</html>
```



④. <jsp:plugin>

- Used to insert Java components in a jsp page.
- Used to include Bean or Applet into a jsp page.
- <param> element can also be used to send parameters to the applet or bean.
- The Syntax is

```
<jsp:plugin type = "bean|applet" code = "classfile"
  align = "left|middle|right" width = "pixels"
  height = "pixels" />
```

Eg:

If a specified plugin is not available, then to display alternate text or error message <jsp:fallback> is used.

The Syntax is

```
<jsp:fallback>
    // Error Msg
</jsp:fallback>
```

plugin.jsp

```
<html>
<body>
<jsp:plugin type="applet" code="MyApplet.class"
    width="300" height="200">
<jsp:fallback>
    Unable to load plugin
</jsp:fallback>
</jsp:plugin>
```

* Execute in internet explorer

"MyApplet.class"

```
import java.awt.*;
import java.applet.*;
public class MyApplet extends Applet
{
    public void paint (Graphics g)
    {
        setBackground (Color.blue);
        g.drawString ("It is a Applet", 10, 50);
        g.drawRect (50, 100, 800, 40);
    }
}
```

Date
Page

<jsp:useBean>

- It is used to find or instantiate a java bean
- It searches for an existing object utilizing the 'id' and 'scope' variables. If the object is not found, it tries to create the specified object.
- The syntax is

```
<jsp:useBean id="name" class="package.class" />
```

- Once a Bean class is loaded, "jsp:setProperty" and "jsp:getProperty" actions used to set or retrieve Bean properties.

<jsp:setProperty>

- Used to set the property of a bean.
- The Bean must have been previously defined before this action.
- The syntax is

```
<jsp:setProperty name="Bean name"
                 property="property name" value="anyval" />
```

<jsp:getProperty>

- Used to retrieve the value of a given bean property and Converts it to a string, finally insert it into output.
- The Syntax is

```
<jsp:getProperty name="Beanname" property="Prop name" />
```

Eg: ①. // demonstrates Student data.

StudData.java

```
package student;
public class StudData
{
    String name;
    public StudData()
    {
        name = "Vivek";
    }
    Public void setName(String n)
    {
        name = n;
    }
    public String getName()
    {
        return(name);
    }
}
```

To access in jsp — bean.jsp.

```
<html>
<body>
<jsp:useBean id = "s1" class = "student.StudData"/>
<%  out.println(s1.getName()); %>
<%  ob.setName("Ramesh"); %>
<%  out.println(s1.getName()); %>
<body>
</html>
```

② // Demonstrate Employee data.

Employee.java

```
package emp;
public class Employee
{  int salary;
   public Employee()
    {  salary = 25000;
    }
   public void setSalary(int x)
    {  salary = x;
    }
   public int getSalary()
    {  return(salary);
    }
}
```

To acces in jsp page

empbean.jsp

```
<html>
 <body>
  <jsp:useBean id="e1" class="emp.Employee" />
  <jsp:getProperty name="e1" property="salary" />
  <jsp:getProperty name="e1" property="salary"
                   value="45000" />
  <jsp:getProperty name="e1" property="salary" />
  </jsp:useBean>
 </body>
</html>
```

<jsp:text>

- It is used to write template text in jsp pages and documents.
- The body of the template can not contain other elements. (only text & expressions).
- The Syntax is

<jsp:text> Template text </jsp:text>

Implicit JSP objects

Date
Page

- During JSP translation phase, a web Container creates built-in objects, called implicit objects.
- These objects are Java objects the JSP container makes available to the developers in each page and the developer can call them directly without explicitly being declared.
- Implicit objects are also called predefined variables.

- The various implicit JSP objects are

1. request – / HttpServletRequest object associated with the request.
   / It provides methods for accessing the client request.
   Eg: / getContentLength()
   getServerName()
   getParameter()
   getParameterNames() etc.

```
<% String x = request.getParameter("user"); %>
```

2. response – / It is HttpServletResponse object associated with response to client.
   / It provides the methods for adding cookies, session or response, setting header etc.

/ addCookie(), addHeader(),
/ setContentType(), getContentType() etc
/ sendRedirect().

Eg:
```
<% out.println("welcome");
   response.setContentType("text/html");
   response.setContentType("text/html");
   response.sendRedirect("Welcome.html");
%>.
```

3. out – / This is the PrintWriter object used to send output to the client
/ It provides methods related to I/O.
/ clear(), newLine(), print(), println()
/ The output object is created by JspWriter.

```
<% out.println("Hello world");
```

4. page – / It is similar to "this" keyword in Java
/ It refers to the current JSP page.
/ It represents Object class.

⑤. exception – It is an object of Throwable.
– This object is for handling error pages and contain information about runtim error.

<%= exception %>

⑥ config – , It is an object of ServletConfig.
, It helps in passing the information to servlet or Jsp page during initialization
, It is used to maintain configuration information about particular servlet.
, getInitParameter()
getServletName() etc.

The init parameters are in <init-param> of "web.xml".

Eg:

web.xml

```
<web-app>
  <servlet>
    <servlet-name> MyDemo </servlet-name>
    <jsp-file> /welcome.jsp </jsp-file>
```

```
<init-param>
    <param-name> userName </param-name>
    <param-value> tiger </param-value>
</init-param>
</servlet>
<servlet-mapping>
    <servlet-name> MyDemo </servlet-name>
    <url-pattern> /demo </url-pattern>
</servlet-mapping>
</web-app>
```

⇓

In Jsp page it can be accessed as

- welcome.jsp"
- "welcome.jsp"

```
<html>
  <body>
    <% String pname = config.getInitParameter("UserNami");
       out.println(pname);
    %>
  </body>
</html>
```

(8). session – , It is an object of HttpSession.
, It is used to access the current client session.
, getId(), getCreationTime(), setAttribute() getAttribute() etc.

Eg:

user.html

```
<html>
  <body>
    <form name="f1" action="hello.jsp">
      <input type="text" name="un">
      <input type="submit" value="click">
    </form>
  </body>
</html>
```

hello.jsp

```
<html>
  <body>
  <% String name= request.getParameter("un");
     out.println("welcome to "+name);
     session.setAttribute("user", name);
  %>
  <a href="getsession.jsp"> Access </a>
  </body>
</html>
```

getSession.jsp

```
<html>
  <body>
    <% String name = (String) session.getAttribute("user");
       out.println("Session user is" + name);
    %>
  </body>
</html>
```

⑦. pageContext – It is an instance of PageContext class. Used to represent the entire JSP page.

- It is used to share data between various JSP pages based on its scope.
- The method to store information is

```
pageContext.setAttribute("name", "value", "scope");
```

- The Scope can be
  - pageContext.SESSION_SCOPE,
  - pageContext.SESSION-SCOPE, (for session)
  - pageContext.PAGE_SCOPE; (for page)
  - pageContext.APPLICATION_SCOPE (for website)
  - pageContext.REQUEST_SCOPE. (for request).

- To retrieve information

```
pageContext.getAttribute("name", "scope");
```

- To remove attribute from the page

```
pageContext.removeAttribute("name", "scope");
```

Eg:

user1.html

```
<html>
  <body>
    <form name = "f1" action = "first.jsp" >
      UserName : <input type="text" name = "un" >
                 <input type = "submit" value = "click" >
    </form>
  </body>
</html>
```

first.jsp

```
<html>
  <body>
  <% String name = Request.getParameter ("un");
     out.println ("welcome" + name);
  pageContext . setAttribute ("user", name, pageContext .
                              SESSION_SCOPE);
  %>
  <a href = "sec.jsp" > Goto </a>
</body > </html>
```

second.jsp

```
<html>
  <body>
    <% String name = (String) pageContext . getAttribute
                     ("user", pageContext . SESSION_SCOPE);
       out.println ("Hello" + name);
    %>
  </bdy>
</html>
```

9. application - It is an instance of ServletContext object.

- It is used to pass parameters to all JSP pages available in a web application (or) website.
- This object is created when jsp page is initialized and removed when calls JspDestroy() method.
- To set variable at application level,

```
application.setAttribute(String key, Object val);
```

- To get variable set at application level

```
application.getAttribute(String key);
```

"Used for application level parameters".

Eg: User name in all webpages,
no. of times a website visited. etc

Eg: // Count of no. of times visited website.

```
<html>
  <body>
  <%
Integer count = (Integer) application.getAttribute(
                    "hitCounter");
  if ( count == null || count == 0)
   {
     out.println("Welcome to site"); // first time
     count = 1;
   }
  else
    {
      out.println("Welcome Back again");
      count = count+1;
    }
application.setAttribute("hitCounter", count);
  %>
  <center>
      <B> <H3> No. of visit is : <%= count %>
      </H3> </B>
  </center>
 <body>
<html>
```

JSP
3. Directive Elements

Date
Page

- Directive elements are used for providing special instructions to JSP container for processing the page.
- Provides directions to container
- The Syntax of directive element is
  <%@ directive attribute = "value" %>
- The directives can have several attributes.
- The attributes are key-value pairs and seperated by commas.

The JSP directives are classified into 3 types.

1. page directive
2. include directive
3. taglib directive

1. page directive

- Used to provide instructions to the container.
- The instructions are limited to page only.
- The syntax is
  <%@ page attribute = "value" %>
- It defines page-dependant attributes, scripting language, error page, buffering requirements etc.

P.T.O.

The attributes of "page" directive are

① import – specifies a list of Packages or classes used in JSP page.

Eg: <%@ page import = "java.util.*" %>
<%@ page import = "java.util.Date" %>
<%@ page import = "java.io.*,
javax.servlet.*,
java.sql.* %>

② session – Specifies whether or not the JSP page participates in HTTP sessions. The possible values are true/false.
<%@ page session = "true/false" %>

③ errorPage – Defines the URL of another JSP that reports on Java unchecked runtime exceptions.
i.e. used to handle exceptions.
<%@ page errorPage = "errorhandler.jsp" %>

④ isErrorPage → It is used to check whether a particular JSP page will act as a exception handler or not. (true/false)
<%@ page isErrorPage = "true/false" %>

⑤ extends — Specifies the super class that the generated servlet must extend.
i-e. current page acquires properties of other class.

<%@ page extends = "className" %>

⑥ language — Defines the programing language used in the JSP page.

<%@ page language = "java" %>

⑦ buffer — It is used to specify the output buffer size. By default it is 8KB.

<%@ page buffer = "16KB" %>

⑧ contentType — Used to specify the type of response formulated by the JSP Container.

<%@ page contentType = "text/html" %>

↓

text/css
text/html
image/gif etc.

Eg. <%@ page import = "java.util.*" contentType = "text/css"
session = "true" errorPage = "handler-jsp" %>

2. include directive

- Includes a file during the translation in current page.
- It is similar to <jsp:include> action.

- include directive gets the content at the time of translation, where as <include> action gets at the time of processing request

<%@ include file = "url" %>

Ex: <%@ include file = "welcome.jsp" %>

3. taglib directive

- Declares a tag library, containing custom actions used in the page.
- JSP allows to define custom JSP tags, those are declared using taglib directive.
- The Syntax is

<%@ taglib uri = "taglibraryname"
prefix = "name" %>