Unit-7
Java Server Pages(JSP)

Servlet

- Servlets usually contain **HTML code embedded in Java** code.

- Servlet=Java+HTML(i.e. HTML within java)

- In addition, **servlets do not separate the presentation logic from the business logic in an application.**

Introduction to JSP

- It can be thought of as an **extension to servlet** because it provides more functionality than servlet such as expression language, jstl etc.

- **JSP** technology is used to create web application just like Servlet technology.
  (JSP=HTML+Java i.e.java within HTML)

- A **JSP page** is a text document that contains

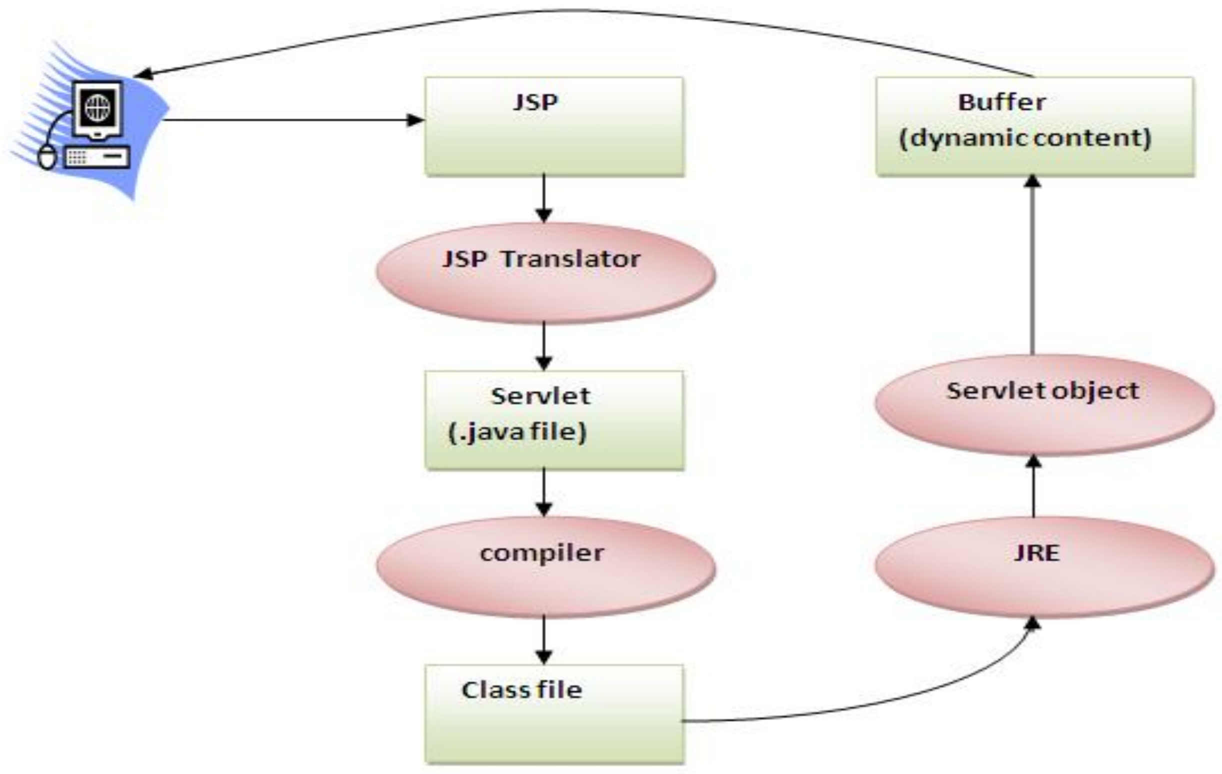
Why JSP is preffered over servlets?

- JSP provides an easier way to code dynamic web pages.
- JSP does not require additional files like, java class files, web.xml etc
- Any change in the JSP code is handled by Web Container(Application server like tomcat), and doesn't require re-compilation.
- JSP pages can be directly accessed, and web.xml mapping is not required like in servlets.

Advantage of JSP over Servlet

1) Extension to Servlet
2) Easy to maintain
3) Fast Development: No need to recompile and redeploy
4) Less code than Servlet

Processing of JSP

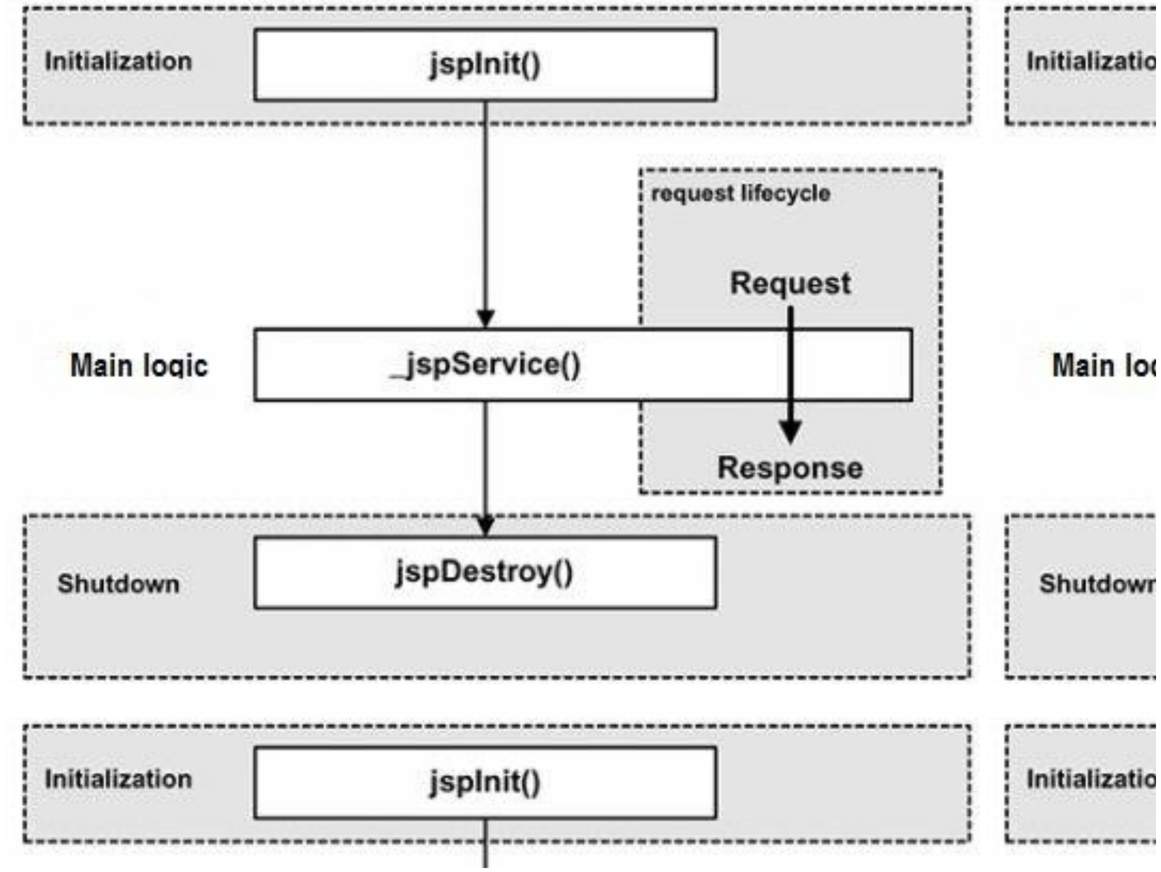


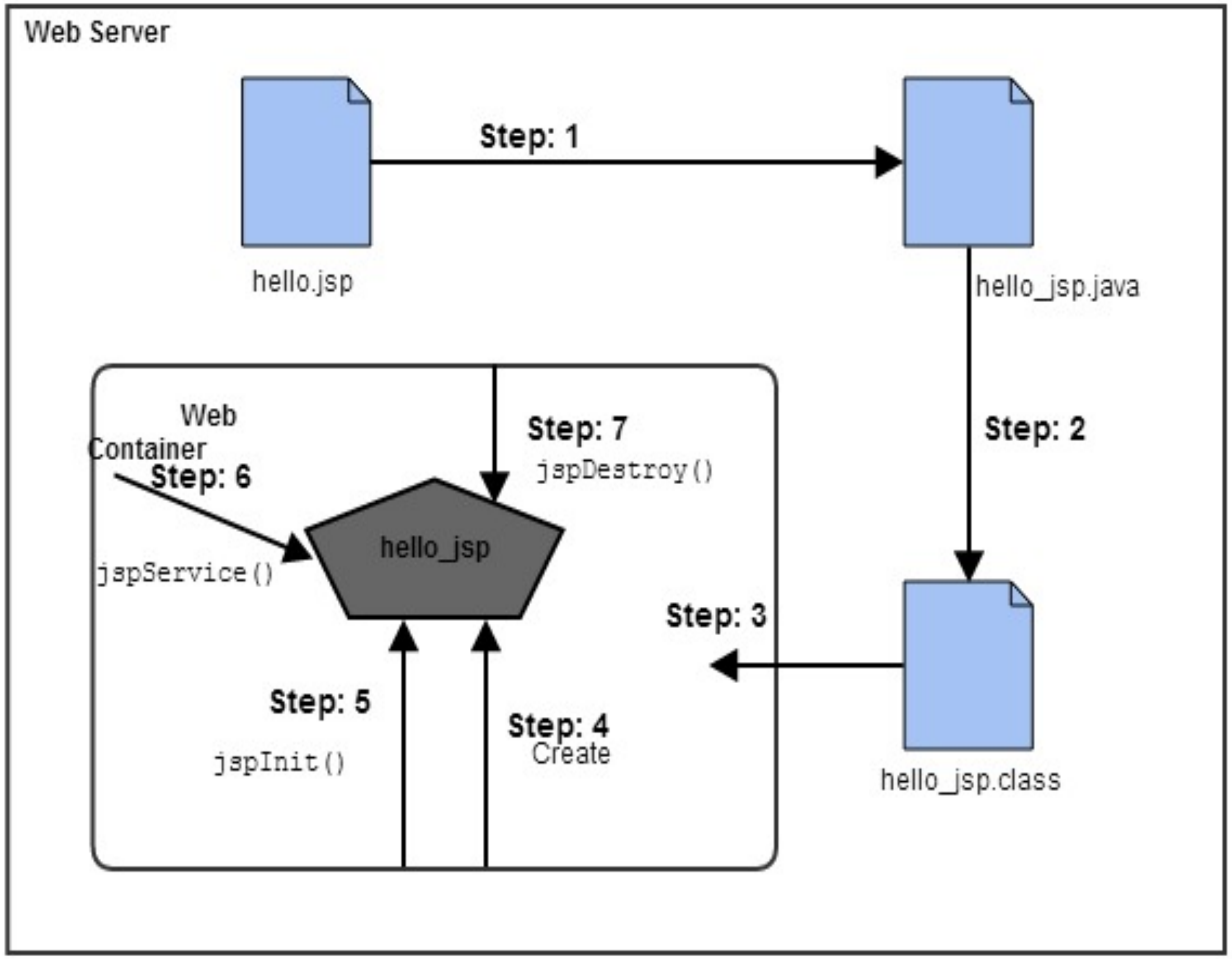
Life cycle of a JSP Page

- The JSP pages follows these phases:

1) **Translation** of JSP Page
2) **Compilation** of JSP Page
3) **Classloading** (class file is loaded by the classloader)
4) **Instantiation** (Object of the Generated Servlet is created).
5) **Initialization** (**jspInit()** method is invoked by the container).

Life Cycle of JSP




Web Server
hello.jsp
Step: 1
hello_jsp.java
Step: 2
hello_jsp.class
Step: 3
Web Container
Step: 6
jspService()
hello_jsp
Step: 7
jspDestroy()
Step: 5
jspInit()
Step: 4
Create

Step -1

- JSP is not processed as such, they first gets converted into Servelts and then the corresponding servlet gets processed by Server.

- whenever container receives request from client, it does translation only when servlet class is older than JSP page otherwsie it **skips this phase.**

Step-2:Then the container

- compiles the corresponding servlet program
- Loads the corresponding servlet class
- Instantiates the servlet class

- Calls the **jspInit() method** to initialize the servlet instance(Jsp container will do this job only when the instance of servlet file is not running or if it is older than the jsp file.)

public void jspInit()

Step:3

3) A new thread is then gets created, which invokes the**_jspService() method**, with a request (HttpServletRequest) and response (HttpServletRespnse) objects as parameters - shown below.

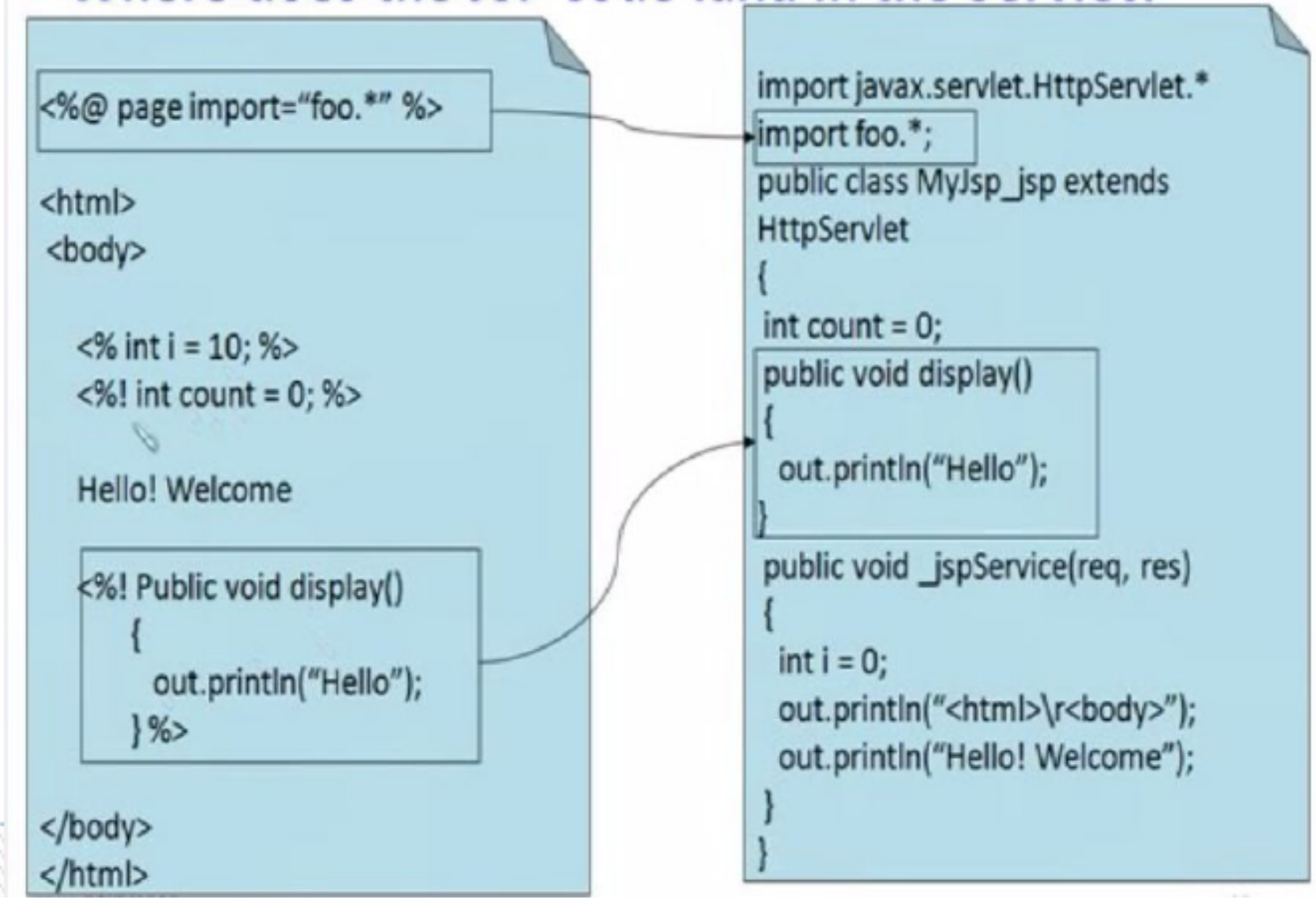
```
void _jspService( HttpServletRequest req,
HttpServletResponse res)
{
//code goes here
```

Step:4

4) Invokes the **jspDestroy() method** to destroy the instance of the servlet class. code will look like below –

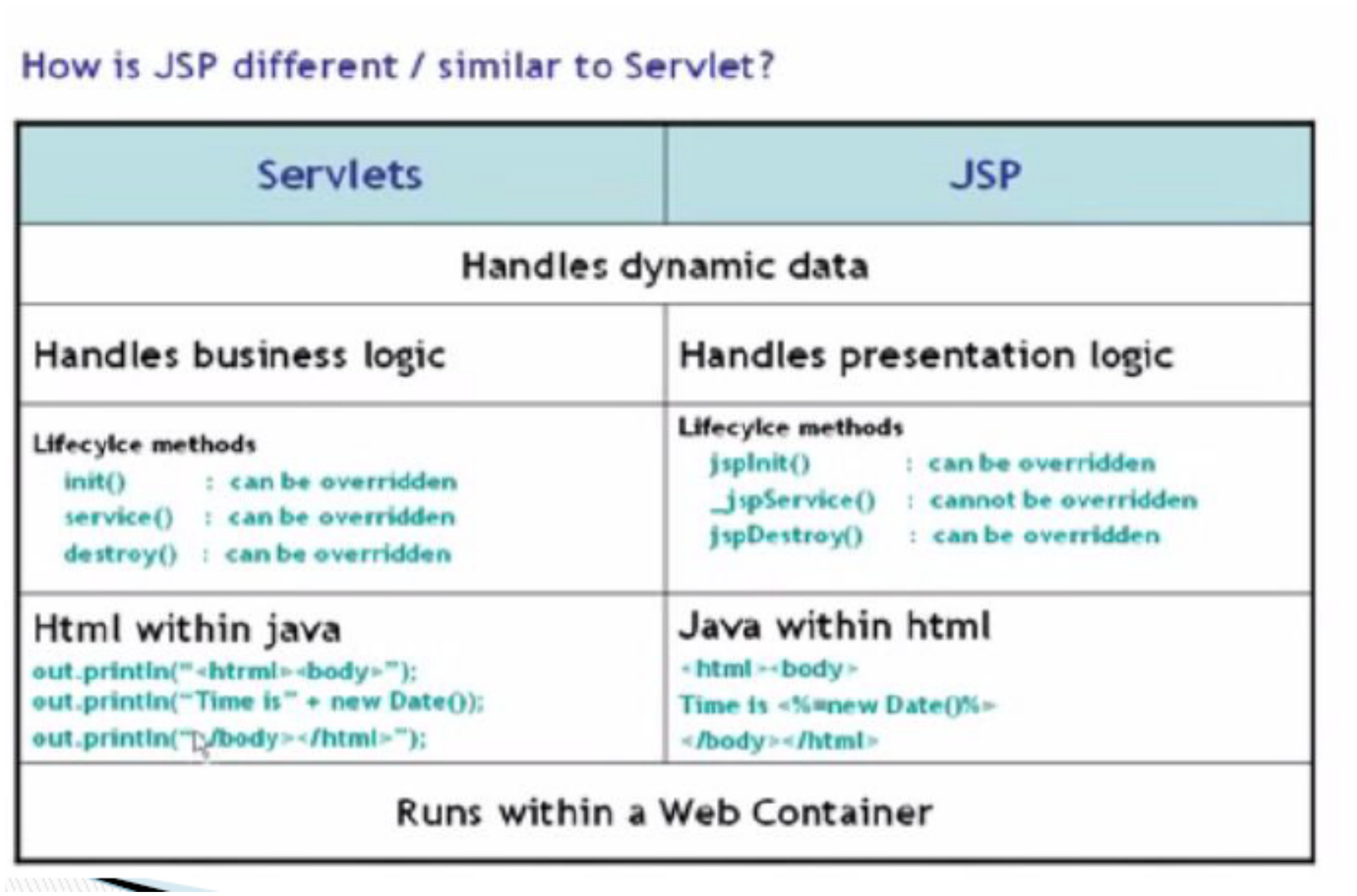
```
public void jspDestory()
 {
 //code to remove the instances of servlet
 class
 }[/code]
```

- **Where does the JSP code land in the Servlet?**



```
<%@ page import="foo.*" %>

<html>
 <body>

   <% int i = 10; %>
   <%! int count = 0; %>

   Hello! Welcome

   <%! Public void display()
       {
         out.println("Hello");
       } %>

</body>
</html>
```

```
import javax.servlet.HttpServlet.*
import foo.*;
public class MyJsp_jsp extends
HttpServlet
{
 int count = 0;
 public void display()
 {
   out.println("Hello");
 }
 public void _jspService(req, res)
 {
   int i = 0;
   out.println("<html>\r<body>");
   out.println("Hello! Welcome");
 }
}
```



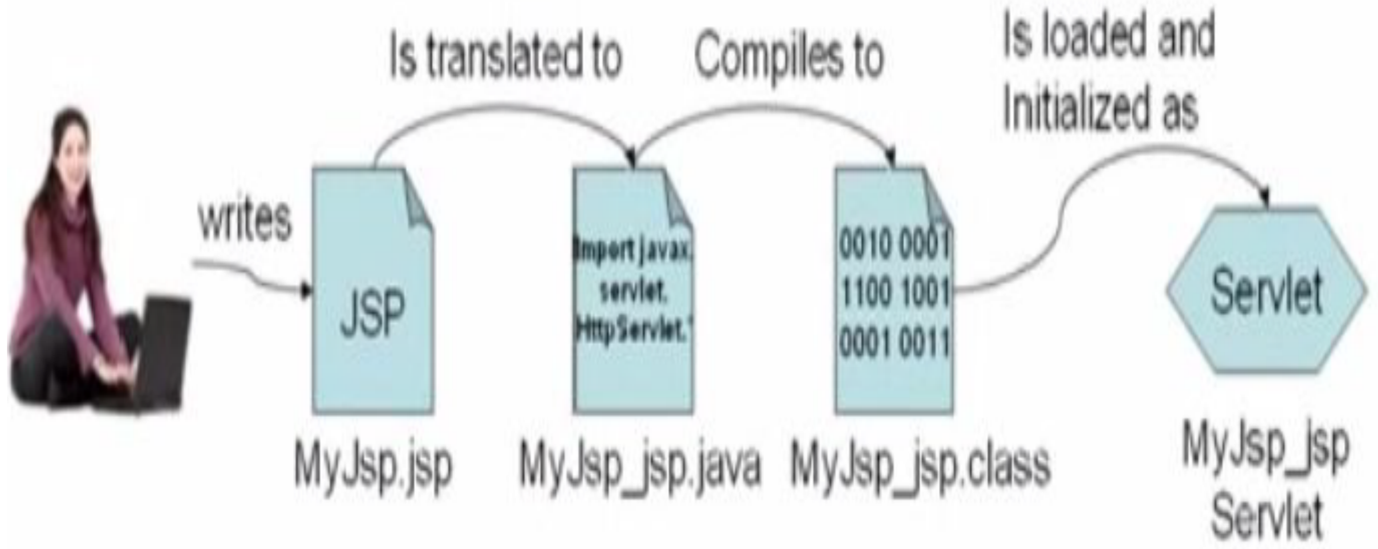
How is JSP different / similar to Servlet?

Servlets	JSP
Handles dynamic data	
Handles business logic	Handles presentation logic
Lifecylce methods init() : can be overridden service() : can be overridden destroy() : can be overridden	**Lifecylce methods** jspInit() : can be overridden _jspService() : cannot be overridden jspDestroy() : can be overridden

Html within java
`out.println("<html><body>");`
`out.println("Time is" + new Date());`
`out.println("</body></html>");`
	Java within html
	`<html><body>`
	`Time is <%=new Date()%>`
	`</body></html>`

Runs within a Web Container	

- In the end, a JSP is just a Servlet



JSP Scripting element

- JSP Scripting element are written inside <% %> tags. These code inside <% %> tags are processed by the JSP engine during translation of the JSP page. Any other text in the JSP page is considered as HTML code or plain text.

- **There are five different types of scripting elements.**

1. **Comment<%-- comment --%>**
2. **Directive<%@ directive %>**

Scriptlet Tag

<% java code %>

```
<html>
<head>
<title>My First JSP Page</title>
</head>
<body> Page Count is
<% uout.println(++cont); %>
</body> </html>
```

Example

- In this example, we will create a simple JSP page which retrieves the name of the user from the request parameter. The **index.html** page will get the username from the user.
- **index.html**

```
<form method="post" action="welcome.jsp">
 Name <input type="text" name="user" >
 <input type="submit" value="submit">
 </form>
```

- **welcome.jsp**

```
<html>
<title>Welcome Page</title>
<% String user = request.getParameter("user");
   %>
<body> Hello, <% out.println(user); %> </body>
   </html>
```

Declaration Tag

- We know that at the end a JSP page is translated into Servlet class. So when we declare a variable or method in JSP inside **Declaration Tag**, it means the declaration is made inside the Servlet class but outside the service(or any other) method. You can declare static member,instance variable and methods inside **Declaration Tag**.

- **Syntax of Declaration Tag**

  <%! declaration %>

Example of Declaration Tag

```
<html>
<head>
<title>My First JSP Page</title>
</head>
<%! int count = 0; %>
<body> Page Count is:
<% out.println(++count); %>
</body>
```

- **The above JSP page becomes this Servlet**

```
public class hello_jsp extends HttpServlet
{           int count=0;
  public void _jspService(HttpServletRequest
   request,HttpServletResponse response) throws
   IOException,ServletException
{
  PrintWriter out = response.getWriter();
   response.setContenType("text/html");
   out.write("<html><body>");
      out.write("Page count is:");
```

Expression Tag

- Expression Tag is used to print out java language expression that is put between the tags. An expression tag can hold any java language expression that can be used as an argument to the **out.print()** method.
- **Syntax of Expression Tag**

  <%= JavaExpression %>
- **When the Container sees this**

  <%= (2*5) %>
- **It turns it into this:**

Example of Expression Tag

```
<html>
<head>
<title>My First JSP Page</title>
</head> <% int count = 0; %>
<body> Page Count is <%= ++count %>
   </body>
</html>
```

Directive Tag

- **Directive Tag** gives special instruction to Web Container at the time of page translation. Directive tag are of three types: **page**,**include** and **taglib**.

1. **<%@ page ... %>**defines page dependent properties such as language, session, errorPage etc.
2. **<%@ include ... %>**defines file to be included.

Page directive

- You can place page directive anywhere in the JSP file, but it is good practice to make it as the first statement of the JSP page.

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

- The **Page directive** defines a number of page dependent properties which communicates with the Web Container at the time of translation.

1.Import **Attribute**

- The import attribute defines the set of classes and packages that must be inported in servlet class definition.
- For example
- <%@ page import="java.util.Date" %> or <%@ page import="java.util.Date,java.net.*" %>

2)contentType

```
<html>
<body>

<%@ page contentType=text/html%>
Today is: <%= new java.util.Date() %>

</body>
</html>
```

3)info

```
<html>
<body>


<%@ page info="Welcom
   e to PU" %>
Today is: <%= new java.ut
   il.Date() %>


</body>
</html>
```

```
public String getServletInf
   o() {
  return "Welcome to
   PU";
}
```

4)buffer

```
<html>
<body>

<%@ page buffer="16kb"
   %>
Today is: <%= new java.ut
   il.Date() %>

</body>
</html>
```

5)language

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

6)isThreadSafe

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

```
public class SimplePage_jsp extends HttpJspBase
   implements SingleThreadModel{
.......
}
```

7)errorPage

```
//index.jsp
<html>
<body>

<%@ page errorPage="my
    errorpage.jsp" %>

 <%= 100/0 %>

</body>
</html>
```

8)isErrorPage

```
//myerrorpage.jsp
<html>
<body>

<%@ page isErrorPage="t
    rue" %>

 Sorry an exception occure
    d!<br/>
The exception is: <%= exc
    eption %>
```

9)session

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

10)The isScriptingEnabled Attribute

- The **isScriptingEnabled** attribute determines if the scripting elements are allowed for use.
- <%@ page isScriptingEnabled = "false" %>

***11)autoFlush* attribute**

autoFlush attribute defines whether the buffered output is flushed automatically. The default value is "true".

12)**isELIgnored**

isELIgnored attribute gives you the ability to disable the evaluation of Expression Language (EL) expressions which has been introduced in JSP 2.0.

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

Implicit Objects in JSP

- JSP provide access to some implicit object which represent some commonly used objects for servlets that JSP page developers might need to use. For example you can retrieve HTML form parameter data by using **request** variable, which represent the **HttpServletRequest** object.

The "request" object is implicit here, associated with HttpServletRequest object

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

 Hello, <% out.println(user); %>
```

The "out" object is implicit in JSP, associated with the **JspWriter** object.

- **Implicit Object Description**

1. **request :**The **HttpServletRequest** object associated with the request.
2. **Response :**The **HttpServletRequest** object associated with the response that is sent back to the browser.
3. **Out :**The **JspWriter** object associated with the output stream of the response.
4. **Session :**The **HttpSession** object associated with the session for the given user of request.
5. **Application :**The **ServletContext** object for

Include Directive

- The *include* directive tells the Web Container to copy everything in the included file and paste it into current JSP file. Syntax of **include** directive.

- <%@ include file="filename.jsp" %>

Example of include directive

- **welcome.jsp**

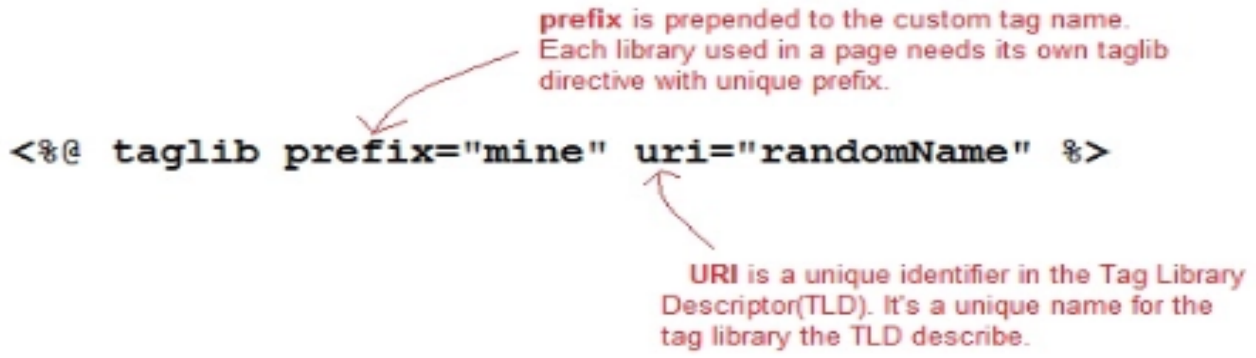
```
<html>
<head>
<title>Welcome Page</title>
</head>
<body> <%@ include file="header.jsp" %>
   Welcome, User </body>
</html>
```

header.jsp

```
<html> <body> <img src="header.jpg“
alt="This is Header image" / > </body> </html>
```

Taglib Directive

- The **taglib** directive is used to define tag library that the current JSP page uses. A JSP page might include several tag library. Syntax of taglib directive:



prefix is prepended to the custom tag name. Each library used in a page needs its own taglib directive with unique prefix.

```
<%@ taglib prefix="mine" uri="randomName" %>
```

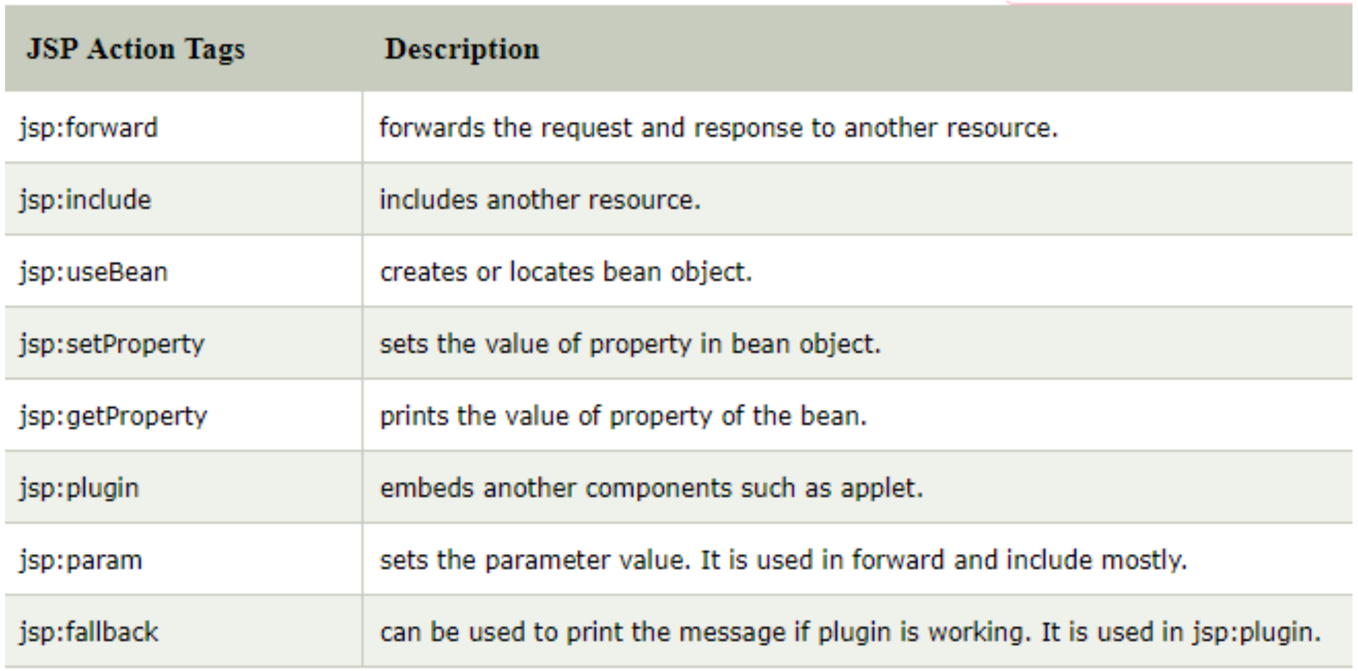
URI is a unique identifier in the Tag Library Descriptor(TLD). It's a unique name for the tag library the TLD describe.

Example of Taglib Directive

- In this example, we are using a tag **userName**. To use this tag we must specify some information to the Web Container using Taglib Directive.
- <html> <head> <title>Welcome Page</title> </head>
- **<%@ taglib prefix="mine" uri="myTags" %>** <body> Welcome, **<mine:userName / >** </body> </html>

JSP Action Tags

- used to control the flow between pages and to use Java Bean.



JSP Action Tags	Description
jsp:forward	forwards the request and response to another resource.
jsp:include	includes another resource.
jsp:useBean	creates or locates bean object.
jsp:setProperty	sets the value of property in bean object.
jsp:getProperty	prints the value of property of the bean.
jsp:plugin	embeds another components such as applet.
jsp:param	sets the parameter value. It is used in forward and include mostly.
jsp:fallback	can be used to print the message if plugin is working. It is used in jsp:plugin.

jsp:forward action tag

- **Syntax of jsp:forward action tag without parameter**

1.

```
<jsp:forward page="relativeURL | <%= expre
ssion %>" />
```

- **Syntax of jsp:forward action tag with parameter**

Example of jsp:forward action tag without parameter

```
<html>
<body>
<h2>this is index page</h2>

<jsp:forward page="printdate.jsp" />
</body>
</html>
```

```
<html>
<body>
<% out.print("Today is:"+java.util.Calendar.getI
   nstance().getTime()); %>
</body>
</html>
```

Example of jsp:forward action tag with parameter

```
<html>
<body>
<h2>this is index page</h2>

<jsp:forward page="printdate.jsp" >
<jsp:param name="name" value="PU" />
</jsp:forward>

</body>
</html>
```

```
<html>
<body>

<% out.print("Today is:"+java.util.Calendar.getI
   nstance().getTime()); %>
<%= request.getParameter("name") %>

</body>
</html>
```

jsp:include action tag

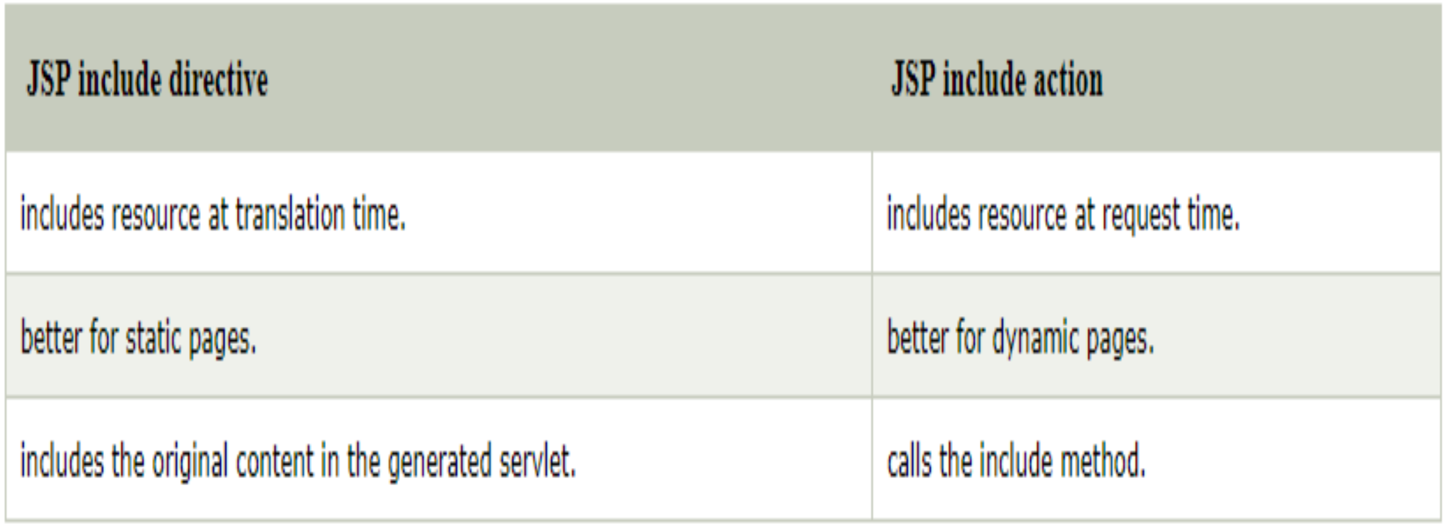
- **Syntax of jsp:include action tag without parameter**

1.
```
<jsp:include page="relativeURL | <%= expre
ssion %>" />
```

- **Syntax of jsp:include action tag with parameter**

2.
```
<jsp:include page="relativeURL | <%= expre
```

Difference between jsp include directive and include action



JSP include directive	JSP include action
includes resource at translation time.	includes resource at request time.
better for static pages.	better for dynamic pages.
includes the original content in the generated servlet.	calls the include method.

Example of jsp:include action tag without parameter

```
<h2>this is index page</h2>

<jsp:include page="printdate.jsp" />

<h2>end section of index page</h2>
```

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

Java Bean

- A Java Bean is a java class that should follow following conventions:

1. It should have a no-arg constructor.
2. It should be Serializable.
3. It should provide methods to set and get the values of the properties, known as getter and setter methods.

Why use Java Bean?

- it is a reusable software component.
- A bean encapsulates many objects into one object, so we can access this object from multiple places.
- Moreover, it provides the easy maintenance.

```
package mypack;
public class Employee implements java.io.Serial
   izable{
private int id;
private String name;

public Employee(){}

public void setId(int id){this.id=id;}

public int getId(){return id;}
```

to access the java bean

```
package mypack;
public class Test{
public static void main(String args[]){

Employee e=new Employee();//object is created


e.setName("Arjun");//setting value to the object


System.out.println(e.getName());
```

jsp:useBean action tag

```
<jsp:useBean id= "instanceName"
scope= "page | request | session | application"
class= "packageName.className"
type= "packageName.className"
beanName="packageName.className |
<%= expression >" >
</jsp:useBean>
```

```java
package com.pu;
public class Calculator{

public int cube(int n)
    {return n*n*n;}

}
```

```jsp
<jsp:useBean id="obj" cla
    ss="com.pu.Calculator"/
    >

<%
int m=obj.cube(5);
out.print("cube of 5 is "+m
    );
%>
```

Syntax of jsp:setProperty action tag

1.
```
<jsp:setProperty name="instanceOfBean" pro
perty= "*"   |
property="propertyName" param="parameterNa
me"  |
property="propertyName" value="
{ string | <%= expression %>}"
/>
```
2.
```
<jsp:setProperty name="bean" property="*" /
```

jsp:getProperty action tag

- jsp:getProperty name="instanceOfBean" prope
  rty="propertyName" />

```
<form action="process.jsp" method="post">
Name:<input type="text" name="name"><br>
Password:
    <input type="password" name="password">
    <br>
Email:<input type="text" name="email"><br>
<input type="submit" value="register">
</form>
```

```
<jsp:useBean id="u" class="org.sssit.User">
   </jsp:useBean>
<jsp:setProperty property="*" name="u"/>

Record:<br>
<jsp:getProperty property="name" name="u"/>
   <br>
<jsp:getProperty property="password" name="u"
   /><br>
<jsp:getProperty property="email" name="u" />
```

```
package org.sssit;

public class User {
private String name,password,email;
//setters and getters
}
```

Displaying applet in JSP (jsp:plugin action tag)

```
<jsp:plugin type= "applet | bean" code= "nameOf
   ClassFile"
codebase= "directoryNameOfClassFile"
</jsp:plugin>
```

```
<html>
    <head>
        <meta http-equiv="Content-
    Type" content="text/html; charset=UTF-8">
        <title>Mouse Drag</title>
    </head>
    <body bgcolor="khaki">
<h1>Mouse Drag Example</h1>

 <jsp:plugin align="middle" height="500" width
```