**Interfaces in javax.servlet package**

**There are many interfaces in javax.servlet package. They are as follows:**

**Servlet**

**ServletRequest**

**ServletResponse**

**RequestDispatcher**

**ServletConfig**

**ServletContext**

**Methods of Servlet interface**

**There are 5 methods in Servlet interface. The init, service and destroy are the life cycle methods of servlet. These are invoked by the web container.**

**public void init(ServletConfig config) initializes the servlet. It is the life cycle method of servlet and invoked by the web container only once.**

**public void service(ServletRequest request,ServletResponse response)**

**provides response for the incoming request. It is invoked at each request by the web container.**

**public void destroy()**

**is invoked only once and indicates that servlet is being destroyed.**

**public ServletConfig getServletConfig()**

**returns the object of ServletConfig.**

**public String getServletInfo() returns information about servlet such as writer, copyright, version etc.**

**Life Cycle of a Servlet (Servlet Life Cycle):-**

**The web container maintains the life cycle of a servlet instance. Let's see the life cycle of the servlet:**

**Servlet class is loaded.**

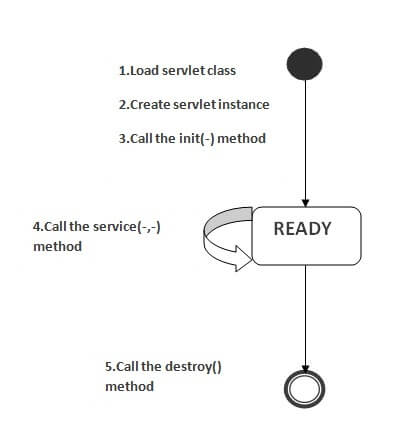
**Servlet instance is created.**

**init method is invoked.**

**service method is invoked.**

**destroy method is invoked.**

**Life cycle of a servlet**



**index.html**

**<form action="welcome" method="get">**

**Enter your name<input type="text" name="name"><br>**

**<input type="submit" value="login">**

**</form>**

**DemoServ.java**

**import javax.servlet.http.\*;**

**import javax.servlet.\*;**

**import java.io.\*;**

**public class DemoServ extends HttpServlet{**

**public void doGet(HttpServletRequest req,HttpServletResponse res)**

**throws ServletException,IOException**

**{**

**res.setContentType("text/html");**

**PrintWriter pw=res.getWriter();**

**String name=req.getParameter("name");//will return value**

**pw.println("Welcome "+name);**

**pw.close();**

**}}**

**<web-app>**

**<servlet>**

**<servlet-name>sonoojaiswal</servlet-name>**

**<servlet-class>DemoServlet</servlet-class>**

**</servlet>**

**<servlet-mapping>**

**<servlet-name>Demoserv</servlet-name>**

**<url-pattern>/welcome</url-pattern>**

**</servlet-mapping>**

**</web-app>**

**RequestDispatcher in Servlet**

**RequestDispatcher Interface**

**Methods of RequestDispatcher interface**

**forward method**

**include method**

**The RequestDispatcher interface provides the facility of dispatching the request to another resource it may be html, servlet or jsp. This interface can also be used to include the content of another resource also. It is one of the way of servlet collaboration.**

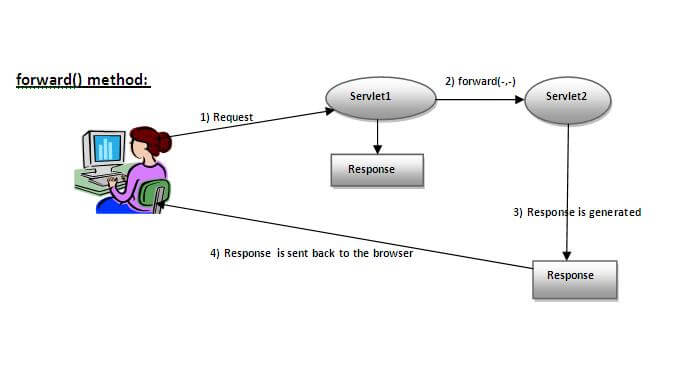
**There are two methods defined in the RequestDispatcher interface.**

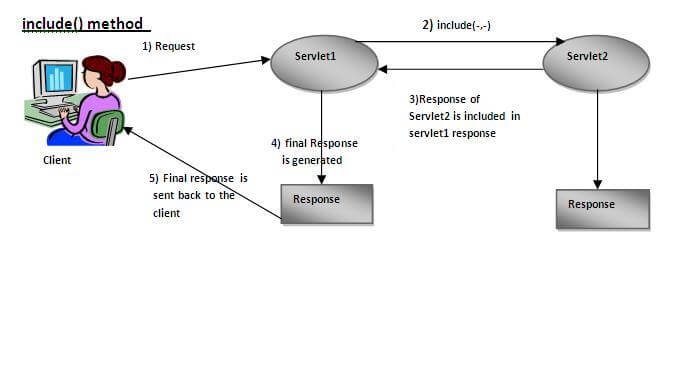
**Methods of RequestDispatcher interface**

**The RequestDispatcher interface provides two methods. They are:**

**public void forward(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException:Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.**

**public void include(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException:Includes the content of a resource (servlet, JSP page, or HTML file) in the response.**





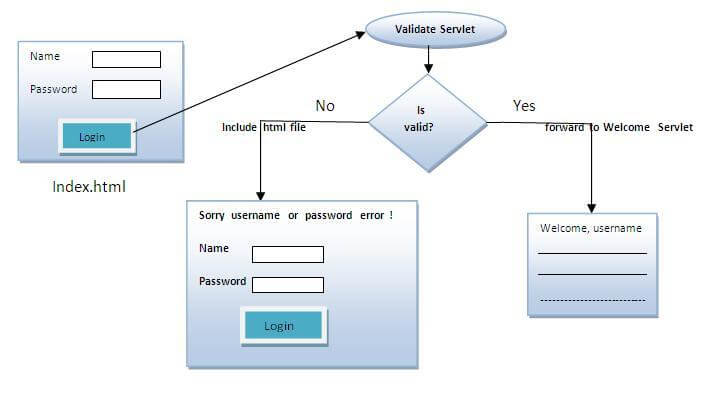
**Example of RequestDispatcher interface**

**index.html file: for getting input from the user.**

**Login.java file: a servlet class for processing the response. If password is servet, it will forward the request to the welcome servlet.**

**WelcomeServlet.java file: a servlet class for displaying the welcome message.**

**web.xml file: a deployment descriptor file that contains the information about the servlet.**



**index.html**

**<form action="servlet1" method="post">**

**Name:<input type="text" name="userName"/><br/>**

**Password:<input type="password" name="userPass"/><br/>**

**<input type="submit" value="login"/>**

**</form>**

**Login.java**

**import java.io.\*;**

**import javax.servlet.\*;**

**import javax.servlet.http.\*;**

**public class Login extends HttpServlet {**

**public void doPost(HttpServletRequest request, HttpServletResponse response)**

**throws ServletException, IOException {**

**response.setContentType("text/html");**

**PrintWriter out = response.getWriter();**

**String n=request.getParameter("userName");**

**String p=request.getParameter("userPass");**

**if(p.equals("servlet"){**

**RequestDispatcher rd=request.getRequestDispatcher("servlet2");**

**rd.forward(request, response);**

**}**

**else{**

**out.print("Sorry UserName or Password Error!");**

**RequestDispatcher rd=request.getRequestDispatcher("/index.html");**

**rd.include(request, response);**

**}**

**}**

**}**

**WelcomeServlet.java**

**import java.io.\*;**

**import javax.servlet.\*;**

**import javax.servlet.http.\*;**

**public class WelcomeServlet extends HttpServlet {**

**public void doPost(HttpServletRequest request, HttpServletResponse response)**

**throws ServletException, IOException {**

**response.setContentType("text/html");**

**PrintWriter out = response.getWriter();**

**String n=request.getParameter("userName");**

**out.print("Welcome "+n);**

**}**

**}**

**web.xml**

**<web-app>**

**<servlet>**

**<servlet-name>Login</servlet-name>**

**<servlet-class>Login</servlet-class>**

**</servlet>**

**<servlet>**

**<servlet-name>WelcomeServlet</servlet-name>**

**<servlet-class>WelcomeServlet</servlet-class>**

**</servlet>**

**<servlet-mapping>**

**<servlet-name>Login</servlet-name>**

**<url-pattern>/servlet1</url-pattern>**

**</servlet-mapping>**

**<servlet-mapping>**

**<servlet-name>WelcomeServlet</servlet-name>**

**<url-pattern>/servlet2</url-pattern>**

**</servlet-mapping>**

**<welcome-file-list>**

**<welcome-file>index.html</welcome-file>**

**</welcome-file-list>**

**</web-app>**

**SendRedirect in servlet**

**sendRedirect method**

**The sendRedirect() method of HttpServletResponse interface can be used to redirect response to another resource, it may be servlet, jsp or html file.**

**It accepts relative as well as absolute URL.**

**It works at client side because it uses the url bar of the browser to make another request. So, it can work inside and outside the server.**

**example of sendRedirect method in servlet**

**import java.io.\*;**

**import javax.servlet.\*;**

**import javax.servlet.http.\*;**

**public class DemoServlet extends HttpServlet{**

**public void doGet(HttpServletRequest req,HttpServletResponse res)**

**throws ServletException,IOException**

**{**

**res.setContentType("text/html");**

**PrintWriter pw=res.getWriter();**

**response.sendRedirect("http://www.google.com");**

**pw.close();**

**}}**

**Servlet Config Interface:-**

**An object of ServletConfig is created by the web container for each servlet. This object can be used to get configuration information from web.xml file.If the configuration information is modified from the web.xml file, we don't need to change the servlet. So it is easier to manage the web application if any specific content is modified from time to time.**

**Advantage of ServletConfig**

**The core advantage of ServletConfig is that you don't need to edit the servlet file if information is modified from the web.xml file.**

**Methods of ServletConfig interface**

**public String getInitParameter(String name):Returns the parameter value for the specified parameter name.**

**public Enumeration getInitParameterNames():Returns an enumeration of all the initialization parameter names.**

**public String getServletName():Returns the name of the servlet.**

**public ServletContext getServletContext():Returns an object of ServletContext.**

**Example of Servlet Config:-**

**DemoServlet.java**

**import java.io.\*;**

**import javax.servlet.\*;**

**import javax.servlet.http.\*;**

**public class DemoServlet extends HttpServlet {**

**public void doGet(HttpServletRequest request, HttpServletResponse response)**

**throws ServletException, IOException {**

**response.setContentType("text/html");**

**PrintWriter out = response.getWriter();**

**ServletConfig config=getServletConfig();**

**String driver=config.getInitParameter("driver");**

**out.print("Driver is: "+driver);**

**out.close();**

**}**

**}**

**web.xml**

**<web-app>**

**<servlet>**

**<servlet-name>DemoServlet</servlet-name>**

**<servlet-class>DemoServlet</servlet-class>**

**<init-param>**

**<param-name>driver</param-name>**

**<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>**

**</init-param>**

**</servlet>**

**<servlet-mapping>**

**<servlet-name>DemoServlet</servlet-name>**

**<url-pattern>/servlet1</url-pattern>**

**</servlet-mapping>**

**</web-app>**

**ServletContext Interface**

**An object of ServletContext is created by the web container at time of deploying the project. This object can be used to get configuration information from web.xml file. There is only one ServletContext object per web application.If any information is shared to many servlet, it is better to provide it from the web.xml file using the <context-param> element.**

**Advantage of ServletContext**

**Easy to maintain if any information is shared to all the servlet, it is better to make it available for all the servlet. We provide this information from the web.xml file, so if the information is changed, we don't need to modify the servlet. Thus it removes maintenance problem.**

**Usage of ServletContext Interface**

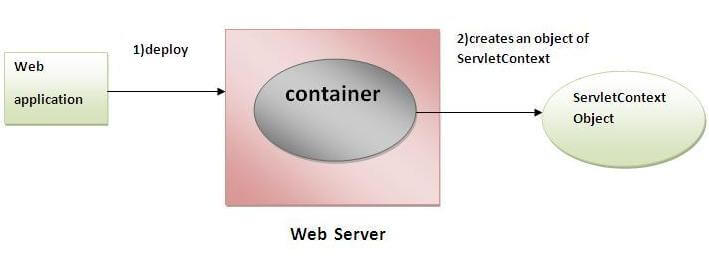
**There can be a lot of usage of ServletContext object. Some of them are as follows:**

**The object of ServletContext provides an interface between the container and servlet.**

**The ServletContext object can be used to get configuration information from the web.xml file.**

**The ServletContext object can be used to set, get or remove attribute from the web.xml file.**

**The ServletContext object can be used to provide inter-application communication.**



**Servlet Context Example:-**

**DemoServlet.java**

**import java.io.\*;**

**import javax.servlet.\*;**

**import javax.servlet.http.\*;**

**public class DemoServlet extends HttpServlet{**

**public void doGet(HttpServletRequest req,HttpServletResponse res)**

**throws ServletException,IOException**

**{**

**res.setContentType("text/html");**

**PrintWriter pw=res.getWriter();**

**//creating ServletContext object**

**ServletContext context=getServletContext();**

**//Getting the value of the initialization parameter and printing it**

**String driverName=context.getInitParameter("dname");**

**pw.println("driver name is="+driverName);**

**pw.close();**

**}**

**}**

**web.xml**

**<web-app>**

**<servlet>**

**<servlet-name>sonoojaiswal</servlet-name>**

**<servlet-class>DemoServlet</servlet-class>**

**</servlet>**

**<context-param>**

**<param-name>dname</param-name>**

**<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>**

**</context-param>**

**<servlet-mapping>**

**<servlet-name>DemoServlet</servlet-name>**

**<url-pattern>/context</url-pattern>**

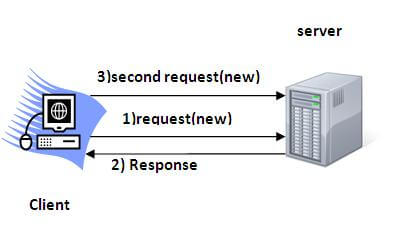
**</servlet-mapping>**

**</web-app>**

**Session Tracking in Servlets**

**Session Tracking is a way to maintain state (data) of an user. It is also known as session management in servlet. Http protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.**

**HTTP is stateless that means each request is considered as the new request. It is shown in the figure given below:**



**Why use Session Tracking?**

**To recognize the user It is used to recognize the particular user.**

**Session Tracking Techniques**

**There are four techniques used in Session tracking:**

**Cookies**

**Hidden Form Field**

**URL Rewriting**

**HttpSession**

**Cookies:-**

**Cookies in Servlet**

**A cookie is a small piece of information that is persisted between the multiple client requests.**

**A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.**

**How Cookie works**

**By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet. So cookie is stored in the cache of the browser. After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user.**

**Advantage of Cookies**

**Simplest technique of maintaining the state.**

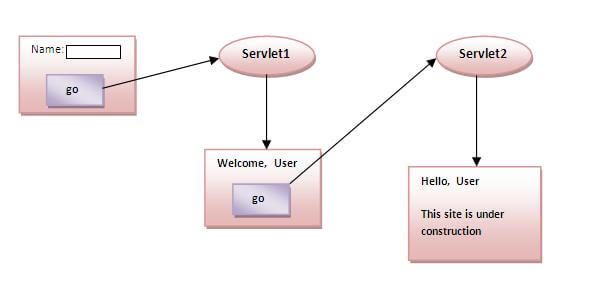
**Cookies are maintained at client side.**

**Disadvantage of Cookies**

**It will not work if cookie is disabled from the browser.**

**Only textual information can be set in Cookie object.**

**Simple example of Servlet Cookies:-**



**index.html**

**<form action="servlet1" method="post">**

**Name:<input type="text" name="userName"/><br/>**

**<input type="submit" value="go"/>**

**</form>**

**FirstServlet.java**

**import java.io.\*;**

**import javax.servlet.\*;**

**import javax.servlet.http.\*;**

**public class FirstServlet extends HttpServlet {**

**public void doPost(HttpServletRequest request, HttpServletResponse response){**

**try{**

**response.setContentType("text/html");**

**PrintWriter out = response.getWriter();**

**String n=request.getParameter("userName");**

**out.print("Welcome "+n);**

**Cookie ck=new Cookie("uname",n);//creating cookie object**

**response.addCookie(ck);//adding cookie in the response**

**//creating submit button**

**out.print("<form action='servlet2'>");**

**out.print("<input type='submit' value='go'>");**

**out.print("</form>");**

**out.close();**

**}catch(Exception e){System.out.println(e);}**

**}**

**}**

**SecondServlet.java**

**import java.io.\*;**

**import javax.servlet.\*;**

**import javax.servlet.http.\*;**

**public class SecondServlet extends HttpServlet {**

**public void doPost(HttpServletRequest request, HttpServletResponse response){**

**try{**

**response.setContentType("text/html");**

**PrintWriter out = response.getWriter();**

**Cookie ck[]=request.getCookies();**

**out.print("Hello "+ck[0].getValue());**

**out.close();**

**}catch(Exception e){System.out.println(e);}**

**}**

**}**

**web.xml**

**<web-app>**

**<servlet>**

**<servlet-name>s1</servlet-name>**

**<servlet-class>FirstServlet</servlet-class>**

**</servlet>**

**<servlet-mapping>**

**<servlet-name>s1</servlet-name>**

**<url-pattern>/servlet1</url-pattern>**

**</servlet-mapping>**

**<servlet>**

**<servlet-name>s2</servlet-name>**

**<servlet-class>SecondServlet</servlet-class>**

**</servlet>**

**<servlet-mapping>**

**<servlet-name>s2</servlet-name>**

**<url-pattern>/servlet2</url-pattern>**

**</servlet-mapping>**

**</web-app>**

**CREATE TABLE "REGISTERUSER"**

**( "NAME" VARCHAR2(4000),**

**"PASS" VARCHAR2(4000),**

**"EMAIL" VARCHAR2(4000),**

**"COUNTRY" VARCHAR2(4000)**

**)**

**/**

**To create the registration page in servlet, we can separate the database logic from the servlet. But here, we are mixing the database logic in the servlet only for simplicity of the program. We will develop this page in JSP following DAO, DTO and Singleton design pattern later.**

**Example of Registration form in servlet**

**In this example, we have created the three pages.**

**register.html**

**Register.java**

**web.xml**

**register.html**

**<html>**

**<body>**

**<form action="servlet/Register" method="post">**

**Name:<input type="text" name="userName"/><br/><br/>**

**Password:<input type="password" name="userPass"/><br/><br/>**

**Email Id:<input type="text" name="userEmail"/><br/><br/>**

**Country:**

**<select name="userCountry">**

**<option>India</option>**

**<option>Pakistan</option>**

**<option>other</option>**

**</select>**

**<br/><br/>**

**<input type="submit" value="register"/>**

**</form>**

**</body>**

**</html>**

**import java.io.\*;**

**import java.sql.\*;**

**import javax.servlet.ServletException;**

**import javax.servlet.http.\*;**

**public class Register extends HttpServlet {**

**public void doPost(HttpServletRequest request, HttpServletResponse response)**

**throws ServletException, IOException {**

**response.setContentType("text/html");**

**PrintWriter out = response.getWriter();**

**String n=request.getParameter("userName");**

**String p=request.getParameter("userPass");**

**String e=request.getParameter("userEmail");**

**String c=request.getParameter("userCountry");**

**try{**

**Class.forName("oracle.jdbc.driver.OracleDriver");**

**Connection con=DriverManager.getConnection(**

**"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");**

**PreparedStatement ps=con.prepareStatement(**

**"insert into registeruser values(?,?,?,?)");**

**ps.setString(1,n);**

**ps.setString(2,p);**

**ps.setString(3,e);**

**ps.setString(4,c);**

**int i=ps.executeUpdate();**

**if(i>0)**

**out.print("You are successfully registered...");**

**}catch (Exception e2) {System.out.println(e2);}**

**out.close();**

**}**

**}**

**web-app>**

**<servlet>**

**<servlet-name>Register</servlet-name>**

**<servlet-class>Register</servlet-class>**

**</servlet>**

**<servlet-mapping>**

**<servlet-name>Register</servlet-name>**

**<url-pattern>/servlet/Register</url-pattern>**

**</servlet-mapping>**

**<welcome-file-list>**

**<welcome-file>register.html</welcome-file>**

**</welcome-file-list>**

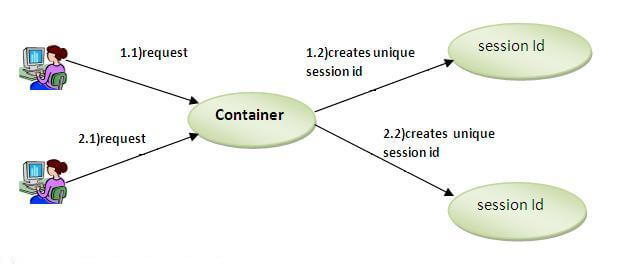
**</web-app>**

**HttpSession interface:-**

**In such case, container creates a session id for each user.The container uses this id to identify the particular user.An object of HttpSession can be used to perform two tasks:**

**bind objects**

**view and manipulate information about a session, such as the session identifier, creation time, and last accessed time.**



**How to get the HttpSession object ?**

**The HttpServletRequest interface provides two methods to get the object of HttpSession:**

**public HttpSession getSession():Returns the current session associated with this request, or if the request does not have a session, creates one.**

**public HttpSession getSession(boolean create):Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session.**

**Commonly used methods of HttpSession interface**

**public String getId():Returns a string containing the unique identifier value.**

**public long getCreationTime():Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.**

**public long getLastAccessedTime():Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT.**

**public void invalidate():Invalidates this session then unbinds any objects bound to it.**

**Example of using HttpSession:-**

**index.html**

**<form action="servlet1">**

**Name:<input type="text" name="userName"/><br/>**

**<input type="submit" value="go"/>**

**</form>**

**FirstServlet.java**

**import java.io.\*;**

**import javax.servlet.\*;**

**import javax.servlet.http.\*;**

**public class FirstServlet extends HttpServlet {**

**public void doGet(HttpServletRequest request, HttpServletResponse response){**

**try{**

**response.setContentType("text/html");**

**PrintWriter out = response.getWriter();**

**String n=request.getParameter("userName");**

**out.print("Welcome "+n);**

**HttpSession session=request.getSession();**

**session.setAttribute("uname",n);**

**out.print("<a href='servlet2'>visit</a>");**

**out.close();**

**}catch(Exception e){System.out.println(e);}**

**}**

**}**

**SecondServlet.java**

**import java.io.\*;**

**import javax.servlet.\*;**

**import javax.servlet.http.\*;**

**public class SecondServlet extends HttpServlet {**

**public void doGet(HttpServletRequest request, HttpServletResponse response)**

**try{**

**response.setContentType("text/html");**

**PrintWriter out = response.getWriter();**

**HttpSession session=request.getSession(false);**

**String n=(String)session.getAttribute("uname");**

**out.print("Hello "+n);**

**out.close();**

**}catch(Exception e){System.out.println(e);}**

**}**

**}**

**JSP Java Server page:-**

**JSP** technology is used to create web application just like Servlet technology. It can be thought of as an extension to Servlet because it provides more functionality than servlet such as expression language, JSTL, etc.

A JSP page consists of HTML tags and JSP tags. The JSP pages are easier to maintain than Servlet because we can separate designing and development. It provides some additional features such as Expression Language, Custom Tags, etc.

Advantages of JSP over Servlet:-

1.extension to servlet.

2.Easy to matain.

3.fast development.

4.Less code than servlet.

The Lifecycle of a JSP Page:-

Translation of JSP Page

Compilation of JSP Page

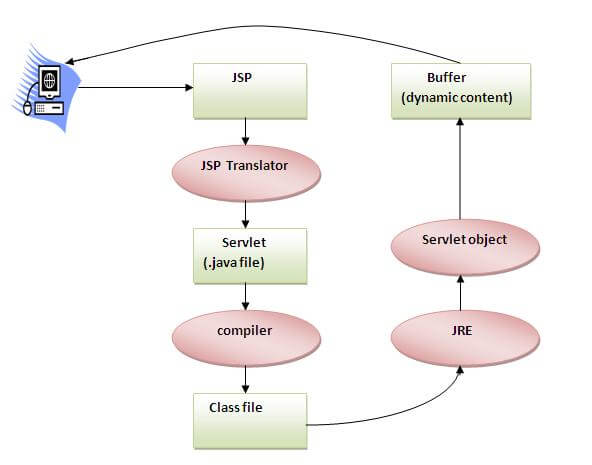
Classloading (the classloader loads class file)

Instantiation (Object of the Generated Servlet is created).

Initialization ( the container invokes jspInit() method).

Request processing ( the container invokes \_jspService() method).

Destroy ( the container invokes jspDestroy() method)



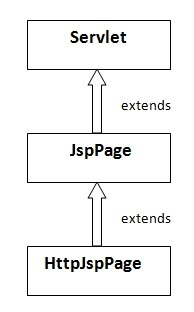
The JSP API consists of two packages:-

javax.servlet.jsp

javax.servlet.jsp.tagext

The JspPage interface

According to the JSP specification, all the generated servlet classes must implement the JspPage interface. It extends the Servlet interface.



JSP Scripting elements:-

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

scriptlet tag

expression tag

declaration tag

JSP scriptlet tag

A scriptlet tag is used to execute java source code in JSP. Syntax is as follows:

<% java source code %>

Example of JSP scriptlet tag that prints the user name:-

File: index.html

<html>

<body>

<form action="welcome.jsp">

<input type="text" name="uname">

<input type="submit" value="go"><br/>

</form>

</body>

</html>

File: welcome.jsp

<html>

<body>

<%

String name=request.getParameter("uname");

out.print("welcome "+name);

%>

</form>

</body>

</html>

JSP 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.

Syntax of JSP expression tag

<%= statement %>

Example:- use expression tag in previous Example.

File: welcome.jsp

<html>

<body>

<%= "Welcome "+request.getParameter("uname") %>

</body>

</html>

JSP 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. So it doesn't get memory at each request.

Syntax of JSP declaration tag

The syntax of the declaration tag is as follows:

<%! field or method declaration %>

Example:-

for Fields:-

index.jsp

<html>

<body>

<%! int data=50; %>

<%= "Value of the variable is:"+data %>

</body>

</html>

for Method:-

index.jsp

<html>

<body>

<%!

int cube(int n){

return n\*n\*n\*;

}

%>

<%= "Cube of 3 is:"+cube(3) %>

</body>

</html>

JSP Implicit Objects:-

There are 9 jsp implicit objects. These objects are created by the web container that are available to all the jsp pages.The available implicit objects are out, request, config, session, application etc. A list of the 9 implicit objects is given below:

Object Type

out JspWriter

request HttpServletRequest

response HttpServletResponse

config ServletConfig

application ServletContext

session HttpSession

pageContext PageContext

page Object

exception Throwable

JSP directives:-

There are three types of directives:

page directive

include directive

taglib directive

Syntax of JSP Directive

<%@ directive attribute="value" %>

JSP page directive

The page directive defines attributes that apply to an entire JSP page.

Syntax of JSP page directive

<%@ page attribute="value" %>

Attributes of JSP page directive

import

contentType

extends

info

buffer

language

isELIgnored

isThreadSafe

autoFlush

session

Example of import attribute

<html>

<body>

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

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

</body>

</html>

Example of contentType attribute

<html>

<body>

<%@ page contentType=application/msword %>

Today is: <%= new java.util.Date() %>

</body>

</html>

Jsp Include Directive:-

The include directive is used to include the contents of any resource it may be jsp file, html file or text file. The include directive includes the original content of the included resource at page translation time (the jsp page is translated only once so it will be better to include static resource).

Advantage of Include directive

Code Reusability

Syntax of include directive

<%@ include file="resourceName" %>

Example of include directive:-

In this example, we are including the content of the header.html file. To run this example you must create an header.html file.

<html>

<body>

<%@ include file="header.html" %>

Today is: <%= java.util.Calendar.getInstance().getTime() %>

</body>

</html>

MVC in JSP

Example of following MVC in JSP

MVC stands for Model View and Controller. It is a design pattern that separates the business logic, presentation logic and data.

Controller acts as an interface between View and Model. Controller intercepts all the incoming requests.

Model represents the state of the application i.e. data. It can also have business logic.

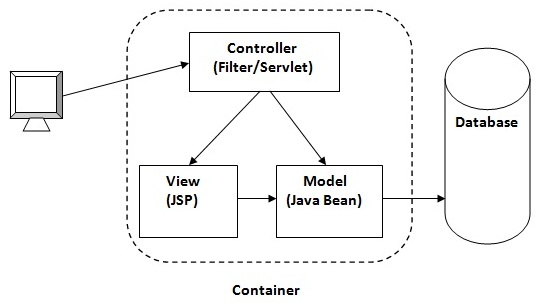
View represents the presentaion i.e. UI(User Interface)

Advantage of MVC (Model 2) Architecture

Navigation Control is centralized

Easy to maintain the large application

mvc architecture



MVC Example in JSP

In this example, we are using servlet as a controller, jsp as a view component, Java Bean class as a model.In this example, we have created 5 pages:

index.jsp a page that gets input from the user.

ControllerServlet.java a servlet that acts as a controller.

login-success.jsp and login-error.jsp files acts as view components.

web.xml file for mapping the servlet.

File: index.jsp

<form action="ControllerServlet" method="post">

Name:<input type="text" name="name"><br>

Password:<input type="password" name="password"><br>

<input type="submit" value="login">

</form>

File: ControllerServlet

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.RequestDispatcher;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class ControllerServlet extends HttpServlet {

protected void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

response.setContentType("text/html");

PrintWriter out=response.getWriter();

String name=request.getParameter("name");

String password=request.getParameter("password");

LoginBean bean=new LoginBean();

bean.setName(name);

bean.setPassword(password);

request.setAttribute("bean",bean);

boolean status=bean.validate();

if(status){

RequestDispatcher rd=request.getRequestDispatcher("login-success.jsp");

rd.forward(request, response);

}

else{

RequestDispatcher rd=request.getRequestDispatcher("login-error.jsp");

rd.forward(request, response);

}

}

@Override

protected void doGet(HttpServletRequest req, HttpServletResponse resp)

throws ServletException, IOException {

doPost(req, resp);

}

}

File: LoginBean.java

package com;

public class LoginBean {

private String name,password;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public boolean validate(){

if(password.equals("admin")){

return true;

}

else{

return false;

}

}

}

File: login-success.jsp

<%@page import="com.javatpoint.LoginBean"%>

<p>You are successfully logged in!</p>

<%

LoginBean bean=(LoginBean)request.getAttribute("bean");

out.print("Welcome, "+bean.getName());

%>

File: login-error.jsp

<p>Sorry! username or password error</p>

<%@ include file="index.jsp" %>

JSP Action Tags

jsp:forward action tag

Example of jsp:forward action tag without parameter

Example of jsp:forward action tag with parameter

There are many JSP action tags or elements. Each JSP action tag is used to perform some specific tasks.

The action tags are used to control the flow between pages and to use Java Bean. The Jsp action tags are given below.

JSTL (JSP Standard Tag Library)

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

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.

JSTL Tags

There JSTL mainly provides five types of tags:

Tag Name Description

Core tags The JSTL core tag provide variable support, URL management, flow control, etc. The URL for the core tag is <http://java.sun.com/jsp/jstl/core>. The prefix of core tag is c.

Function tags The functions tags provide support for string manipulation and string length. The URL for the functions tags is <http://java.sun.com/jsp/jstl/functions> and prefix is fn.

Formatting tags The Formatting tags provide support for message formatting, number and date formatting, etc. The URL for the Formatting tags is <http://java.sun.com/jsp/jstl/fmt> and prefix is fmt.

XML tags 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.

SQL tags The JSTL SQL tags provide SQL support. The URL for the SQL tags is <http://java.sun.com/jsp/jstl/sql> and prefix is sql.

**JSTL Core Tags**

**The JSTL core tag provides variable support, URL management, flow control etc. The syntax used for including JSTL core library in your JSP is:**

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

JSTL Core Tags List

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