Module 2

Java EE Web Applications

Content to be discussed

- Introduction to Eclipse & Apache Tomcat Server
- Servlet API Fundamentals
- ServletContext
- Session
- Cookies
- Request Redirection Techniques

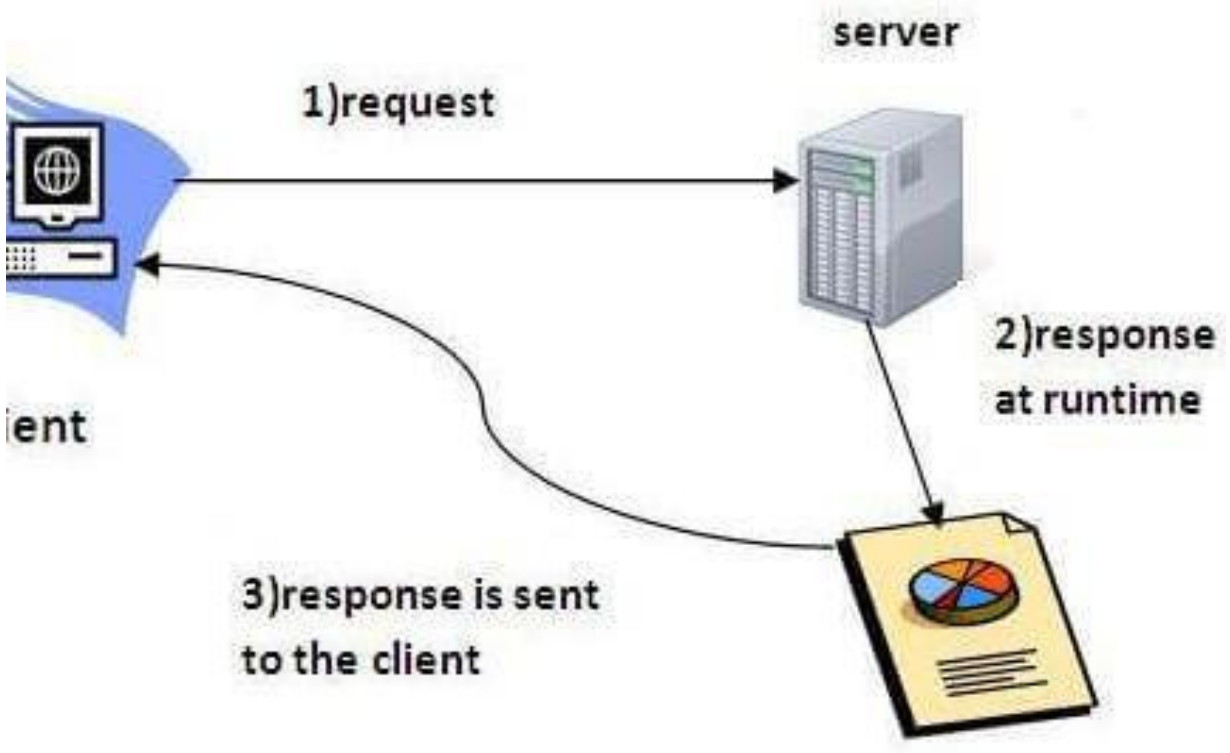
Eclipse & Apache Tomcat Server

- For Web & EE app development the requirement is “Eclipse IDE for Enterprise Java and Web Developers”
- **Apache Tomcat is a web server and servlet container that's used to deploy and serve Java web applications.**
- **Install Tomcat on Windows - https://tomcat.apache.org/download-90.cgi**
- **Install**

Servlet

- **Servlet** technology is used to create a web application (resides at server side and generates a dynamic web page).
- Servlet is an API that provides many interfaces and classes including documentation.
- Servlet is an interface that must be implemented for creating any Servlet.
- Servlet is a class that extends the capabilities of the servers and responds to the incoming requests. It can respond to any requests.
- Servlet is a web component that is deployed on the server to create a dynamic web page.
- **Servlet** technology is robust and scalable because of java language.
- Before Servlet, CGI (Common Gateway Interface) scripting language was common as a server-side programming language.
- There are many interfaces and classes in the Servlet API such as Servlet, GenericServlet, HttpServlet, ServletRequest, ServletResponse, etc.

How servlet operates?



Life cycle methods of a Servlet

Called only once

service() handles multiple client requests and send sends

Called only once

What is a web application?

- A web application is an application accessible from the web.
- A web application is composed of web components like Servlet, JSP, Filter, etc. and other elements such as HTML, CSS, and JavaScript.
- The web components typically execute in Web Server and respond to the HTTP request.

Advantages of Servlet

- There are many advantages of Servlet over CGI. The web container creates threads for handling the multiple requests to the Servlet. Threads have many benefits over the Processes such as they share a common memory area, lightweight, cost of communication between the threads are low. The advantages of Servlet are as follows:

1. **Better performance:** because it creates a thread for each request, not process.
2. **Portability:** because it uses Java language.
3. **Robust:** JVM manages Servlets, so we don't need to worry about the memory leak, garbage collection, etc.
4. **Secure:** because it uses java language.

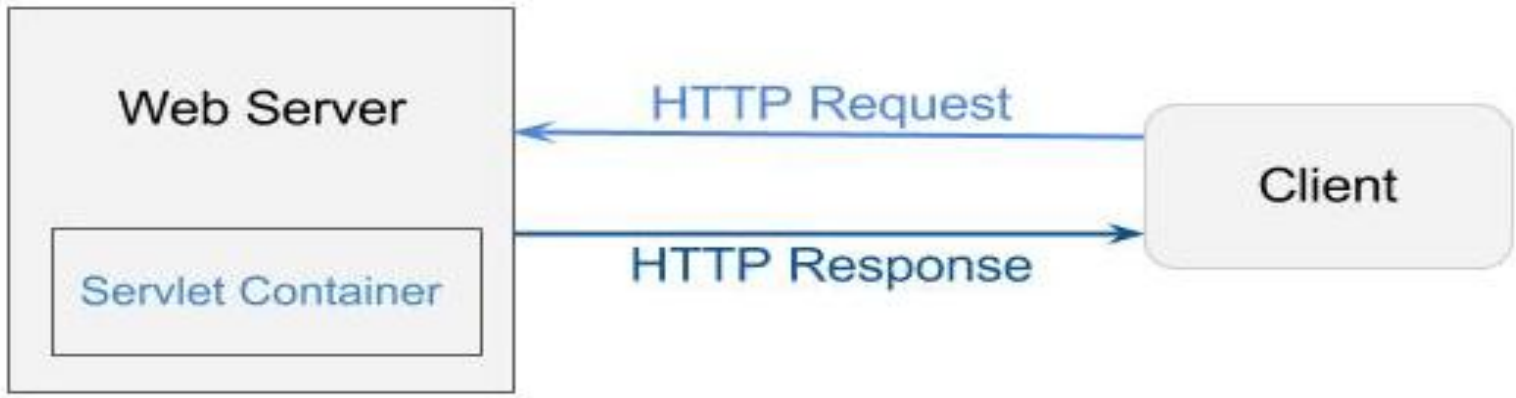
HTTP request methods

- The HTTP request method indicates the method to be performed on the resource identified by the **Requested URI (Uniform Resource Identifier)**. This method is case-sensitive and should be used in uppercase.

HTTP Request	Description
GET	Asks to get the resource at the requested URL.
POST	Asks the server to accept the body info attached. It is like GET request with extra info sent with the request.
HEAD	Asks for only the header part of whatever a GET would return. Just like GET but with no body.
TRACE	Asks for the loopback of the request message, for testing or troubleshooting.
PUT	Says to put the enclosed info (the body) at the requested URL.
DELETE	Says to delete the resource at the requested URL.
OPTIONS	Asks for a list of the HTTP methods to which the thing at the request URL can respond

Servlet Container

- It provides the runtime environment for JavaEE (j2ee) applications. The client/user can request only a static WebPages from the server. If the user wants to read the web pages as per input then the servlet container is used in java.
- The servlet container is the part of web server which can be run in a separate process. We can classify the servlet container states in three types:



Servlet Container States

- The servlet container is the part of web server which can be run in a separate process. We can classify the servlet container states in three types:
- **Standalone:** It is typical Java-based servers in which the servlet container and the web servers are the integral part of a single program. For example:- Tomcat running by itself
- **In-process:** It is separated from the web server, because a different program runs within the address space of the main server as a plug-in. For example:- Tomcat running inside the JBoss.
- **Out-of-process:** The web server and servlet container are different programs which are run in a different process. For performing the communications between them, web server uses the plug-in provided by the servlet container.

The Servlet Container performs many operation

- Life Cycle Management
- Multithreaded support
- Object Pooling
- Security etc.

Servlet API

- The javax.servlet and javax.servlet.http packages represent interfaces and classes for servlet api.
- The **javax.servlet** package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol.
- The **javax.servlet.http** package contains interfaces and classes that are responsible for http requests only.

Interfaces in javax.servlet package

- Servlet
- ServletRequest
- ServletResponse
- RequestDispatcher
- ServletConfig
- ServletContext
- SingleThreadModel
- Filter
- FilterConfig
- FilterChain
- ServletRequestListener
- ServletRequestAttributeListener
- ServletContextListener
- ServletContextAttributeListener

Classes in javax.servlet package

- GenericServlet
- ServletInputStream
- ServletOutputStream
- ServletRequestWrapper
- ServletResponseWrapper
- ServletRequestEvent
- ServletContextEvent
- ServletRequestAttributeEvent
- ServletContextAttributeEvent
- ServletException
- UnavailableException

Interfaces in javax.servlet.http package

- HttpServletRequest
- HttpServletResponse
- HttpSession
- HttpSessionListener
- HttpSessionAttributeListener
- HttpSessionBindingListener
- HttpSessionActivationListener
- HttpSessionContext (deprecated now)

Classes in javax.servlet.http package

1. HttpServlet
2. Cookie
3. HttpServletRequestWrapper
4. HttpServletResponseWrapper
5. HttpSessionEvent
6. HttpSessionBindingEvent
7. HttpUtils (deprecated now)

Servlet Interface

- **Servlet interface provides** common behavior to all the servlets. Servlet interface defines methods that all servlets must implement.
- Servlet interface needs to be implemented for creating any servlet (either directly or indirectly).
- It provides 3 life cycle methods that are used to initialize the servlet, to service the requests, and to destroy the servlet and 2 non-life cycle methods.

Methods of Servlet interface

Method	Description
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.

Servlet Example by implementing Servlet interface

```
import java.io.*;
import javax.servlet.*;

public class First implements Servlet {
      ServletConfig config=null;

public void init(ServletConfig config){
     this.config=config;

   System.out.println("servlet is initialized");
}
public void service(ServletRequest req,ServletResponse res)
  throws IOException,ServletException{
   res.setContentType("text/html");
   PrintWriter out=res.getWriter();
out.print("<html><body>");
out.print("<b>hello simple servlet</b>");
out.print("</body></html>");
  }
public void destroy() {
System.out.println("servlet is destroyed");
}
public ServletConfig getServletConfig() {
return config;
}
public String getServletInfo() {
return "copyright 2007-1010";}
}
```

GenericServlet class

- implements **Servlet**, **ServletConfig** and **Serializable** interfaces. It provides the implementation of all the methods of these interfaces except the service method.
- can handle any type of request so it is protocol-independent.

- Methods

1. **public void init(ServletConfig config)** is used to initialize the servlet.
2. **public abstract void service(ServletRequest request, ServletResponse response)** provides service for the incoming request. It is invoked at each time when user requests for a servlet.
3. **public void destroy()** is invoked only once throughout the life cycle and indicates that servlet is being destroyed.
4. **public ServletConfig getServletConfig()** returns the object of ServletConfig.
5. **public String getServletInfo()** returns information about servlet such as writer, copyright, version etc.

1. **public void init()** it is a convenient method for the servlet programmers, now there is no need to call super.init(config)
2. **public ServletContext getServletContext()** returns the object of ServletContext.
3. **public String getInitParameter(String name)** returns the parameter value for the given parameter name.
4. **public Enumeration getInitParameterNames()** returns all the parameters defined in the web.xml file.
5. **public String getServletName()** returns the name of the servlet object.
6. **public void log(String msg)** writes the given message in the servlet log file.
7. **public void log(String msg,Throwable t)** writes the explanatory message in the servlet log file and a stack trace.

Servlet Example by inheriting the GenericServlet class

```
import java.io.*;
import javax.servlet.*;

public class First extends GenericServlet  {
public void service(ServletRequest req,ServletResponse res)  throws IOException,ServletException {
      res.setContentType("text/html");
      PrintWriter out=res.getWriter();
      out.print("<html><body>");
      out.print("<b>hello generic servlet</b>");
      out.print("</body></html>");
    }
}
```

HttpServlet class

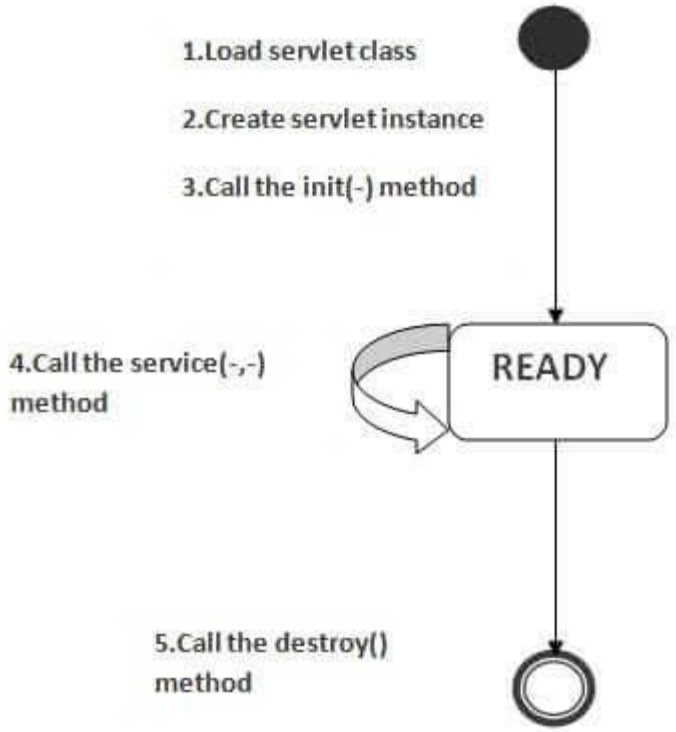
- The HttpServlet class extends the GenericServlet class and implements Serializable interface. It provides http specific methods such as doGet, doPost, doHead, doTrace etc.
- Methods of HttpServlet class

1. **public void service(ServletRequest req,ServletResponse res)** dispatches the request to the protected service method by converting the request and response object into http type.
2. **protected void service(HttpServletRequest req, HttpServletResponse res)** receives the request from the service method, and dispatches the request to the doXXX() method depending on the incoming http request type.
3. **protected void doGet(HttpServletRequest req, HttpServletResponse res)** handles the GET request. It is invoked by the web container.
4. **protected void doPost(HttpServletRequest req, HttpServletResponse res)** handles the POST request. It is invoked by the web container.

1. **protected void doHead(HttpServletRequest req, HttpServletResponse res)** handles the HEAD request. It is invoked by the web container.
2. **protected void doOptions(HttpServletRequest req, HttpServletResponse res)** handles the OPTIONS request. It is invoked by the web container.
3. **protected void doPut(HttpServletRequest req, HttpServletResponse res)** handles the PUT request. It is invoked by the web container.
4. **protected void doTrace(HttpServletRequest req, HttpServletResponse res)** handles the TRACE request. It is invoked by the web container.
5. **protected void doDelete(HttpServletRequest req, HttpServletResponse res)** handles the DELETE request. It is invoked by the web container.
6. **protected long getLastModified(HttpServletRequest req)** returns the time when HttpServletRequest was last modified since midnight January 1, 1970 GMT.

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.



Servlet Life Cycle

- 1) Servlet class is loaded
- The classloader is responsible to load the servlet class. The servlet class is loaded when the first request for the servlet is received by the web container.
- 2) Servlet instance is created
- The web container creates the instance of a servlet after loading the servlet class. The servlet instance is created only once in the servlet life cycle.
- 3) init method is invoked
- The web container calls the init method only once after creating the servlet instance. The init method is used to initialize the servlet. It is the life cycle method of the javax.servlet.Servlet interface. Syntax of the init method is given below:

- **public void** init(ServletConfig config) **throws** ServletException
- 4) service method is invoked
- The web container calls the service method each time when request for the servlet is received. If servlet is not initialized, it follows the first three steps as described above then calls the service method. If servlet is initialized, it calls the service method. Notice that servlet is initialized only once. The syntax of the service method of the Servlet interface is given below:
- **public void** service(ServletRequest request, ServletResponse response)
- **throws** ServletException, IOException
- 5) destroy method is invoked
- The web container calls the destroy method before removing the servlet instance from the service. It gives the servlet an opportunity to clean up any resource for example memory, thread etc. The syntax of the destroy method of the Servlet interface is given below:
- **public void** destroy()

Steps to create a servlet example

- There are given 6 steps to create a **servlet example**. These steps are required for all the servers.

1. The servlet example can be created by three ways:

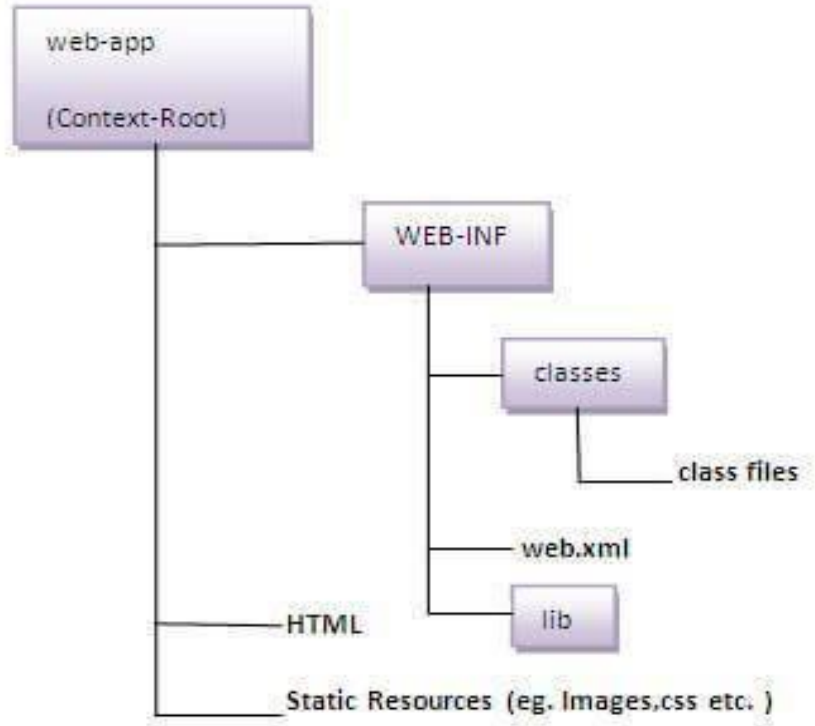
- By implementing Servlet interface,
- By inheriting GenericServlet class, (or)
- By inheriting HttpServlet class
- The mostly used approach is by extending HttpServlet because it provides http request specific method such as doGet(), doPost(), doHead() etc.
- Here, we are going to use **apache tomcat server** in this example.

The steps are as follows:

1. Create a directory structure
2. Create a Servlet
3. Compile the Servlet
4. Create a deployment descriptor
5. Start the server and deploy the project
6. Access the servlet

1. Create a directory structures

- The **directory structure** defines that where to put the different types of files so that web container may get the information and respond to the client.
- the servlet class file must be in the classes folder. The web.xml file must be under the WEB-INF folder.



2. Create a Servlet

- There are three ways to create the servlet.

1. By implementing the Servlet interface
2. By inheriting the GenericServlet class
3. By inheriting the HttpServlet class

- The HttpServlet class is widely used to create the servlet because it provides methods to handle http requests such as doGet(), doPost, doHead() etc.

- **DemoServlet.java**

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class DemoServlet extends HttpServlet {
public void doGet(HttpServletRequest req,HttpServletResponse res)
throws ServletException,IOException  {
res.setContentType("text/html");//setting the content type
PrintWriter pw=res.getWriter();//get the stream to write the data
  //writing html in the stream
pw.println("<html><body>");
pw.println("Welcome to servlet");
pw.println("</body></html>");
  pw.close();//closing the stream
}
}
```

3. Compile the servlet

- For compiling the Servlet, jar file is required to be loaded. Different Servers provide different jar files:
- Two ways to load the jar file

1. set classpath
2. paste the jar file in JRE/lib/ext folder

- Put the java file in any folder. After compiling the java file, paste the class file of servlet in **WEB-INF/classes** directory.

Jar file	Server
1) servlet-api.jar	Apache Tomcat
2) weblogic.jar	Weblogic
3) javaee.jar	Glassfish
4) javaee.jar	JBoss

4. Create the deployment descriptor (web.xml file)

- The **deployment descriptor** is an xml file, from which Web Container gets the information about the servet to be invoked.
- The web container uses the Parser to get the information from the web.xml file. There are many xml parsers such as SAX, DOM and Pull.
- There are many elements in the web.xml file. Here is given some necessary elements to run the simple servlet program.

5. Description of the elements of web.xml file

- **web.xml file**

```
<web-app>
    <servlet>
        <servlet-name>Servlet Name</servlet-name>
        <servlet-class>DemoServlet</servlet-class>
     </servlet>
    <servlet-mapping>
    <servlet-name>Servlet Namel</servlet-name>
     </servlet-mapping>
</web-app>
```

- **<web-app>** represents the whole application.
- **<servlet>** is sub element of <web-app> and represents the servlet.
- **<servlet-name>** is sub element of <servlet> represents the name of the servlet.
- **<servlet-class>** is sub element of <servlet> represents the class of the servlet.
- **<servlet-mapping>** is sub element of <web-app>. It is used to map the servlet.
- **<url-pattern>** is sub element of <servlet-mapping>. This pattern is used at client side to invoke the servlet.

6. Start the Server and deploy the project

- To start Apache Tomcat server, double click on the startup.bat file under apache-tomcat/bin directory.
- One Time Configuration for Apache Tomcat Server
- You need to perform 2 tasks:
- set JAVA_HOME or JRE_HOME in environment variable (It is required to start server).
- Change the port number of tomcat (optional). It is required if another server is running on same port (8080).

How to access the servlet

- http://localhost:8080/demo/welcome

ServletRequest Interface

- An object of ServletRequest is used to provide the client request information to a servlet such as content type, content length, parameter names and values, header informations, attributes etc.

index.html

```
<form action="welcome" method="get">
Enter your name<input type="text" name="name"><br>
<input type="submit" value="login">
</form>
```

Example

- **DemoServ.java**

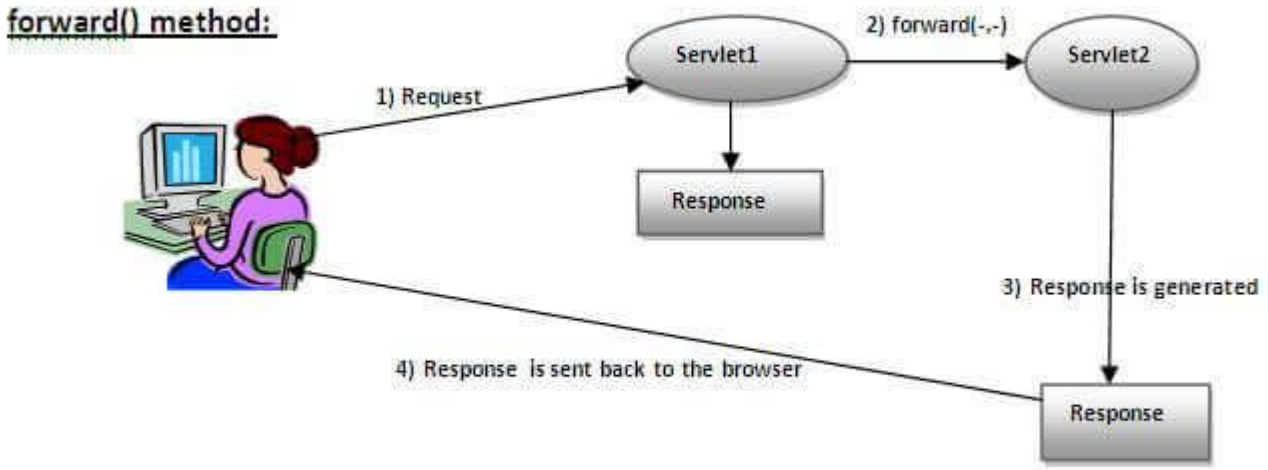
```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class DemoServ extends HttpServlet   {

   public void doGet(HttpServletRequest req,HttpServletResponse res)  throws ServletException,IOExcepti
   on    {
         res.setContentType("text/html");
         PrintWriter pw=res.getWriter();
         String name=req.getParameter("name");//will return value
         pw.println("Welcome "+name);
         pw.close();
      }
}
```

RequestDispatcher in Servlet

- 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.

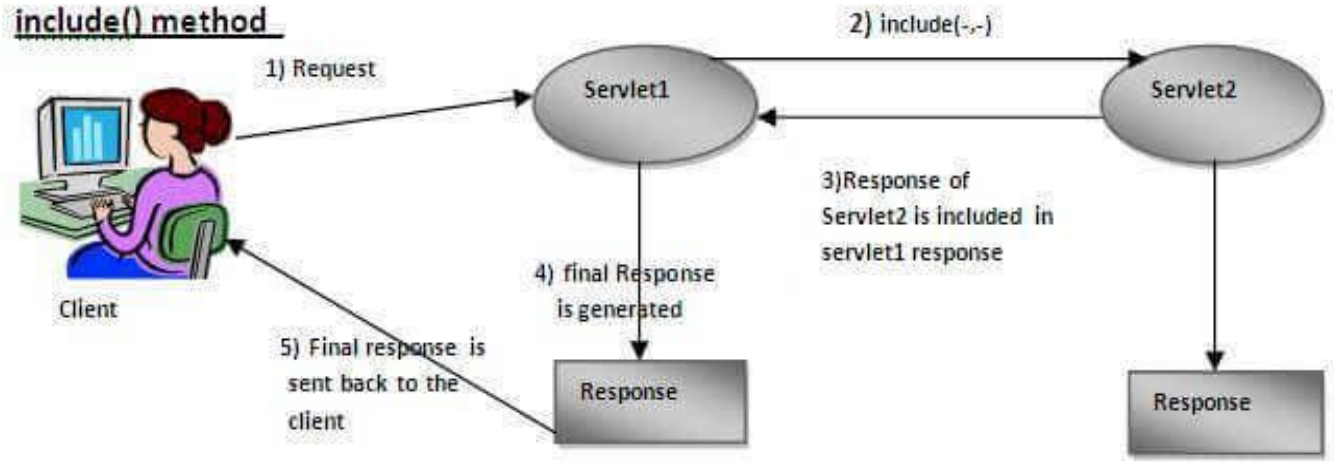
forward method

response of second servlet is sent to the client. Response of the first servlet is not displayed to the user



Include method

- response of second servlet is included in the response of the first servlet that is being sent to the client.



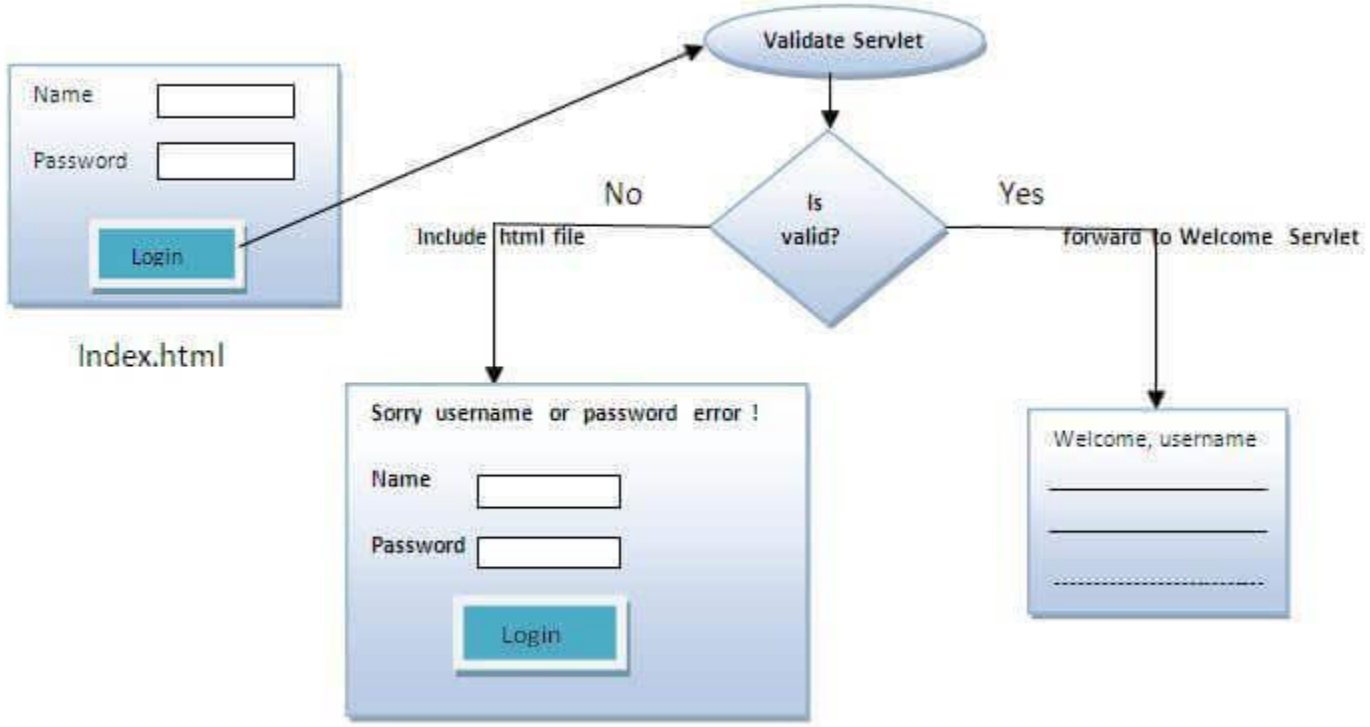
How to get the object of RequestDispatcher

- The getRequestDispatcher() method of ServletRequest interface returns the object of RequestDispatcher. Syntax:
- Syntax of getRequestDispatcher method
- **public** RequestDispatcher getRequestDispatcher(String resource);
- Example of using getRequestDispatcher method
- RequestDispatcher rd=request.getRequestDispatcher("servlet2");
- //servlet2 is the url-pattern of the second servlet
- rd.forward(request, response);//method may be include or forward

Example of RequestDispatcher interface

- In this example, validating the password entered by the user. If password is servlet, it will forward the request to the WelcomeServlet, otherwise will show an error message: sorry username or password error!. In this program, we are cheking for hardcoded information.
- **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.

Example


Validate Servlet
Name
Password
Login
Index.html
No
Is
valid?
Yes
Include html file
forward to Welcome Servlet
Sorry username or password error !
Name
Password
Login
Welcome, username

Example

- **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") {
```

PRESIDENCY UNIVERSITY
Private University Estd. in Karnataka

- **WelcomeServlet.java**

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class WelcomeServlet extends HttpServlet {

  public void doPost(HttpServletRequest request, HttpServletRespose 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

- 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

- **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();

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

pw.close();
}}
```

Creating custom google search using sendRedirect

In this example, we are using sendRedirect method to send request to google server with the

ServletConfig 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.
- 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.

Example of ServletConfig to get initialization parameter

- getting the one initialization parameter from the web.xml file and printing this information in the servlet.

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();
```

PRESIDENCY UNIVERSITY
Private University Estd. in Karnataka State by Act No. 41 of 2013

Example of ServletConfig to get all the initialization parameters

- In this example, we are getting all the initialization parameter from the web.xml file and printing this information in the servlet.
-

DemoServlet.java

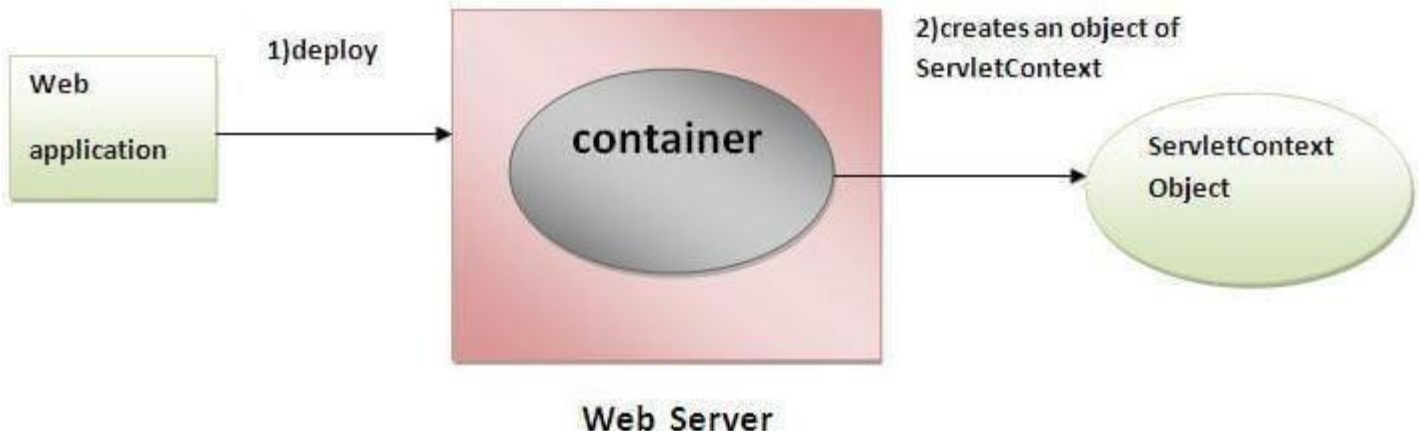
```
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Enumeration;

import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


public class DemoServlet extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
```

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.
- **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.
- Use
  1. The object of ServletContext provides an interface between the

Difference between ServletConfig and ServletContext

How to get the object of ServletContext interface



The servletconfig object refers to the single servlet whereas servletcontext object refers to the whole web application.

How to get the object of ServletContext interface

- **getServletContext() method** of ServletConfig interface returns the object of ServletContext.
- **getServletContext() method** of GenericServlet class returns the object of ServletContext.
- Syntax of getServletContext() method
- **public** ServletContext getServletContext()
- Example of getServletContext() method
- //We can get the ServletContext object from ServletConfig object
- ServletContext application=getServletConfig().getServletContext();
-
- //Another convenient way to get the ServletContext object
- ServletContext application=getServletContext();

Example of ServletContext to get the initialization parameter

- In this example, we are getting the initialization parameter from the web.xml file and printing the value of the initialization parameter. Notice that the object of ServletContext represents the application scope. So if we change the value of the parameter from the web.xml file, all the servlet classes will get the changed value. So we don't need to modify the servlet. So it is better to have the common information for most of the servlets in the web.xml file by context-param element
- **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
{
```

Example of ServletContext to get all the initialization parameters

- getting all the initialization parameter from the web.xml file. For getting all the parameters, we have used the getInitParameterNames() method in the servlet class.
- **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 out=res.getWriter();
ServletContext context=getServletContext();
```

Attribute in Servlet

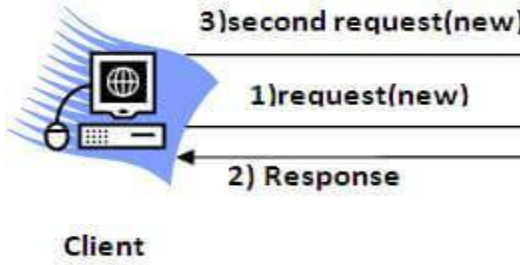
- An **attribute in servlet** is an object that can be set, get or removed from one of the following scopes:
- request scope
- session scope
- application scope
- The servlet programmer can pass informations from one servlet to another using attributes. It is just like passing object from one class to another so that we can reuse the same object again and again.

Example of ServletContext to set and get attribute

- In this example, we are setting the attribute in the application scope and getting that value from another servlet.
- DemoServlet1.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class DemoServlet1 extends HttpServlet{
public void doGet(HttpServletRequest req,HttpServletResponse res)
{
try{
```

Session Tracking/Session Management



- Session - a particular interval of time.
- **Session Tracking** is a way to maintain state (data) of an user.
- Http protocol is a stateless so 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

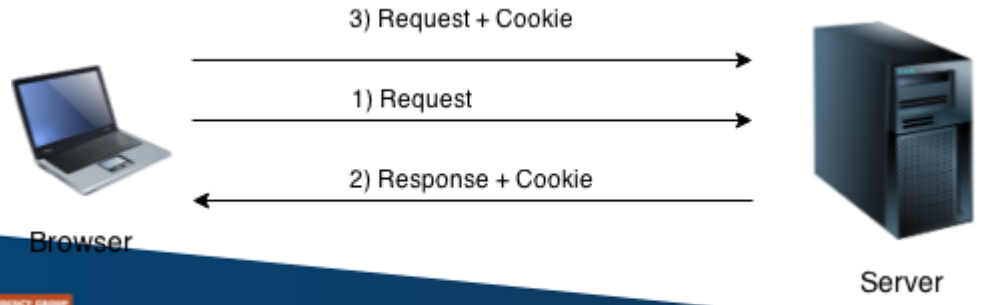
- There are four techniques used in Session tracking:

1. **Cookies**
2. **Hidden Form Field**
3. **URL Rewriting**
4. **HttpSession**

Cookies

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.



Types of Cookie

There are 2 types of cookies in servlets.

1. Non-persistent cookie
2. Persistent cookie

- Non-persistent cookie

  It is **valid for single session** only. It is removed each time when user closes the browser.

- Persistent cookie

  It is **valid for multiple session** . It is not removed each time when user closes the browser. It is removed only if user logout or signout.

- **Advantage of Cookies**
  - Simplest technique of maintaining the state.
  - Cookies are maintained at client side.

PRESIDENCY UNIVERSITY
Private University Estd. in Karnataka State by Act No. 41 of 2013

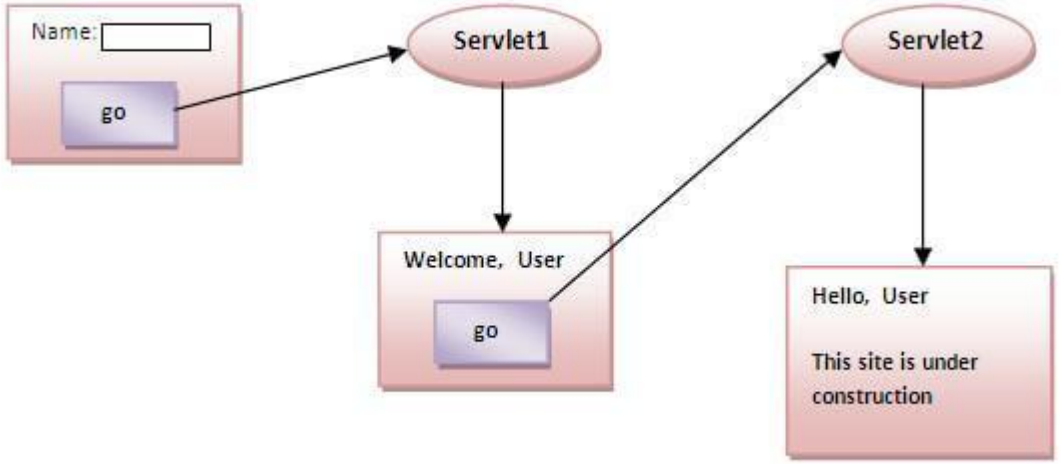
How create/delete/get/cookie

- create cookie.
- Cookie ck=**new** Cookie("user","sonoo jaiswal");//creating cookie object
- response.addCookie(ck);//adding cookie in the response
- delete Cookie
- It is mainly used to logout or signout the user.
- Cookie ck=**new** Cookie("user","");//deleting value of cookie
- ck.setMaxAge(0);//changing the maximum age to 0 seconds
- response.addCookie(ck);//adding cookie in the response
- get Cookies
- to get all the cookies.

PRESIDENCY UNIVERSITY
Private University Estd. in Karnataka State by Act No. 41 of 2013
40 YEARS OF ACADEMIC WISDOM

Example

- storing the name of the user in the cookie object and accessing it in another servlet. As we know well that session corresponds to the particular user. So if you access it from too many browsers with different values, you will get the different value.



index.html

- <form action="servlet1" method="post">
- Name:<input type="text" name="userName"/>
- <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
```

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);}
    }
}
```

PRESIDENCY UNIVERSITY
Private University Estd. in Karnataka State by Act No. 41 of 2013
40 YEARS OF ACADEMIC WISDOM

web.xml

https://www.javatpoint.com/servlet-login-and-logout-example-using-cookies

```
<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>
```

PRESIDENCY UNIVERSITY
Private University Estd. in Karnataka State by Act No. 41 of 2013
40 YEARS OF ACADEMIC WISDOM

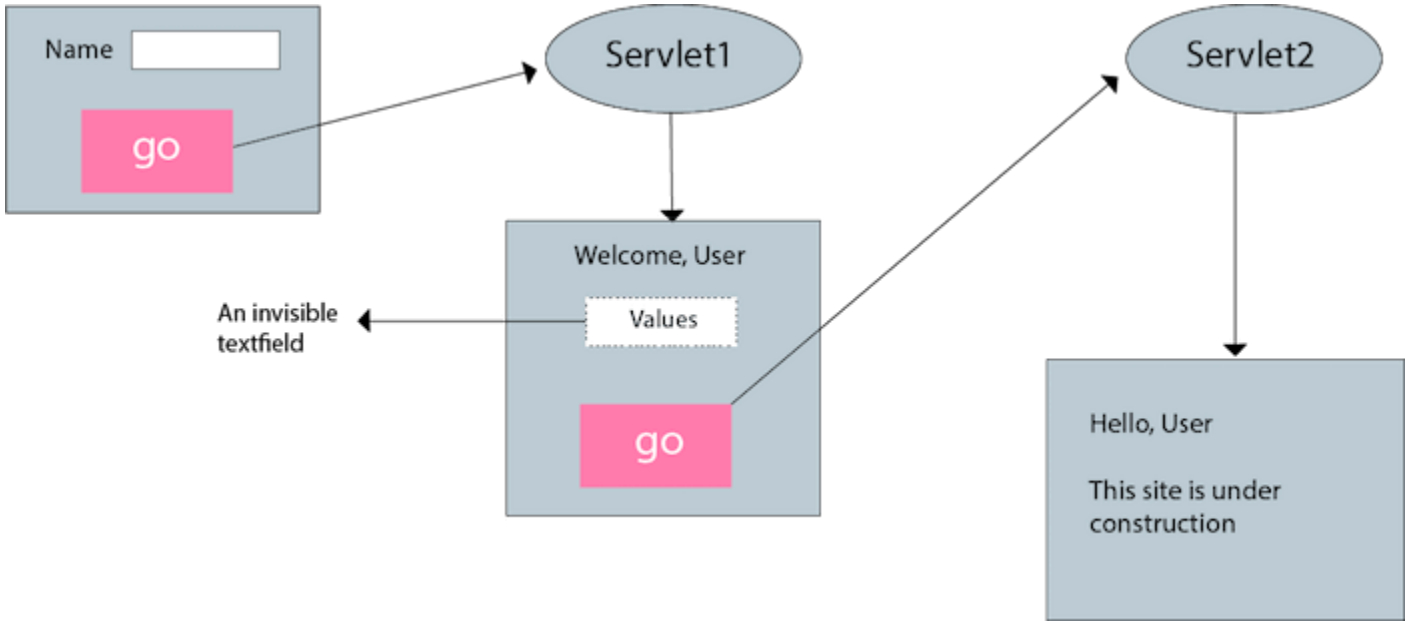
Hidden Form Field

- In case of Hidden Form Field **a hidden (invisible) textfield** is used for maintaining the state of an user.
- In such case, we store the information in the hidden field and get it from another servlet. This approach is better if we have to submit form in all the pages and we don't want to depend on the browser.
- Let's see the code to store value in hidden field.
- <input type="hidden" name="uname" value="Vimal Jaiswal">
- Here, uname is the hidden field name and Vimal Jaiswal is the hidden field value.

Hidden Field

- Real application of hidden form field
- It is widely used in comment form of a website. In such case, we store page id or page name in the hidden field so that each page can be uniquely identified.
- Advantage of Hidden Form Field
- It will always work whether cookie is disabled or not.
- Disadvantage of Hidden Form Field:
- It is maintained at server side.
- Extra form submission is required on each pages.
- Only textual information can be used.

Example of using Hidden Form Field



Ex

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 re
  sponse){
    try{
```

Ex

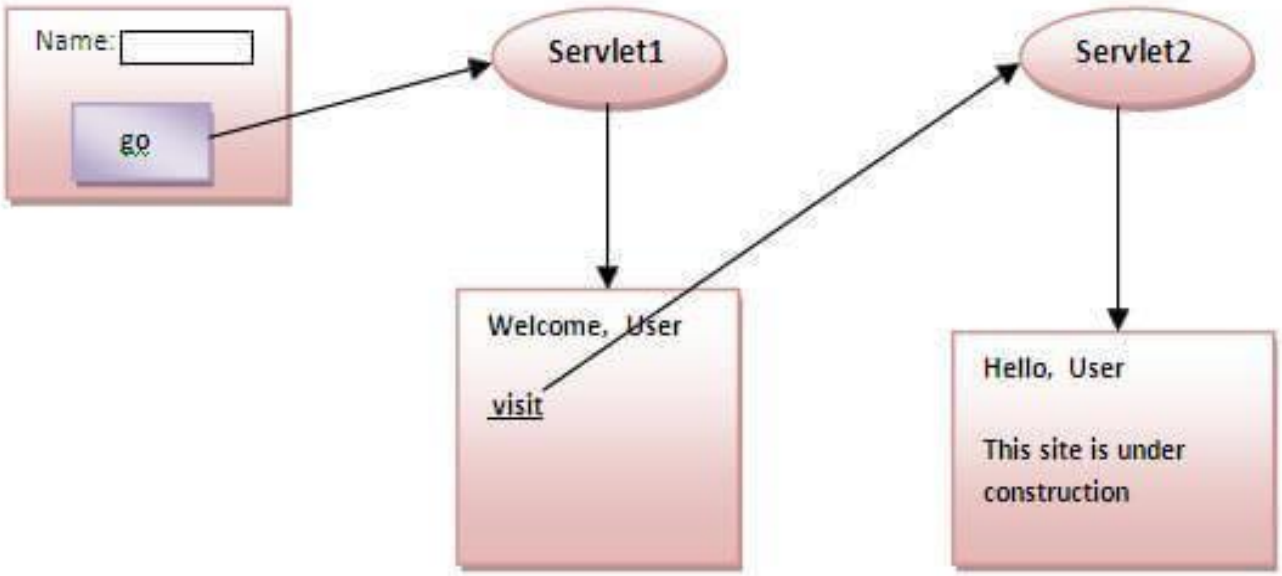
- SecondServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SecondServlet extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse re
  sponse)
      try{
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      //Getting the value from the hidden field
      String n=request.getParameter("uname");
      out.print("Hello "+n);
```

URL rewriting

- URL rewriting, we append a token or identifier to the URL of the next Servlet or the next resource. We can send parameter name/value pairs using the following format:
- url?name1=value1&name2=value2&??
- A name and a value is separated using an equal = sign, a parameter name/value pair is separated from another parameter using the ampersand(&). When the user clicks the hyperlink, the parameter name/value pairs Advantage of URL Rewriting
- It will always work whether cookie is disabled or not (browser independent).
- Extra form submission is not required on each pages.
- Disadvantage of URL Rewriting
- It will work only with links.

Example


Name:
go
Servlet1
Servlet2
Welcome, User
visit
Hello, User
This site is under construction

Example

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();
```

Example

- SecondServlet.java

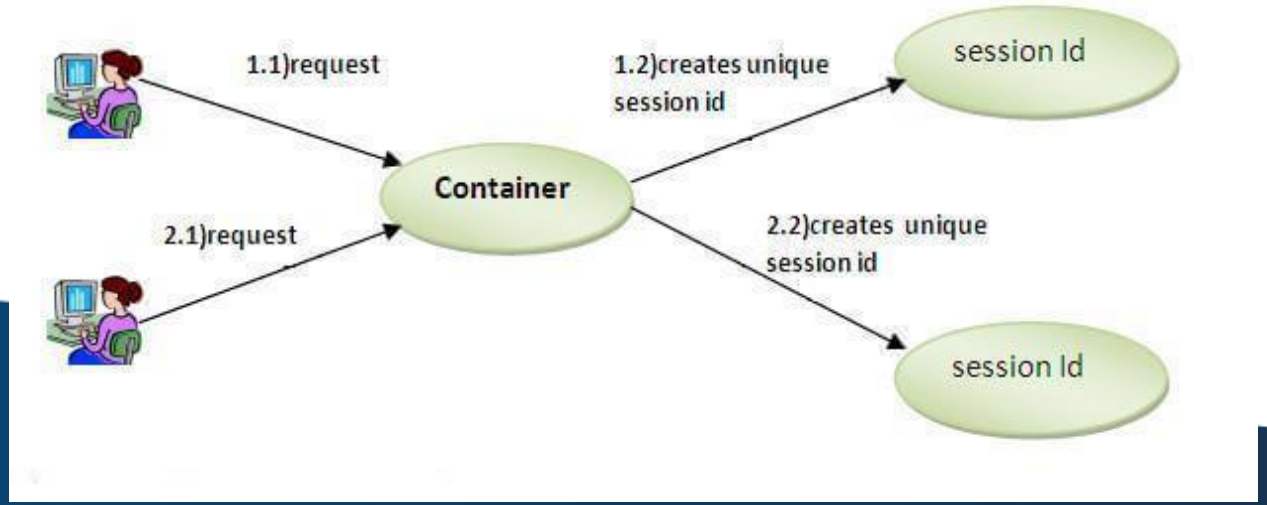
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SecondServlet extends HttpServlet {

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

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    //getting value from the query string
    String n=request.getParameter("uname");
    out.print("Hello "+n);

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

PRESIDENCY UNIVERSITY
Private University Estd. in Karnataka State by Act No. 41 of 2013

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:

1. bind objects
2. view and manipulate information about a session, such as the session identifier, creation time, and last accessed time.



HttpSession

- 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();
```

JSP

- **JSP** is a technology
- 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.

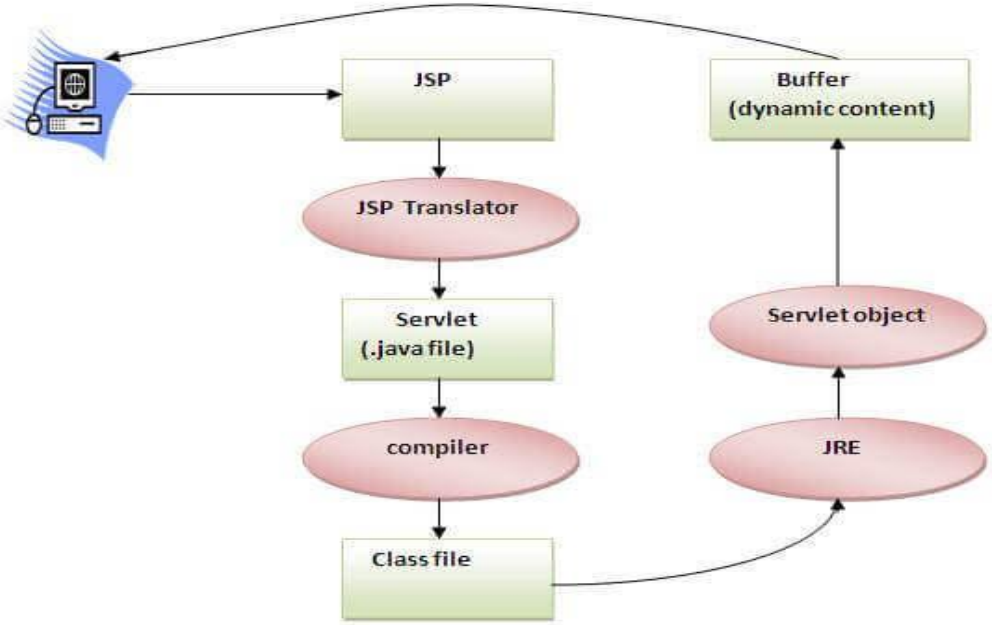
Advantages of JSP over Servlet

1) Extension to Servlet

- JSP technology is the extension to Servlet technology. We can use all the features of the Servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.
- 2) Easy to maintain
- JSP can be easily managed because we can easily separate our business logic with presentation logic. In Servlet technology, we mix our business logic with the presentation logic.
- 3) Fast Development: No need to recompile and redeploy
- If JSP page is modified, we don't need to recompile and redeploy the project. The Servlet code needs to be updated and recompiled if we have to change the look and feel of the application.

PRESIDENCY UNIVERSITY
Private University Estd. in Karnataka State by Act No. 41 of 2013

Lifecycle of a JSP Page

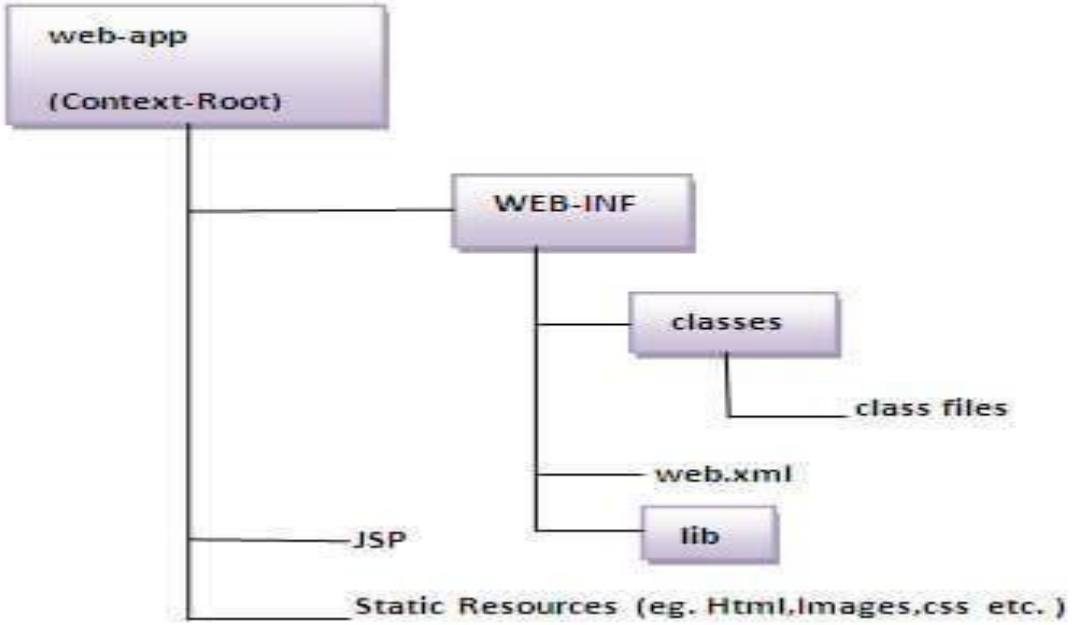


- The JSP pages follow these phases:
- Translation of JSP Page
- Compilation of JSP Page
- Classloading (the classloader loads class file)
- Instantiation (Object of the Generated Servlet is created
- Initialization (the container invokes jspInit() method).
- Request processing (the container invokes _jspService() method).
- Destroy (the container invokes jspDestroy() method).
- Note: jspInit(), _jspService() and jspDestroy() are the life cycle methods of JSP.
- JSP page is translated into Servlet by the help of JSP translator. The JSP translator is a part of the web server which is responsible for translating the JSP page into Servlet. After that, Servlet page is compiled by the compiler and gets converted into the class file. Moreover, all the processes that happen in Servlet are performed on JSP later like initialization, committing response to the browser and destroy.

Creating a simple JSP Page

- To create the first JSP page, write some HTML code as given below, and save it by .jsp extension. We have saved this file as index.jsp. Put it in a folder and paste the folder in the web-apps directory in apache tomcat to run the JSP page.
- **index.jsp**Let's see the simple example of JSP where we are using the scriptlet tag to put Java code in the JSP page.
- <html>
- <body>
- <% out.print(2*5); %>
- </body>
- </html>
- How to run a simple JSP Page?
- Follow the following steps to execute this JSP page:
- Start the server
- Put the JSP file in a folder and deploy on the server
- Visit the browser by the URL http://localhost:portno/contextRoot/jspfile, for example, http://localhost:8888/myapplication/index.jsp

The Directory structure of JSP



The JSP API

- The JSP API consists of two packages:

1. javax.servlet.jsp
2. javax.servlet.jsp.tagext

- **javax.servlet.jsp package**
- The javax.servlet.jsp package has two interfaces and classes.The two interfaces are as follows:

1. JspPage
2. HttpJspPage

- The classes are as follows:

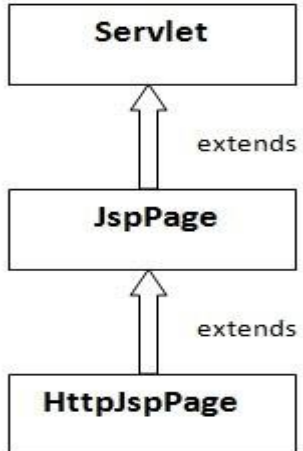
➢JspWriter
➢PageContext
➢JspFactory

PRESIDENCY UNIVERSITY
Private University Estd. in Karnataka State by Act No. 41 of 2013
40 YEARS OF ACADEMIC WISDOM

JSP Page Interface

- According to the JSP specification, all the generated servlet classes must implement the JspPage interface. It extends the Servlet interface. It provides two life cycle methods.

1. **public void jspInit():** It is invoked only once during the life cycle of the JSP when JSP page is requested firstly. It is used to perform initialization. It is same as the init() method of Servlet interface.
2. **public void jspDestroy():** It is invoked only once during the life cycle of the JSP before the JSP page is destroyed. It can be used to perform some clean up operation.



HttpJspPage interface

- **3.public void _jspService():** It is invoked each time when request for the JSP page comes to the container. It is used to process the request. The underscore _ signifies that you cannot override this method.

JSP Scriptlet tag (Scripting elements)

In JSP, java code can be written inside the jsp page using the scriptlet tag

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

```
<html>
<body>
<% out.print("welcome to jsp"); %>
</body>
</html>
```

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

PRESIDENCY UNIVERSITY
40 YEARS OF ACADEMIC WISDOM
Private University Established in Karnataka State by Act No. 41 of 2013

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 - simply displaying a welcome message.

<html>

<body>

<%= "welcome to jsp" %>

</body>

</html>

- Example – display current time

<html>

<body>

Current Time: **<%= java.util.Calendar.getInstance().getTime() %>**

</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

<%! field or method declaration %>

Example of JSP declaration tag that declares field

- declaring the field and printing the value of the declared field using the jsp expression tag.
- index.jsp

<html>
<body>
<%! int data=50; %>

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

Object	Type
out	JspWriter
request	HttpServletRequest
response	HttpServletResponse
config	ServletConfig
application	ServletContext
session	HttpSession
pageContext	PageContext
page	Object

JSP out implicit object

- For writing any data to the buffer, JSP provides an implicit object named out. It is the object of JspWriter. In case of servlet you need to write:
- PrintWriter out=response.getWriter(); Example of out implicit object
- In this example we are simply displaying date and time.

index.jsp

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

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.
- It can also be used to set, get and remove attributes from the jsp request scope.
- Example of JSP request implicit object
- **index.html**

```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

JSP 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.
- Example of response implicit object
- **index.html**

```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

- **welcome.jsp**

JSP config implicit object

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

```
<form action="welcome">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

- **web.xml file**

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

PRESIDENCY UNIVERSITY
Private University Estd. in Karnataka State by Act No. 41 Of 2013

JSP application implicit object

- In JSP, application is an implicit object of type *ServletContext*.
- 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.
- This initialization parameter can be used by all jsp pages.
- Example of application implicit object:
- **index.html**

```
<form action="welcome">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

web.xml file

```
<web-app>

<servlet>
  <servlet-name>sonoojaiswal</servlet-name>
  <jsp-file>/welcome.jsp</jsp-file>
</servlet>
```

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.Example of session implicit object
- **index.html**

```
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
</body>
</html>
```

welcome.jsp

```
<html>
<body>
<%
```

pageContext implicit object

- In JSP, pageContext is an implicit object of type PageContext class.The pageContext object can be used to set,get or remove attribute from one of the following scopes:page
- request
- session
- application
- In JSP, page scope is the default scope.Example of pageContext implicit object

index.html

```
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
</body>
```

PRESIDENCY UNIVERSITY
Private University Estd. in Karnataka State by Act No. 41 of 2013

page implicit object:

- In JSP, 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;For using this object it must be cast to Servlet type.For example:<% (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

- In JSP, 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.It is better to learn it after page directive. Let's see a simple example:Example of exception implicit object:
- **error.jsp**

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

Sorry following exception occured:<%= exception %>

</body>
</html>
```

JSP directives

- The **jsp directives** are messages that tells the web container how to translate a JSP page into the corresponding servlet.
- 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

PRESIDENCY UNIVERSITY
40 YEARS OF ACADEMIC WISDOM

import

- The import attribute is used to import class,interface or all the members of a package.It is similar to import keyword in java class or interface.Example of import attribute
- <html>
- <body>
-
- <%@ page **import**="java.util.Date" %>
- Today is: <%= **new** Date() %>
-
- </body>
- </html>

contentType

- The contentType attribute defines the MIME(Multipurpose Internet Mail Extension) type of the HTTP response.The default value is "text/html;charset=ISO-8859-1".
- Example of contentType attribute

```
<html>
<body>

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

</body>
</html>
```

Extends & info

- The extends attribute defines the parent class that will be inherited by the generated servlet.It is rarely used.
- This attribute simply sets the information of the JSP page which is retrieved later by using getServletInfo() method of Servlet interface.
- Example of info attribute

```
<html>
<body>

<%@ page info="composed by Sonoo Jaiswal" %>
Today is: <%= new java.util.Date() %>

</body>
</html>
```

language & isELIgnored & isThreadSafe

- The language attribute specifies the scripting language used in the JSP page. The default value is "java".
- We can ignore the Expression Language (EL) in jsp by the isELIgnored attribute. By default its value is false i.e. Expression Language is enabled by default. We see Expression Language later.
- <%@ page isELIgnored="true" %>//Now EL will be ignored
- Servlet and JSP both are multithreaded.If you want to control this behaviour of JSP page, you can use isThreadSafe attribute of page directive.The value of isThreadSafe value is true.If you make it false, the web container will serialize the multiple requests, i.e. it will wait until the JSP finishes responding to a request before passing another request to it.If you make the value of isThreadSafe attribute like:<%@ page isThreadSafe="false" %>
- The web container in such a case, will generate the servlet as:
- **public class** SimplePage_jsp **extends** HttpJspBase
- **implements** SingleThreadModel{
- .....
- }

errorPage & isErrorPage

- The errorPage attribute is used to define the error page, if exception occurs in the current page, it will be redirected to the error page.
- Example of errorPage attribute

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

<%@ page errorPage="myerrorpage.jsp" %>

 <%= 100/0 %>

</body>
</html>
```

- The isErrorPage attribute is used to declare that the current page is the error page.

Note: The exception object can only be used in the error page.

Example of isErrorPage attribute

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

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

PRESIDENCY UNIVERSITY
Private University Estd. in Karnataka State by Act No. 41 of 2013
40 YEARS OF ACADEMIC WISDOM

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" %>
```

PRESIDENCY UNIVERSITY
Private University Estd. in Karnataka State by Act No. 41 of 2013
40 YEARS OF ACADEMIC WISDOM

JSP Taglib directive

- 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.
- Syntax JSP Taglib directive
- <%@ taglib uri="uriofthetaglibrary" prefix="prefixoftaglibrary" %>
- Example of JSP Taglib directive
- In this example, we are using our tag named currentDate. To use this tag we must specify the taglib directive so the container may get information about the tag.

```
<html>
<body>

<%@ taglib uri="http://www.javatpoint.com/tags" prefix="mytag" %>

<mytag:currentDate/>

</body>
</html>
```

PRESIDENCY UNIVERSITY
Private University in Karnataka State by Act No. 41 of 2013
40 YEARS OF ACADEMIC WISDOM

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. There may occur exception any time in your web application. So handling exceptions is a safer side for the web developer. In JSP, there are two ways to perform exception handling:

1. By **errorPage** and **isErrorPage** attributes of page directive
2. By **<error-page>** element in web.xml file

Example of exception handling in jsp by the elements of page directive

In this case, you must define and create a page to handle the exceptions, as in the error.jsp page.

The pages where may occur exception, define the errorPage attribute of page directive, as in the process.jsp page.

There are 3 files:

index.jsp for input values

process.jsp for dividing the two numbers and displaying the result

error.jsp for handling the exception

index.jsp

JSP Action Tags

- 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 belo

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:forward action tag

The jsp:forward action tag is used to forward the request to another resource it may be jsp, html or another resource.

Syntax of jsp:forward action tag without parameter

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

Syntax of jsp:forward action tag with parameter

```
<jsp:forward page="relativeURL | <%= expression %>">
 <jsp:param name="parametername" value="parametervalue | <%=expression%>" />
</jsp:forward>
```

Example of jsp:forward action tag without parameter

index.jsp

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

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

printdate.jsp

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

Example of jsp:forward action tag with parameter

- In this example, we are forwarding the request to the printdate.jsp file with parameter and printdate.jsp file prints the parameter value with date and time.

- index.jsp

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

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

</body>
</html>
```

- printdate.jsp
- <html>
- <body>
- <% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
- <%= request.getParameter("name") %>

jsp:include action tag

- The **jsp:include action tag** is used to include the content of another resource it may be jsp, html or servlet.
- The jsp include action tag includes the resource at request time so it is **better for dynamic pages** because there might be changes in future.
- The jsp:include tag can be used to include static as well as dynamic pages.
- Advantage of jsp:include action tag
- **Code reusability** : We can use a page many times such as including header and footer pages in all pages. So it saves a lot of time.
- Syntax of jsp:include action tag without parameter
- <jsp:include page="relativeURL | <%= expression %>" />

Syntax of jsp:include action tag with parameter

<jsp:include page="relativeURL | <%= expression %>">
<jsp:param name="parametername" value="parametervalue | <%=expressio

JavaBean

- A JavaBean is a Java class that should follow the following conventions:
- It should have a no-arg constructor.
- It should be Serializable.
- It should provide methods to set and get the values of the properties, known as getter and setter methods.
- Why use JavaBean?
- According to Java white paper, it is a reusable software component. A bean encapsulates many objects into one object so that we can access this object from multiple places. Moreover, it provides easy maintenance.
- Advantages of JavaBean

Example

//Employee.java

```
package mypack;
public class Employee implements java.io.Serializable{
private int id;
private String name;
public Employee(){}
public void setId(int id){this.id=id;}
public int getId(){return id;}
public void setName(String name){this.name=name;}
public String getName(){return name;}
}
```

- How to access the JavaBean class?

- To access the JavaBean class, we should use getter and setter methods.

```
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
```

PRESIDENCY UNIVERSITY
Private University Estd. in Karnataka State by Act No. 41 of 2013

jsp:useBean action tag

- The jsp:useBean action tag is used to locate or instantiate a bean class. If bean object of the Bean class is already created, it doesn't create the bean depending on the scope. But if object of bean is not created, it instantiates the bean.
- Syntax of 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>

example of jsp:useBean action tag

- Calculator.java (a simple Bean class)

```
package com.javatpoint;
public class Calculator{

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

}
```

- index.jsp file

```
<jsp:useBean id="obj" class="com.javatpoint.Calculator"/>

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

jsp:setProperty and jsp:getProperty action tags

- The setProperty and getProperty action tags are used for developing web application with Java Bean. In web devlopment, bean class is mostly used because it is a reusable software component that represents data.
- The jsp:setProperty action tag sets a property value or values in a bean using the setter method.
- Syntax of jsp:setProperty action tag

```
<jsp:setProperty name="instanceOfBean" property= "*"   |
property="propertyName" param="parameterName"  |
property="propertyName" value="{ string | <%= expression %>}"
/>
```

Example of bean development in JSP

index.html

```
<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>
```

process.jsp

```
<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" /><br>
```

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" %>

Tags	Description
c:out	It display the result of an expression, similar to the way <%=...%> tag work.
c:import	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	It sets the result of an expression under evaluation in a 'scope' variable.
c:remove	It is used for removing the specified scoped variable from a particular scope.
c:catch	It is used for Catches any Throwable exceptions

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. Hence they aren't evaluated as actual tags.
- Let's see the simple example of c:out tag:
- **<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>**

<html>
<head>
<title>Tag Example</title>
</head>
<body>

PRESIDENCY UNIVERSITY
Private University Estd. in Karnataka State by Act No. 41 of 2013
40 YEARS OF ACADEMIC WISDOM

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.
- For example: Using an import tag the content from a different FTP server and website can be accessed.
- Let's see the simple example of c:import tag:
- Play Video
- **<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>**

<html>
<head>
<title>Tag Example</title>
</head>
<body>

PRESIDENCY UNIVERSITY
Private University Estd. in Karnataka State by Act No. 41 of 2013
40 YEARS OF ACADEMIC WISDOM

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.
- Let's see the simple example of <c:set> tag:
- **<%@** taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" **%>**

<html>
<head>
<title>Core Tag Example**</title>**
</head>
<body>
<c:set var="Income" scope="session" value="${4000*4}"**/>**

PRESIDENCY UNIVERSITY
Private University Estd. in Karnataka State by Act No. 41 of 2013
40 YEARS OF ACADEMIC WISDOM

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.
- The <c:remove > tag removes the variable from either a first scope or a specified scope.
- Let's see the simple example of c:remove tag:
- **<%@** taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" **%>**

<html>
<head>
<title>Core Tag Example**</title>**
</head>
<body>
<c:set var="income" scope="session" value="${4000*4}"**/>**
** **Before Remove Value is: **<c:out** value="${income}"**/> **

PRESIDENCY UNIVERSITY
Private University Estd. in Karnataka State by Act No. 41 of 2013

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 to deal more easily with the problem occur in program.
- The < c:catch > tag catches any exceptions that occurs in a program body.
- Let's see the simple example of c:catch tag:
- **<%@** taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" **%>**

<html>
<head>
<title>Core Tag Example**</title>**
</head>
<body>

PRESIDENCY UNIVERSITY
Private University Estd. in Karnataka State by Act No. 41 of 2013
40 YEARS OF ACADEMIC WISDOM

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.
- Let's see the simple example of c:if tag:
- **<%@** taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" **%>**

<html>
<head>
<title>Core Tag Example**</title>**
</head>
<body>
<c:set var="income" scope="session" value="${4000*4}"**/>**

PRESIDENCY UNIVERSITY
Private University Estd. in Karnataka State by Act No. 41 of 2013
40 YEARS OF ACADEMIC WISDOM

JSTL Core <c:choose>, <c:when>, <c:otherwise> Tag

- The < c:choose > 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.
- The <c:when > is subtag of <choose > that will include its body if the condition evaluated be 'true'.
- The < c:otherwise > is also subtag of < choose > it follows &l;twhen > tags and runs only if all the prior condition evaluated is 'false'.
- The c:when and c:otherwise works like if-else statement. But it must be placed inside c:choose tag.
- %@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<html>

JSTL Core <c:forEach> Tag

- The <c:for each > 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. The < c:for each > tag is most commonly used tag because it iterates over a collection of object.
- Let's see the simple example of tag:
- **<%@** taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" **%>**

```
<html>
<head>
<title>Core Tag Example</title>
</head>
<body>
<c:forEach var="j" begin="1" end="3">
  Item <c:out value="${j}"/><p>
</c:forEach>
</body>
</html>
```

JSTL Core <c:forTokens> Tag

- he < c:forTokens > tag iterates over tokens which is separated by the supplied delimeters. It is used for break a string into tokens and iterate through each of the tokens to generate output.
- This tag has similar attributes as < c:forEach > tag except one additional attributes **delims** which is used for specifying the characters to be used as delimiters.
- Let's see the simple example of < c:forTokens > tag:
- **<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>**

<html>
<head>
<title>Core Tag Example</title>
</head>

PRESIDENCY UNIVERSITY
Private University Estd. in Karnataka State by Act No. 41 of 2013

JSTL Core <c:param> Tag

- The < c:param > tag add the parameter in a containing 'import' tag's URL. It 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.
- Let's see the simple example of tag:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Core Tag Example</title>
</head>
<body>
<c:url value="/index1.jsp" var="completeURL"/>
```

PRESIDENCY UNIVERSITY
Private University Estd. in Karnataka State by Act No. 41 of 2013
40 YEARS OF ACADEMIC WISDOM

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.
- Let's see the simple example of < c:redirect > tag:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Core Tag Example</title>
</head>
<body>
  <c:set var="url" value="0" scope="request"/>
    <c:if test="${url<1}">
      <c:redirect url="http://javatpoint.com"/>
```

PRESIDENCY UNIVERSITY
Private University Estd. in Karnataka State by Act No. 41 of 2013

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. The advantage of url tag is proper URL encoding and including the parameters specified by children. **param** tag.

Let's see the simple example of < c:url > tag:

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

PRESIDENCY UNIVERSITY
Private University Estd. in Karnataka State by Act No. 41 of 2013

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 Functions	Description
fn:contains()	It is used to test if an input string containing the specified substring in a program.
fn:containsIgnoreCase()	It is used to test if an input string contains the specified substring as a case insensitive way.
fn:endsWith()	It is used to test if an input string ends with the specified suffix.
fn:escapeXml()	It escapes the characters that would be interpreted as XML

JSTL fn:contains() Function

- The fn:contains() is used for testing if the string containing the specified substring. If the specified substring is found in the string, it returns true otherwise false.
- **The syntax used for including the fn:contains() function is:**
- boolean contains(java.lang.String, java.lang.String)
- Let's see the simple example to understand the functionality of fn:contains() function:

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

<html>
<head>
<title>Using JSTL Functions</title>
```

JSTL fn:containsIgnoreCase() Function

- The fn:containsIgnoreCase() 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
- **The syntax used for including the fn:containsIgnoreCase() function is:**
- boolean containsIgnoreCase(java.lang.String, java.lang.String)
- Let's see the simple example to understand the functionality of fn:containsIgnoreCase() function:

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

PRESIDENCY UNIVERSITY
Private University Estd. in Karnataka State by Act No. 41 of 2013
40 YEARS OF ACADEMIC WISDOM

JSTL fn:endsWith() Function

- The fn:endsWith() 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.
- **The syntax used for including the fn:endsWith() function is:**
- boolean endsWith(java.lang.String, java.lang.String)
- Let's see the simple example to understand the functionality of fn:endsWith() function:

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

<html>
<head>
<title>Using JSTL Functions</title>
```

PRESIDENCY UNIVERSITY
Private University Estd. in Karnataka State by Act No. 41 of 2013
40 YEARS OF ACADEMIC WISDOM

JSTL fn:escapeXml() Function

- The fn:escapeXml() function escapes the characters that would be interpreted as XML markup. It is used for escaping the character in XML markup language.
- **The syntax used for including the fn:escapeXml() function is:**
- java.lang.String escapeXml(java.lang.String)
- Let's see the simple example to understand the functionality of fn:escapeXml() function:

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

<html>
<head>
<title>Using JSTL Functions</title>
```

PRESIDENCY UNIVERSITY
Private University Estd. in Karnataka State by Act No. 41 of 2013
40 YEARS OF ACADEMIC WISDOM

JSTL fn:indexOf() Function

- The fn:indexOf() function return an index of string. It is used for determining the index of string specified in substring.
- **The syntax used for including the fn:indexOf() function is:**
- int indexOf(java.lang.String, java.lang.String)
- Let's see the simple example to understand the functionality of fn:indexOf() function:

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

<html>
<head>
<title>Using JSTL Functions</title>
</head>
```

PRESIDENCY UNIVERSITY
Private University Estd. in Karnataka State by Act No. 41 of 2013

JSTL fn:trim() Function

- The fn:trim() 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.
- **The syntax used for including the fn:trim() function is:**
- java.lang.String trim(java.lang.String)
- Let's see the simple example to understand the functionality of fn:trim() function:

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

<html>
<head>
<title>Using JSTL Functions</title>
```

JSTL fn:startsWith() Function

- The fn:startsWith() function test if an input string is started with the specified substring.
- **The syntax used for including the fn:startsWith() function is:**
- boolean fn:startsWith(String input, String prefix)
- 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.
- Let's see the simple example to understand the functionality of fn:startsWith() function:

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

JSTL fn:split() Function

- The fn:split() function splits the string into an array of substrings. It is used for splitting the string into an array based on a delimiter string.
- **The syntax used for including the fn:split() function is:**
- java.lang.String[] split(java.lang.String, java.lang.String)
- Let's see the simple example to understand the functionality of fn:split() function:
- **<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>**
- **<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>**

<html>
<head>

JSTL fn:toLowerCase() Function

- 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.
- **The syntax used for including the fn:toLowerCase() function is:**
- String fn:toLowerCase(String input)
- Let's see the simple example to understand the functionality of fn:toLowerCase() function:

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

<html>
<head>
<title> Using JSTL Function </title>
```

JSTL fn:toUpperCase() Function

- The fn:toUpperCase() 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.
- **The syntax used for including the fn:toUpperCase() function is:**
- String fn:toUpperCase(String input)
- It returns the string value after converting the input string to uppercase.
- Let's see the simple example to understand the functionality of fn:toUpperCase() function:

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

JSTL fn:substring() Function

- The fn:substring() 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.
- **The syntax used for including the fn:substring() function is:**
- String fn:substring(String inputstring, int start, int end)
- start: It is starting position of substring
- end: It is end position of substring
- inputstring: It is string from which a substring needed to be taken
- Return type of the function: String
- Let's see the simple example to understand the functionality of fn:substring() function:

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

JSTL fn:substringAfter() Function

- The fn:substringAfter() function returns the subset of string followed by a specific substring. It returns the part of string which lies after the provided string value.
- **The syntax used for including the fn:substringAfter() function is:**
- String fn:substringAfter(String input, String afterstring)
- Let's see the simple example to understand the functionality of fn:substringAfter() function:

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

<html>
<head>
<title>Using JSTL Function </title>
</head>
```

JSTL fn:substringBefore() Function

- The fn:substringBefore() 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.
- **The syntax used for including the fn:substringBefore() function is:**
- String fn:substringBefore(String input, String beforestring)
- Let's see the simple example to understand the functionality of fn:substringBefore() function:

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

<html>
<head>
<title>Using JSTL Function </title>
</head>
<body>
```

JSTL fn:length() Function

- The fn:length() 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.
- **The syntax used for including the fn:length() function is:**
- int length(java.lang.Object)
- It returns the length of object. The return type of this function is **int** .
- Let's see the simple example to understand the functionality of fn:length() function:

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

<html>
<head>
```

JSTL fn:replace() Function

- The fn:replace() 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.
- **The syntax used for including the fn:replace() function is:**
- String fn:replace(String input, String search_for, String replace_with)
- It searches the search_for string in the input and replaces it with replace_with string. In function three strings argument is used whose return type is also string.

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

<html>
<head>
<title>Using JSTL Function </title>
```

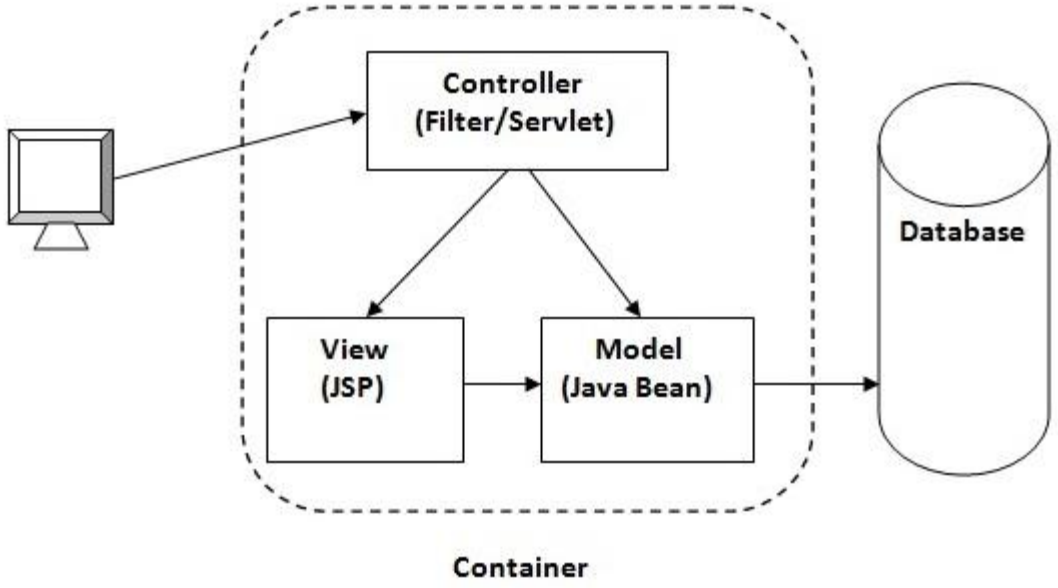
PRESIDENCY UNIVERSITY
Private University Estd. in Karnataka State by Act No. 41 of 2013
40 YEARS OF ACADEMIC WISDOM

MVC

- **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

1. Navigation Control is centralized
2. Easy to maintain the large application

(Model 2) Architecture



MVC Example

- 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

```
package com.javatpoint;
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, HttpServletResp
   onse response)
            throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out=response.getWriter();
```

File: LoginBean.java

```
package com.javatpoint;
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;
}
```

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" %>
```

File: web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  instance"
xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.su
  n.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.su
  n.com/xml/ns/javaee/web-app_3_0.xsd"
id="WebApp_ID" version="3.0">

  <servlet>
  <servlet-name>s1</servlet-name>
  <servlet-class>com.javatpoint.ControllerServlet</servlet-class>
  </servlet>
  <servlet-mapping>
  <servlet-name>s1</servlet-name>
```

MVC app with JDBC

- To implement a web application based on the MVC design pattern, we'll create an **Employee Registration** module using **JSP**, **Servlet**, **JDBC**, and **MySQL** database.
- EmployeeServlet class will act as a **Controller**, and for the presentation layer, we'll create employees.jsp page.

MySQL Database Setup

- Let's create a database named "mysql_database" in MySQL. Let's create an *employee* table using below DDL script:
- CREATE TABLE `employee` (`id` int(3) NOT NULL, `first_name` varchar(20) DEFAULT NULL, `last_name` varchar(20) DEFAULT NULL, `username` varchar(250) DEFAULT NULL, `password` varchar(20) DEFAULT NULL, `address` varchar(45) DEFAULT NULL, `contact` varchar(45) DEFAULT NULL, PRIMARY KEY (`id`)) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
- SELECT * FROM mysql_database.employee;

Model Layer

- Let's create the Employee class which will act as our **Model** class.
- **Employee**

```
package net.javaguides.jsp.jdbc.bean;
 import java.io.Serializable;
public class Employee implements Serializable {
      /** * */
     private static final long serialVersionUID = 1 L;
    private String firstName;
    private String lastName;
    private String username;
    private String password;
    private String address;
    private String contact;
```

DAO Layer

- **EmployeeDao.java**
- Let's create *EmployeeDao* class that contains JDBC code to connect with the MySQL database.
- Add the following code to an *EmployeeDao* class:

```
package net.javaguides.jsp.jdbc.database;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

import net.javaguides.jsp.jdbc.bean.Employee;

public class EmployeeDao {
```

Controller Layer

- Let's create an *EmployeeServlet* class to process HTTP request parameters and redirect to the appropriate JSP page after request data stored in the database:

```
package net.javaguides.employeemanagement.web;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import  net.javaguides.employeemanagement.dao.EmployeeDao;
import net.javaguides.employeemanagement.model.Employee;
@WebServlet("/register")
public class EmployeeServlet extends HttpServlet {    private static final
```

View Layer

- The unbounded wildcard type represents the list of an unknown type such as List<?>. This approach can be useful in the following scenarios: -
- When the given method is implemented by using the functionality provided in the Object class.
- When the generic class contains the methods that don't depend on the type parameter.
- **employeeregister.jsp**
- Let's design an employee registration HTML form with the following fields:
- firstName
- lastName
- username

View

```
<%@ page language="java" contentType="text/html; charset=ISO-
  8859-1" pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
 <head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
 </head>
 <body>
 <div align="center">
  <h1>Employee Register Form</h1>
<form action="<%= request.getContextPath() %>/register"
```

PRESIDENCY UNIVERSITY
Private University Estd. in Karnataka State by Act No. 41 of 2013

employeedetails.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-
  8859-1" pageEncoding="ISO-8859-1"%>
 <%@page import="net.javaguides.employeemanagement.dao.*"%>
  <!DOCTYPE html>
 <html>
<head>
 <meta charset="ISO-8859-1">
 <title>Insert title here</title>
 </head>
<body> <h1>User successfully registered!</h1>
 </body>
</html>
```

Reference

- https://www.javatpoint.com/MVC-in-jsp
- https://www.javaguides.net/2019/03/registration-form-using-jsp-servlet-jdbc-mysql-example.html