UNIT-2

SERVER SIDE PROGRAMMING

Servlet technology is used to create web application. Servlet technology is robust and scalable because of java language. Before servlet, CGI (Common Gateway Interface) scripting language was popular as a server-side programming language. But there was many disadvantages of this technology.

There are many interfaces and classes in the servlet API such as Servlet, GenericServlet, HttpServlet, ServletRequest, ServletResponse etc.

2.1 SERVLET

Servlet can be described in many ways, depending on the context.

- Servlet is a technology i.e. used to create web application.
- Servlet is an API that provides many interfaces and classes including documentations.
- 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 request. It can respond to any type of requests.
- Servlet is a web component that is deployed on the server to create dynamic web page.

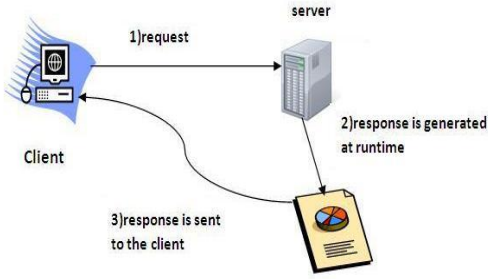


Fig 2.1 Servlet

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 components such as HTML. The web components typically execute in Web Server and respond to HTTP request.

CGI (Common Gateway Interface)

CGI technology enables the web server to call an external program and pass HTTP request information to the external program to process the request. For each request, it starts a new process.

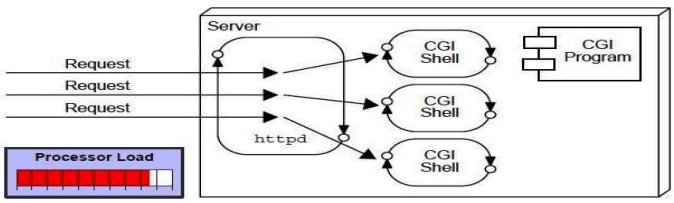


Fig 2.2 Common Gateway Interface

Disadvantages of CGI

There are many problems in CGI technology:

1. If number of client's increases, it takes more time for sending response.
2. For each request, it starts a process and Web server is limited to start processes.
3. It uses platform dependent language e.g. C, C++, Perl.

Advantage of Servlet

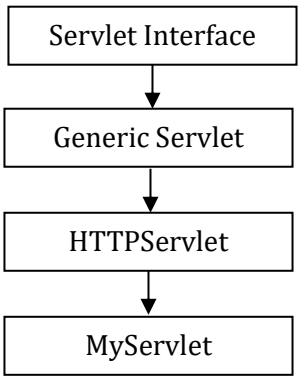
There are many advantages of servlet over CGI. The web container creates threads for handling the multiple requests to the servlet. Threads have a lot of benefits over the Processes such as they share a common memory area, lightweight, cost of communication between the threads are low.

The basic benefits of servlet are as follows:

- **Better Performance:** because it creates a thread for each request not process.
- **Portability:** because it uses java language.
- **Robust:** Servlets are managed by JVM so no need to worry about memory leak, garbage collection etc.
- **Secure:** because it uses java language.

2.2 ARCHITECTURE OF SERVLET

A Servlet is a class, which implements the javax.servlet.Servlet interface. However instead of directly implementing the javax.servlet.Servlet interface we extend a class that has implemented the interface like javax.servlet.GenericServletor javax.servlet.http.HttpServlet.



Servlet Execution

This is the process of a servlet execution takes place when client (browser) makes a request to the webserver.

Fig 2.3 Servlet Execution

Servlet architecture includes:

a) Servlet Interface

To write a servlet we need to implement Servlet interface. Servlet interface can be implemented directly or indirectly by extending **GenericServlet** or **HttpServlet** class.

b) Request handling methods
There are 3 methods defined in Servlet interface: **init(), service() and destroy()**.
The first time a servlet is invoked, the init method is called. It is called only once during the lifetime of a servlet.
The Service method is used for handling the client request. As the client request reaches to the container it creates a thread of the servlet object, and request and response object are also created. These request and response object are then passed as parameter to the service method, which then process the client request. The service method in turn calls the doGet or doPost methods (if the user has extended the class from HttpServlet).

c) Number of instances

Basic Structure of a Servlet

```
Public class firstServlet extends HttpServlet
{
  publicvoidinit()
  {
            /* Put your initialization code in this method,
              * as this method is called only once */
   }
  publicvoid service()
  {
            // Service request for Servlet
   }
  publicvoid destroy()
  {
        // For taking the servlet out of service, this method is called only once
  }
}
```

2.3 Life Cycle of Servlet

The javax.servlet.Servlet interface defines the life cycle methods of servlet such as init(), service() and destroy(). The Web container invokes the init(), service() and destroy() methods of a servlet during its life cycle. The sequence in which the Web container calls the life cycle methods of a servlet is:

1. The Web container loads the servlet class and creates one or more instances of the servlet class.
2. The Web container invokes init() method of the servlet instance during initialization of the servlet. The init() method is invoked only once in the servlet life cycle.
3. The Web container invokes the service() method to allow a servlet to process a client request.
4. The service() method processes the request and returns the response back to the Web container.
5. The servlet then waits to receive and process subsequent requests as explained in steps 3 and 4.
6. The Web container calls the destroy() method before removing the servlet instance from the service. The destroy() method is also invoked only once in a servlet life cycle.

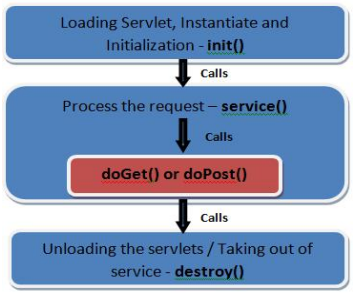


Fig 2.4 Web Container

The init() Method

- The init() method is called during initialization phase of the servlet life cycle. The Web container first maps the requested URL to the corresponding servlet available in the Web container and then instantiates the servlet.
- The Web container then creates an object of the ServletConfig interface, which contains the startup configuration information, such as initialization parameters of the servlet. The Web container then calls the init() method of the servlet and passes the ServletConfig object to it.

- The init() method throws a ServletException if the Web container cannot initialize the servlet resources. The servlet initialization completes before any client requests are accepted. The following code snippet shows the init() method:

```
Public class ServletLifeCycle extends HttpServlet
{
      static int count;
      public void init (ServletConfig config) throws ServletException
      {
            count=0;
      }
}
```

The service() Method

The service() method processes the client requests. Each time the Web container receives a client request, it invokes the service() method. The service() method is invoked only after the initialization of the servlet is complete. When the Web container calls the service() method, it passes an object of the ServletRequest interface and an object of the ServletResponse interface. The ServletRequest object contains information about the service request made by the client. The ServletResponse object contains the information returned by the servlet to the client. The following code snippet shows the service() method:

```
public void service(ServletRequest req, ServletResponse res) throws
ServletException, IOException
```

The service() method throws ServletException exception when an exception occurs that interferes with the normal operation of the servlet. The service() method throws IOException when an input or output exception occurs.

The service() method dispatches a client request to one of the request handler methods of the HttpServlet interface, such as the doGet(), doPost(), doHead() or

doPut(). The request handler methods accept the objects of the HttpServletRequest and HttpServletResponse as parameters from the service() method.

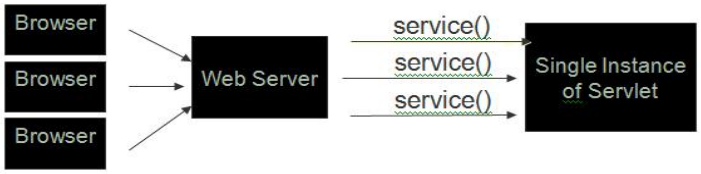


Fig 2.5 Life Cycle of Servlet

The destroy() Method

The destroy() method marks the end of the life cycle of a servlet. The Web container calls the destroy() method before removing a servlet instance from the service.

The Web container calls the destroy() method as

- The time period specified for the servlet has elapsed. The time period of a servlet is the period for which the servlet is kept in the active state by the Web container to service the client request.
- The Web container needs to release servlet instances to conserve memory.
- The Web container is about to shut down.

In the destroy() method we can write the code to release the resources occupied by the servlet. The destroy() method is also used to save any persistent information before the servlet instance is removed from the service.

The following code snippet shows the destroy() method:

```
public void destroy();
```

2.4 GENERIC SERVLET AND HTTP SERVLET

Generic Servlet:

GenericServlet class defines a protocol-independent (HTTP-less) servlet. However while building a website or an online application, you may want to have Http protocol, in that case extend HttpServlet instead of GenericServlet class.

The blue print of GenericServlet class:

```
public abstract class GenericServlet extends java.lang.Object implements Servlet,
ServletConfig,java.io.Serializable
```

An abstract class and it extends Object class & implements Servlet, ServletConfig and Serializable interfaces.

Methods of Generic Servlet class

- **void destroy()**: It is called by servlet container to indicate that the servlet life cycle is finished.
- **java.lang.String getInitParameter (java.lang.String name)**: It returns the value of the specified initialization parameter. If the parameter does not exist then it returns null.
- **java.util.Enumeration getInitParameterNames()**: This method returns all the initialization parameters in form of a String enumeration. By using enumeration methods we can fetch the individual parameter names.
- **ServletConfig getServletConfig()**: It returns this servlet's ServletConfig object.
- **ServletContext getServletContext()**: It returns a reference to the ServletContext in which this servlet is running.

- **java.lang.StringgetServletInfo()**: It returns information about the servlet, such as author, version, and copyright.
- **java.lang.StringgetServletName()**: This method returns the name of this servlet instance.
- **void init()**: This is the first method of servlet life cycle and it is used for initializing a servlet.
- **void init(ServletConfig config)**: Called by the servlet container to indicate to a servlet that the servlet is being placed into service.
- **void log (java.lang.Stringmsg)**: Writes the specified message to a servlet log file, prefixed with servlet's name.
- **void log (java.lang.String message, java.lang.Throwable t)**
- **abstract void service (ServletRequest req, ServletResponse res)**: It is called by the servlet container to allow the servlet to respond to the client's request.

Example of Generic Servlet
index.html

```
<a href="welcome"> Click to call Servlet </a>
```

ExampleGeneric.java

```
import java.io.*;
import javax.servlet.*;
public class ExampleGeneric extends GenericServlet
{
   public void service(ServletRequest req , ServletResponse res) throws IOException,
   ServletException
   {
        res.setContentType("text/html");
        PrintWriterpwriter=res.getWriter();
        pwriter.print("<html>");
        pwriter.print("<body>");
        pwriter.print("<h2> Generic Servlet Example </h2>");
        pwriter.print("</body>");
        pwriter.print("</html>");
   }
}
```

web.xml

```
<web-app>
    <servlet>
        <servlet-name> Beginners book </servlet-name>
        <servlet-class> /ExampleGeneric </servlet-class>
   </servlet>
    <servlet-mapping>
        <servlet-name> Beginners book </servlet-name>
        <url-pattern> welcome </url-pattern>
   </servlet-mapping>
</web-app>
```

HTTP Servlet:

The Http Servlet class implements Serializable interface and extends Generic Servlet class. It provides the definition of HTTP specific methods such as doGet(), doPost(), doTrace() etc.

Methods

The methods of HttpServlet class are as follows:

1. **protected void doGet (HttpServletRequest request, HttpServletResponse response)**: It handles the GET request. It is invoked by the web container.
2. **protected void doPost (HttpServletRequest request, HttpServletResponse response)**: Similar to doGet() method, it also gets invoked by the web container and it handles the POST request.
3. **public void service (ServletRequest request , ServletResponse response)**: There are two types of service method, one is public and other one is protected. The purpose of this public method is to dispatch the request to the protected service method by converting the request and response object into HTTP type.
4. **protected void service (HttpServletRequest request , HttpServletResponse response)**: This method receives the request from the public service() method. It dispatches the request to the doXXX() method depending on the incoming http request type.
5. **protected void doHead (HttpServletRequest request , HttpServletResponse response)**: It handles the HEAD request.
6. **protected void doOptions (HttpServletRequest request , HttpServletResponse response)**: Performs the HTTP OPTIONS operation; the default implementation of this method automatically determines what HTTP Options are supported.
7. **protected void doPut (HttpServletRequest request , HttpServletResponse response)**: Performs the HTTP PUT operation; the default implementation reports an HTTP BAD_REQUEST error.
8. **protected void doTrace (HttpServletRequest request , HttpServletResponse response)**: Performs the HTTP TRACE operation; the default implementation of this method causes a response with a message containing all of the headers sent in the trace request.
9. **protected void doDelete (HttpServletRequest request , HttpServletResponse response)**: Performs the HTTP DELETE operation; the default implementation reports an HTTP BAD_REQUEST error.
10. **protected long getLastModified (HttpServletRequest request)**: Gets the time the requested entity was last modified; the default implementation returns a negative number, indicating that the modification time is unknown and hence should not be used for conditional GET operations or for other cache control operations as this implementation will always return the contents.

2.5 COOKIES

Cookies are short pieces of data sent by web servers to the client browser. The cookies are saved to client's hard disk in the form of small text file. Cookies help the web servers to identify web users, by this way server tracks the user. Cookies pay very important role in the session tracking.

Cookie Class

- In JSP cookie are the objects of the class **javax.servlet.http.Cookie**. This class is used to create a cookie, a small amount of information sent by a servlet to a Web browser, saved by the browser, and later sent back to the

server. A cookie's value can uniquely identify a client, so cookies are commonly used for session management. 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.

- The getCookies() method of the request object returns an array of Cookie objects. Cookies can be constructed using the following code:

Cookie(java.lang.String name, java.lang.String value)

Cookie objects have the following methods.

Method	Description
getComment()	Returns the comment describing the purpose of this cookie, or null if no such comment has been defined.
getMaxAge()	Returns the maximum specified age of the cookie.
getName()	Returns the name of the cookie.
getPath()	Returns the prefix of all URLs for which this cookie is targeted.
getValue()	Returns the value of the cookie.
setComment(String)	If a web browser presents this cookie to a user, the cookie's purpose will be described using this comment.
setMaxAge(int)	Sets the maximum age of the cookie. The cookie will expire after that many seconds have passed. Negative values indicate the default behavior: the cookie is not stored persistently, and will be deleted when the user web browser exits. A zero value causes the cookie to be deleted
setPath(String)	This cookie should be presented only with requests beginning with this URL.
setValue(String)	Sets the value of the cookie. Values with various special characters (white space, brackets and parentheses, the equals sign, comma, double quote, slashes, question marks, the "at" sign, colon, and semicolon) should be avoided. Empty values may not behave the same way on all browsers.

Example Using Cookies

To write code in JSP file to set and then display the cookie.

Create Form

Here is the code of the form (cookieform.jsp) which prompts the user to enter his/her name.

```
<%@ page language="java" %>
<html>
        <head>
                <title> Cookie Input Form </title>
        </head>
        <body>
                <form method="post" action="setcookie.jsp">
                        <p> <b> Enter Your Name: </b> <input type="text" name="username"> <br>
```

```
                    <input type="submit" value="Submit">
                </form>
        </body>
</html>
```

Above form prompts the user to enter the user name. User input are posted to the setcookie.jsp file, which sets the cookie. Here is the code of **setcookie.jsp file:**

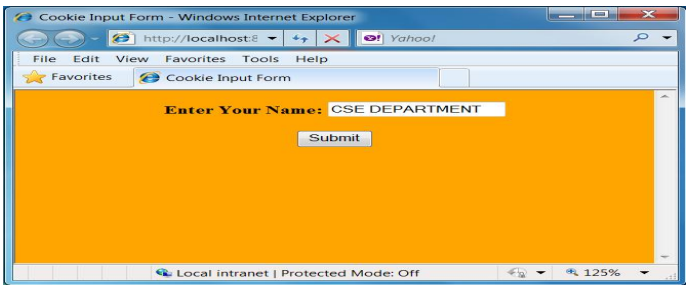
```
<%@ page language="java" import="java.util.*"%>
        <%
                String username=request.getParameter("username");
                if(username==null) username="";
                Date now = new Date();
                String timestamp = now.toString();
                Cookie cookie = new Cookie ("username",username);
                cookie.setMaxAge(365 * 24 * 60 * 60);
                response.addCookie(cookie);
        %>
<html>
        <head>
                <title> Cookie Saved </title>
        </head>
        <body>
                <p> <a href="showcookievalue.jsp"> Next Page to view the cookie value </a> <p>
        </body>
</html>
```

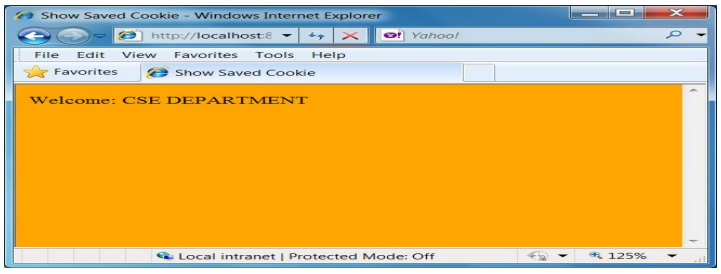
Above code sets the cookie and then displays a link to view cookie page. Here is the code of display cookie page **(showcookievalue.jsp):**

```
<%@ page language="java" %>
        <%
                String cookieName = "username";
                Cookie cookies [] = request.getCookies ();
                Cookie myCookie = null;
                if (cookies != null)
                {
                        for (inti = 0; i<cookies.length; i++)
                        {
                                if (cookies [i].getName().equals (cookieName))
                                {
                                        myCookie = cookies[i];
                                        break;
                                }
                        }
                }
        %>
<html>
        <head>
                <title>Show Saved Cookie</title>
        </head>
```

```
<body>
        <%
                if (myCookie == null)
                {
        %>
                No Cookie found with the name <%=cookieName%>
        <%
                }
                else
                {
        %>
                <p>Welcome: <%=myCookie.getValue()%>.
        <%
                }
        %>
</body> </html>
```

When user navigates to the above the page, cookie value is displayed.





2.6 FILTERS

A filter is an object that is invoked at the preprocessing and post processing of a request. It is mainly used to perform filtering tasks such as conversion, logging, compression, encryption and decryption, input validation etc. The servlet filter is pluggable, i.e. its entry is defined in the web.xml file, if we remove the entry of filter from the web.xml file, filter will be removed automatically and we don't need to change the servlet. So maintenance cost will be less.

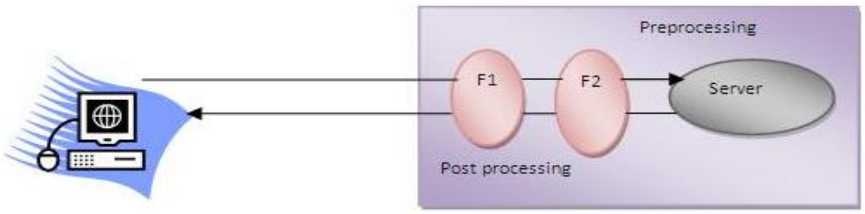


Fig 2.6 Filter

Usage of Filter

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

Advantage of Filter

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

Filter API

The servlet filter have its won API. The javax.servlet package contains the three interfaces of Filter API.

1. Filter
2. FilterChain
3. FilterConfig

1) Filter interface

For creating any filter, you must implement the Filter interface. Filter interface provides the life cycle methods for a filter.

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

2) FilterChain interface

The object of FilterChain is responsible to invoke the next filter or resource in the chain. This object is passed in the doFilter method of Filter interface. The FilterChain interface contains only one method:

public void doFilter(HttpServletRequest request, HttpServletResponse response): it passes the control to the next filter or resource.

How to define Filter

We can define filter same as servlet. Let's see the elements of filter and filter-mapping.

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

For mapping filter we can use, either url-pattern or servlet-name. The url-pattern elements has an advantage over servlet-name element i.e. it can be applied on servlet, JSP or HTML.

Example of Filter

We are simply displaying information that filter is invoked automatically after the post processing of the request.

index.html

```
<a href="servlet1"> click here </a>
```

MyFilter.java

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.*;
public class MyFilter implements Filter
{
public void init(FilterConfig arg0) throws ServletException {}
public void doFilter(ServletRequest req, ServletResponse resp,  FilterChain chain) th
rows IOException,
ServletException
{
        PrintWriter out=resp.getWriter();
        out.print("filter is invoked before");
        chain.doFilter(req, resp);              //sends request to next resource
         out.print("filter is invoked after");
 }
 public void destroy() {}
}
```

HelloServlet.java

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.*;
public class HelloServlet extends HttpServlet
```

```
{
public void doGet(HttpServletRequest request, HttpServletResponse response)throw
s ServletException,
IOException
{
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.print("<br> welcome to servlet <br>");
}
}
```

web.xml

For defining the filter, filter element of web-app must be defined just like servlet.

```
<web-app>
  <servlet>
        <servlet-name> s1 </servlet-name>
        <servlet-class> Hello Servlet </servlet-class>
  </servlet>
  <servlet-mapping>
        <servlet-name> s1 </servlet-name>
        <url-pattern>/servlet1</url-pattern>
  </servlet-mapping>
  <filter>
        <filter-name> f1 </filter-name>
        <filter-class> MyFilter </filter-class>
  </filter>
  <filter-mapping>
        <filter-name> f1 </filter-name>
        <url-pattern> /servlet1 </url-pattern>
        </filter-mapping>
</web-app>
```

2.7 JAVA SERVER PAGES (JSP)

- JSP abbreviated as Java Server Pages.
- JSP is Sun's solution for developing dynamic web sites.
- JSP provide excellent server side scripting support for creating database driven web applications.
- JSP is a server side technology that enables web programming to generate web pages dynamically in response to client requests.
- JSP uses HTML, XML and Java code, the application are secure, fast and independent of server platforms.
- JSP is nothing but high level abstraction of java servlet technology.
- It allows us to directly embed pure java code in an HTML page.
- JSP pages run under the supervision of an environment called web container.
- The web container compiles the page on the server and generates a servlet, which is loaded in the Java Runtime Environment (JRE).
- JSP enables the developers to directly insert java code into JSP file.
- JSP pages are efficient, they load into the web server's memory on receiving the request at the very first time and the subsequent calls are served within a very short period of time.

- This makes the development of the web applications convenient, simple and fast.
- Maintenance also becomes very easy.
- In today's environment, most website server's dynamic pages are based on user request.
- Database is very convenient way to store the data of users.
- JSP file have the extension .jsp

JSP and ASP

- JSP is portable or running the entire platform. Asp runs only in MS windows platform.
- Similar operation

JSP and JavaScript

JSP

- It dynamically Web Pages can be changed on the server.
- It can access server side resources like database, catalogs, pricing information.
- It can access session tracking and cookies.

JavaScript

- It can generate dynamically on the client side.
- It does not access this type of resources.
- It does not have.

JSP working Model:

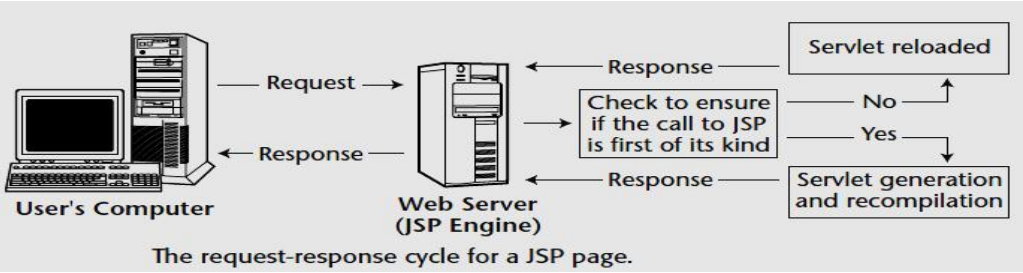


Fig 2.7 JSP Working Model

2.8 JSP Engines

- To Process JSP page, a JSP engine is needed.
- JSP Engine is nothing but a specialized servlet, which runs under the supervision of the servlet engine.
- The JSP Engine is typically connected with a web server or can be integrated inside a web server or an application server.

Many servers are

- Tomcat
- Java Web Server
- WebLogic
- WebSphere

Life Cycle of a Servlet and JSP

- A web server is responsible for initializing, invoking and destroying.
- A web server communicates with a servlet through a simple interfaces as
    1. JspInit()
    2. JspService()
    3. JspDestroy()

How JSP works?

1. Translates the JSP source code into the servlet source code.
2. Complies the servlet source code to generate a class file.
3. Loads the class file and create an instance.
4. Initializes the servlet instance by calling the jspInit() method.
5. Invokes the jspService() method, passing the request and response objects.

2.9 COMPONENTS OF JSP

JSP Syntax:

The syntax of JSP is almost similar to that of XML.

The general rules are

- Tags must have their matching end tags.
- Attributes must appear in the start tags.
- Attribute values in the tag must be quoted.

Components of JSP:

JSP page is built using components such as

- Directives
- Scripting Elements
    - Expressions
    - Declarations
    - Scriptlets
- Standard Actions
- Custom Tags

1. Directives

- A Directives element in a JSP page provides global information amount a particular JSP page.
- The syntax of jsp directive is

  <%@ directive-name attribute="value" %>

The three standard directives available in all JSP environments:

- Page
- include
- Taglib

Page Directives:

- The page directive defines attributes that notify the web container about the general settings of a JSP page.
- It has the following syntax:

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

Example:

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

Page Directives Examples

```
<%@ page language=”java” %>
<html>
        <head>
                <title> Simple JSP example </title>
        </head>
        <body>
                <h1> Welcome </h1> <hr>
                <h3>Today’s date and time: </h3>
        <%-- This is the JSP content that displays the server time by using the method Date() --%>
                <% =new java.util.Date() %>
        </body>
</html>
```

Output

Include Directives

- Include directives are too used to specify the name of the files to be inserted during the compilation of the JSP page.
- This directive is to include the HTML, JSP or Servlet file into a JSP file.
- This is a static inclusion to a file.

Syntax

<%@ include file="filename" %>

Example

```
<%@ include file="/header.html" %>
<%@ include file="/doc/legal/disclaimer.html" %>
<%@ include file="sortmethod" %>
```

Include Directives Examples

first.jsp

```
<html>
        <head>
                <title> An Include example </title>
        </head>
        <body>
                <h2> This is the content of first.jsp </h2> <hr>
                <% =”Hello, welcome to the world of Java Server Pages!!!” %>
                <%@ include file=”second.jsp” %>
        </body>
```

</html>

Second.jsp

```
<html>
        <head>
                <title> An Include example </title>
        </head>
        <body>
                <h2> This is the content of second. Jsp </h2> <hr>
                <%= “Hope you’re having fun learning JSP” %>
        </body>
</html>
```

Output

Comments

- The JSP comment syntax is
  <%- - This is a hidden JSP comment - - %>
- and the latter looks like this:
  <! - - This is included in the generated HTML - ->

2. Scripting Elements

The 3 different types of Scripting Elements are

1. Expressions
2. Declarations
3. Scriptlets

Expressions

Syntax of JSP Expression is

<% = expression %>

- JSP Expression start with <% = and ends with %>.
- Between these, any number of expressions can be embedded.
- The result of expression is converted to the string to get displayed.

Example

<%=“Hello World!” %>

Expression examples

```
<HTML>
        <HEAD>
```

```
        <TITLE> My first JSP Example </TITLE>
    </HEAD>

    <BODY>
        <H1> My first JSP example </H1> <HR>
        <%-- This is the JSP content --%>
        <%=”Hello World” %>
    </BODY>
</HTML>
```

Declarations:

The syntax of JSP Declaratives is

```
<%!
    // declare all the variables here
%>
```

- JSP declaration begins with <%! and ends with %>.
- We can embed any amount of java code in the JSP declarations.
- Variables and functions in the declarations are class level and can be used anywhere in the JSP page.

2.10 SCRIPLETS:

The Syntax of JSP Scriptlets

```
<%
    //java code
%>
```

Example:

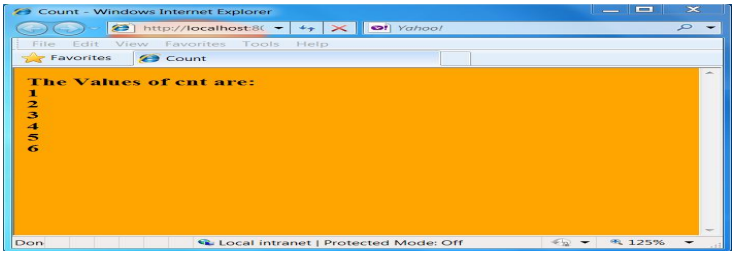
```
<%
    String username=cse;
    Username = request.getParameter(“username”);
%>
```

Program 1:

```
<%@ page language="java"%>
<html>
    <head>
        <title> Count program in JSP </title>
    </head>
    <body>
        <%!
            Int cnt=0;
            private int getcount()
            {
                // increment cnt and return the value
                cnt++;
                return cnt;
            }
        %>
        <p> The Values of cnt are:
```

```
                <%=getcount()%>
                <%=getcount()%>
                <%=getcount()%>
                <%=getcount()%>
        </body>
</html>
```

Output



- We can embed any amount of java code in the JSP Scriptlets.
- JSP Engine places these codes in JspService() methods.
- Variables available to the JSP Scriptlets are

  Request – HttpServletRequest
  Response – HttpServletResponse
  Session – HTTP session object associated with the request
  Out - is an object of output stream and is used to send any output to the client.

2.11 Standard Action:

- Actions are high-level JSP elements that create, modify, or use other objects.
- The syntax for the JSP standard Action is

  <tagname [attr="value" attr="value" ...] >
  ...
  </tag-name>

The various Tag name are

- Include
- Forward
- Param
- Plugin

Include:

- The Inclusion of file is dynamic and the specified file is included in the JSP file at runtime.
- Output of the included file is inserted into the JSP file.

Syntax:

<jsp: include page =“filename”>

Include Example

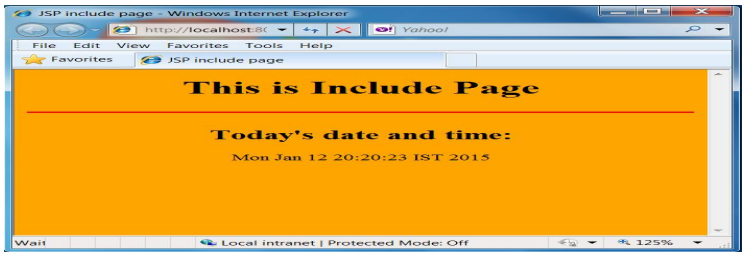
```
<%@page language="java"%>
<html>
```

```
<head>
        <title> JSP include page </title>
</head>
<body>
        <%-- using the <jsp:include> action --%>
        <h1 align="center"> This is Include Page </h1> <hr color="red">
        <jsp:include page="dateutil.jsp" flush="true"/>
</body>
</html>
```

dateutil.jsp

```
<%@ page language="java"%>
<html>
<head>
        <title> A Dynamic Content example </title>
</head>
<body bgcolor="orange">
    <center>
        <h2> Today's date and time: </h2>
        <%= new java.util.Date() %>
     </center>
</body>
</html>
```

Output:



Forward

This will redirect to the different page without notifying browser.

Syntax:

<jsp: forward page =“filename”/>

Param and Plugin:

<jsp:param>	Binds a value to a name and passes the binding to another resource invoked with <jsp:include> or <jsp:forward>. Syntax is <jsp:param name="*name*" value="*value*" />
<jsp:plugin>	Used to generate the appropriate HTML linkage for downloading the Java plugin: <jsp:plugin type="*bean | applet*" code="*objectCode*" codebase="*objectCodebase*" { align="*alignment*" } { archive="*archiveList*" } { height="*height*" } { hspace="*hspace*" } { jreversion="*jreversion*" } { name="*componentName*" } { vspace="*vspace*" } { width="*width*"} { nspluginurl="*url*" } { iepluginurl="*url*" } > { <jsp:params> { <jsp:param name="*name*" value="*value*" /> }+</jsp:params> }}</jsp:plugin>

2.12 CUSTOM TAGS

- The taglib directive makes custom actions available in the current page through the use of a tag library.

The syntax of the directive is

<%@ tagliburi="tagLibraryURI" prefix="tagPrefix" %>

Attribute	Value
tagLibraryURI	The URL of a Tag Library Descriptor.
tagPrefix	A unique prefix used to identify custom tags used later in the page.

Implicit Objects

Variable Name	Value
request	The ServletRequest or HttpServletRequest being serviced.
response	The ServletResponse or HttpServletResponse that will receive the generated HTML output.
pageContext	The PageContext object for this page. This object is a central repository for attribute data for the page, request, session, and application.
session	If the JSP page uses an HttpSession, it is available here under the name session.
application	The servlet context object.
Variable Name	**Value**
out	The character output stream used to generate the output HTML.
config	The ServletConfig object for this servlet context.
page	A reference to the JSP page itself.
exception	An uncaught exception that causes the error page to be invoked. This variable is available only to pages with isErrorPage="true".

2.13 SESSION TRACKING:

Session tracking is a mechanism that servlets use to maintain state about a series of requests from the same user (that is, requests originating from the same browser) across some period of time.

Session-Tracking basics

Every user of a site is associated with a javax.servlet.http.HttpSession object that servlets can use to store or retrieve information about that user
Basic session tracking functionalities,

- Obtain a session (an **HttpSession** object) for a user.
- Store or get data from the HttpSession object.
- Invalidate the session (optional).

Get Session

- A servlet uses its request object's getSession() method to retrieve the current HttpSession object.
- public HttpSessionHttpServletRequest.getSession(boolean create)
- This method returns the current session associated with the user making the request.
- If the user has no current valid session, this method creates one if create is true or returns null if create is false.
- To ensure the session is properly maintained, this method must be called at least once before any output is written to the response.

Put Value

- You can add data to an HttpSession object with the putValue() method:
- public void HttpSession.putValue(String name, Object value)
- This method binds the specified object value under the specified name. Any existing binding with the same name object from a session, use getValue():
- public Object HttpSession.getValue(String name)
- This method returns the object bound under the specified name or null if there is no binding.
- You can also get the names of all of the objects bound to a session with
- getValueNames():
- public String[] HttpSession.getValueNames()
- This method returns an array that contains the names of all objects bound to this session or an empty (zero length) array if there are no bindings.

Remove Value

- Finally, you can remove an object from a session with removeValue():
- public void HttpSession.removeValue(String name)
- This method removes the object bound to the specified name or does nothing if there is no binding
- Each of these methods can throw a java.lang.IllegalStateException if the session being accessed is invalid.

The Session Life Cycle

- Sessions do not last forever.
- A session either expires automatically, after a set time of inactivity (for the Java Web Server the default is 30 minutes), or manually, when it is explicitly invalidated by a servlet.
- When a session expires (or is invalidated), the HttpSession object and the data values it contains are removed from the system.
- Beware that any information saved in a user's session object is lost when the session is invalidated.

- If you need to retain information beyond that time, you should keep it in an external location (such as a database) and store a handle to the external data in the session object (or your own persistent cookie)

URL Rewriting

The Methods involved in managing the session life cycle

1. **Public boolean HttpSession.isNew() -** This method returns whether the session is new or not.
2. **public void HttpSession.invalidate() -**This method causes the session to be immediately invalidated. All objects stored in the session are unbound.
3. **public long HttpSession.getCreationTime() -** This method returns the time at which the session was created.
4. **public long HttpSession.getLastAccessedTime() -** This method returns the time at which the client last sent a request associated with this session.

2.14 DATABASE CONNECTIVITY

The connectivity from MYSQL database with JSP. We take an example of **Books** database. This database contains a table named **books_details**. This table contains three fields- **id**, **book_name**&**author**. We start from very beginning. First we learn how to create tables in MySQl database after that we write a html page for inserting the values in '**books_details**' table in database. After submitting values a table will be showed that contains the book name and author name.

Database

The database in example consists of a single table of three columns or fields. The database name is "books" and it contains information about **books names & authors**.

Table:books_details

ID	Book Name	Author
1.	Java I/O	Tim Ritchey
2.	Java & XML,2 Edition	Brett McLaughlin
3.	Java Swing, 2nd Edition	Dave Wood, Marc Loy

Start MYSQL prompt and type this SQL statement & press Enter-

MYSQL>CREATE DATABASE `books`;

This will create "books" database.

Now we create table a table "books_details" in database "books".

MYSQL>CREATE TABLE `books_details` (`id` INT(11) NOT NULL AUTO_INCREMENT , `book_name` VARCHAR(100) NOT NULL , `author` VARCHAR(100) NOT NULL , PRIMARY KEY (`id`)) TYPE = MYISAM ;

This will create a table "books_details" in database "books"

JSP Code

The following code contains html for user interface & the JSP back end-

```
<%@ page language="java" import="java.sql.*" %>
<%
        String driver = "org.gjt.mm.mysql.Driver";
```

```
        Class.forName(driver).newInstance();
        Connection con=null;
        ResultSetrst=null;
        Statement stmt=null;
        try   {
                String url="jdbc:mysql://localhost/books?user=
user>&password=<password>";
                con=DriverManager.getConnection(url);
                stmt=con.createStatement();
        }
        catch(Exception e) {
                System.out.println(e.getMessage());
        }
        if(request.getParameter("action") != null){
                String bookname=request.getParameter("bookname");
                String author=request.getParameter("author");
                stmt.executeUpdate("insert into books_details(book_name,author)
                      values('"+bookname+"','"+author+"')");
                rst=stmt.executeQuery("select * from books_details");
%>
                <html>
                <body>
                <center>
                        <h2> Books List </h2>
        <table border="1" cellspacing="0" cellpadding="0">
                        <tr>
                                <td> <b> S.No </b> </td>
                                <td> <b> Book Name </b> </td>
                                <td> <b> Author </b>  </td>
                        </tr>
        <%
                        int no=1;
                        while(rst.next()){
        %>

<tr>
                        <td> <%=no%> </td>
                        <td> <%=rst.getString("book_name")%> </td>
                <td> <%=rst.getString("author") </td>
                %>
                </tr>
                <%
                   no++;
        }
        rst.close();
        stmt.close();
        con.close();
            %>
        </table>
        </center>
```

```
        </body>
        </html>
<%}else{%>
        <html>
        <head>
                <title>Book Entry FormDocument</title>
                <script language="javascript">
                   function validate(objForm){
                        if(objForm.bookname.value.length==0){
                        alert("Please enter Book Name!");
                        objForm.bookname.focus();
                        return false;
                        }
                        if(objForm.author.value.length==0){
                        alert("Please enter Author name!");
                        objForm.author.focus();
                        return false;
                        }
                        return true;
                        }
                        </script>
                </head>
                <body>
                <center>
<form action="BookEntryForm.jsp" method="post" name="entry" onSubmit="return
validate(this)">
        <input type="hidden" value="list" name="action">
        <table border="1" cellpadding="0" cellspacing="0">
        <tr>
        <td>
        <table>
        <tr>
        <td colspan="2" align="center">
<h2> Book Entry Form </h2> </td>
        </tr>
        <tr>
        <td colspan="2"> </td>
        </tr>
        <tr>
        <td>Book Name:</td>
<td> <input name="bookname" type="text" size="50"> </td>
        </tr>
        <tr>
        <td> Author: </td> <td> <input name="author" type="text" size="50"> </td>
        </tr>
        <tr>
        <td colspan="2" align="center">
<input type="submit" value="Submit"> </td>
        </tr>
</table>
```

```
        </td>
</tr>
</table>
</form>
        </center>
        </body>
        </html>
<%}%>
```

Now we explain the above codes.

1. Declaring Variables:

Java is a strongly typed language which means, that variables must be explicitly declared before use and must be declared with the correct data types. In the above example code we declare some variables for making connection. These variables are-

Connection con=null;
ResultSetrst=null;
Statement stmt=null;

The objects of type Connection, ResultSet and Statement are associated with the Java sql. "con" is a Connection type object variable that will hold Connection type object. "rst" is a ResultSet type object variable that will hold a result set returned by a database query. "stmt" is a object variable of Statement .Statement Class methods allow to execute any query.

2. Connection to database:

The first task of this programmer is to load database driver. This is achieved using the single line of code:-

```
String driver = "org.gjt.mm.mysql.Driver";
Class.forName(driver).newInstance();
```

The next task is to make a connection. This is done using the single line of code :-

```
String
url="jdbc:mysql://localhost/books?user=<userName>&password=<password>
";
con=DriverManager.getConnection(url);
```

When url is passed into getConnection() method of DriverManager class it returns connection object.

Executing Query or Accessing data from database:

This is done using following code :-

```
stmt=con.createStatement(); //create a Statement object
rst=stmt.executeQuery("select * from books_details");
```

stmt is the Statement type variable name and **rst** is the RecordSet type variable. A query is always executed on a Statement object.

A Statement object is created by calling createStatement() method on connection object **con.**

The two most important methods of this Statement interface are executeQuery() and executeUpdate(). The executeQuery() method executes an SQL statement that returns a single ResultSet object. The executeUpdate() method executes an insert,

update, and delete SQL statement. The method returns the number of records affected by the SQL statement execution.
After creating a Statement, a method executeQuery() or executeUpdate() is called on Statement object **stmt** and a SQL query string is passed in method executeQuery() or executeUpdate().
This will return a ResultSet**rst**related to the query string.

Reading values from a ResultSet:

```
while(rst.next()){
 %>
<tr><td><%=no%></td><td><%=rst.getString("book_name")%></td><td><%=
rst.getString("author")%></td></tr>
  <%
}
```

The ResultSet represents a table-like database result set. A ResultSet object maintains a cursor pointing to its current row of data. Initially, the cursor is positioned before the first row. Therefore, to access the first row in the ResultSet, you use the next() method. This method moves the cursor to the next record and returns true if the next row is valid, and false if there are no more records in the ResultSet object.
Other important methods are getXXX() methods, where XXX is the data type returned by the method at the specified index, including String, long, and int. The indexing used is 1-based. For example, to obtain the second column of type String, you use the following code:

```
resultSet.getString(2);
```

You can also use the getXXX() methods that accept a column name instead of a column index. For instance, the following code retrieves the value of the column LastName of type String.

```
resultSet.getString("book_name");
```

The above example shows how you can use the next() method as well as the getString() method. Here you retrieve the 'book_name' and 'author' columns from a table called 'books_details'. You then iterate through the returned ResultSet and print all the book name and author name in the format " book name | author " to the web page.