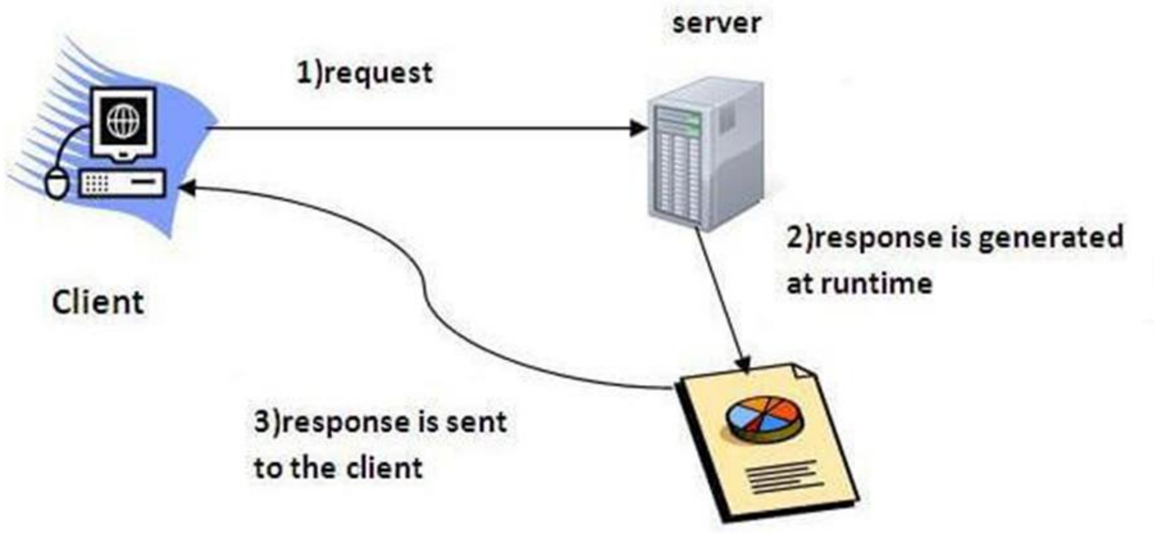
SERVLET

What is a Servlet?

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

- Servlet is a technology which is used to create a web application.
- 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.


server
1)request
2)response is generated at runtime
Client
3)response is sent to the client

Servlet technology is used to create a web application (resides at server side and generates a dynamic web page).

1. Consider a request for a static web page. A user enters a Uniform Resource Locator (URL) into a browser.
2. The browser generates an HTTP request to the appropriate web server. The web server maps this request to a specific file. That file is returned in an HTTP response to the browser.
3. The HTTP header in the response indicates the type of the content. The Multipurpose Internet Mail Extensions (MIME) are used for this purpose. For example, ordinary ASCII text has a MIME type of text/plain. The Hypertext Markup Language (HTML) source code of a web page has a MIME type of text/html.

Consider dynamic content. Assume that an online store uses a database to store information about its business. This would include items for sale, prices, availability, orders, and so forth. It wishes to make this information accessible to customers via web pages. The contents of those web pages must be dynamically generated to reflect the latest information in the database.

In the early days of the Web, a server could dynamically construct a page by creating a separate process to handle each client request. The process would open connections to one or more databases in order to obtain the necessary information. It communicated with the web server via an interface known as the Common Gateway Interface (CGI). CGI allowed the separate process to read data from the HTTP request and write data to the HTTP response. A variety of different languages were used to build CGI programs. These included C, C++, and Perl .However, CGI suffered serious performance problems. It was expensive in terms of processor and memory resources to create a separate process for each client request. It was also expensive to open and close database connections for each client request. In addition,the CGI programs were not platform-independent. Therefore, other techniques were introduced. Among these are servlets. Servlets offer several advantages in comparison with CGI.

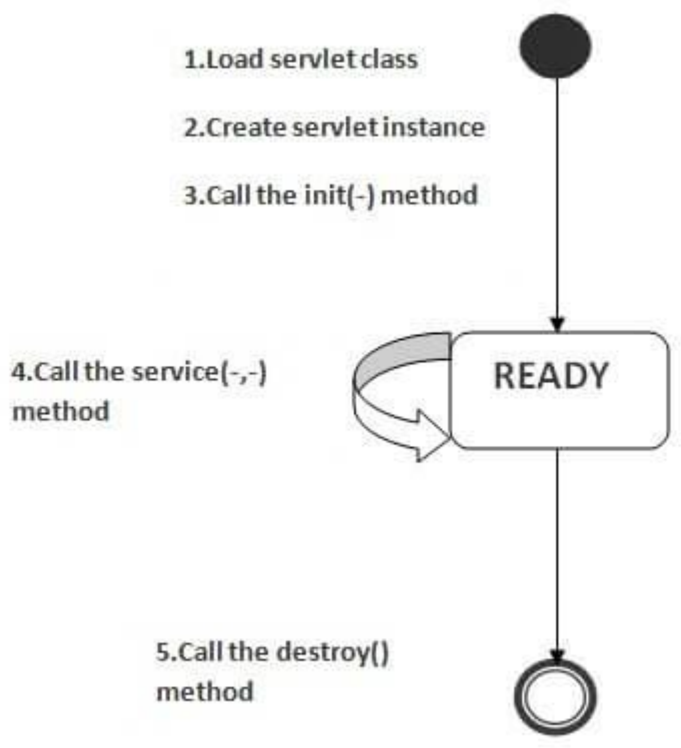
First, performance is significantly better. Servlets execute within the address space of a web server. It is not necessary to createa separate process to handle each client request.

Second, servlets are platform-independent because they are written in Java.

Third, the Java security manager on the server enforces a set of restrictions to protect the resources on a server machine.

Finally, the full functionality of the Java class libraries is available to a servlet. It can communicate with applets, databases, or other software via the sockets and RMI mechanisms that you have seen already.

1. Servlet class is loaded.
2. Servlet instance is created.
3. init method is invoked.
4. service method is invoked.
5. destroy method is invoked.


1.Load servlet class
2.Create servlet instance
3.Call the init(-) method
4.Call the service(-,-) method
READY
5.Call the destroy() method

Static website

Static website is the basic type of website that is easy to create. You don't need the knowledge of web programming and database design to create a static website. Its web pages are coded in HTML. The codes are fixed for each page so the information contained in the page does not change and it looks like a printed page.

Dynamic website

Dynamic website is a collection of dynamic web pages whose content changes dynamically. It accesses content from a database or Content Management System (CMS). Therefore, when you alter or update the content of the database, the content of the website is also altered or updated.Dynamic website uses client-side scripting or server-side scripting, or both to generate dynamic content.

Client side scripting generates content at the client computer on the basis of user input. The web browser downloads the web page from the server and processes the code within the page to render information to the user.In server side scripting, the software runs on the server and processing is completed in the server then plain pages are sent to the user.

The Life Cycle of a Servlet

Three methods are central to the life cycle of a servlet. These are **init( )**, **service( )**, and **destroy( )**.They are implemented by every servlet and are invoked at specific times by the server. Let us consider a typical user scenario to understand when these methods are called.

First, assume that a user enters a Uniform Resource Locator (URL) to a web browser. The browser then generates an HTTP request for this URL. This request is then sent to the appropriate server.

Second, this HTTP request is received by the web server. The server maps this request to a particular servlet. The servlet is dynamically retrieved and loaded into the address space of the server.

Third, the server invokes the **init( )** method of the servlet. This method is invoked only when the servlet is first loaded into memory. It is possible to pass initialization parameters to the servlet so it may configure itself.

Fourth, the server invokes the **service( )** method of the servlet. This method is called to process the HTTP request. You will see that it is possible for the servlet to read data that has been provided in the HTTP request. It may also formulate an HTTP response for the client. The servlet remains in the server's address space and is available to process any other HTTP requests received from clients. The **service( )** method is called for each HTTP request. Finally, the server may decide to unload the servlet from its memory. The algorithms by which this determination is made are specific to each server. The server calls the **destroy( )** method to relinquish any resources such as file handles that are allocated for the servlet. Important data may be saved to a persistent store. The memory allocated for the servlet and its objects can then be garbage collected.

The Servlet API

Two packages contain the classes and interfaces that are required to build servlets. These are

1. **javax.servlet**
2. **javax.servlet.http**

The javax.servlet Package

The **javax.servlet** package contains a number of interfaces and classes that establish the framework in which servlets operate.

Interface	Description
Servlet	Declares life cycle methods for a servlet.
ServletConfig	Allows servlets to get initialization parameters.
ServletContext	Enables servlets to log events and access information about their environment.
ServletRequest	Used to read data from a client request.
ServletResponse	Used to write data to a client response.

Class	Description
GenericServlet	Implements the Servlet and ServletConfig interfaces.
ServletInputStream	Provides an input stream for reading requests from a client.
ServletOutputStream	Provides an output stream for writing responses to a client.
ServletException	Indicates a servlet error occurred.
UnavailableException	Indicates a servlet is unavailable

javax.servlet.http Package

The **javax.servlet.http** package contains a number of interfaces and classes that are commonly used by servlet developers.

Interface	Description
HttpServletRequest	Enables servlets to read data from an HTTP request.
HttpServletResponse	Enables servlets to write data to an HTTP response.
HttpSession	Allows session data to be read and written.
HttpSessionBindingListener	Informs an object that it is bound to or unbound from a session.

The following table summarizes the core classes that are provided in this package. The most important of these is **HttpServlet**. Servlet developers typically extend this class in order to process HTTP requests.

Class	Description
Cookie	Allows state information to be stored on a client machine.
HttpServlet	Provides methods to handle HTTP requests and responses.
HttpSessionEvent	Encapsulates a session-changed event.

• HttpSessionBindingEvent	Indicates when a listener is bound to or unbound from a Session value, or that a session attribute changed.

Handling HTTP Requests and Responses

Handling HTTP GET Requests

The servlet is invoked when a form on a web page is submitted. The example contains two files. A web page is defined in **ColorGet.html**, and a servlet is defined in **ColorGetServlet.java**. It defines a form that contains a select element and a submit button.

The URL identifies a servlet to process the HTTP GET request.

```
<html>
<body>
<center>
<form name="Form1" action="http://localhost:8080/servlets
examples/servlet/ColorGetServlet">
<B>Color:</B>
<select name="color" size="1">

<option value="Red">Red</option>

<option value="Green">Green</option>
<option value="Blue">Blue</option>
</select>
<br><br>
<input type=submit value="Submit">
</form>
</body>
</html>
```

The **doGet( )** method is overridden to process any HTTP GET requests that are sent to this servlet. It uses the **getParameter( )** method of **HttpServletRequest** to obtain the selection that was made by the user.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ColorGetServlet extends HttpServlet {
public void doGet(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
String color = request.getParameter("color");
response.setContentType("text/html");
PrintWriter pw = response.getWriter();
pw.println("<B>The selected color is: ");
pw.println(color);
pw.close();
}
}
```

Handling HTTP POST Requests

The servlet is invoked when a form on a web page is submitted. A web page is defined in **ColorPost.html**, and a servlet is defined in the class **ColorPostServlet.java**. It is identical to **ColorGet.html** except that the method parameter for the form tag explicitly specifies that the POST method should be used, and the action parameter for the form tag specifies a different servlet.

```
<html>
<body>
<center>
<form name="Form1"method="post"
action="http://localhost:8080/servletsexamples/servlet/ColorPostServlet">
<B>Color:</B>
<select name="color" size="1">
<option value="Red">Red</option>
```

```
<option value="Green">Green</option>
<option value="Blue">Blue</option>
</select>
<br><br>
<input type=submit value="Submit">
</form>
</body>
</html>
```

The source code for **ColorPostServlet.java** is shown in the following listing. The **doPost( )** method is overridden to process any HTTP POST requests that are sent to this servlet. It uses the **getParameter( )** method of **HttpServletRequest** to obtain the selection that was made by the user. import java.io.*;

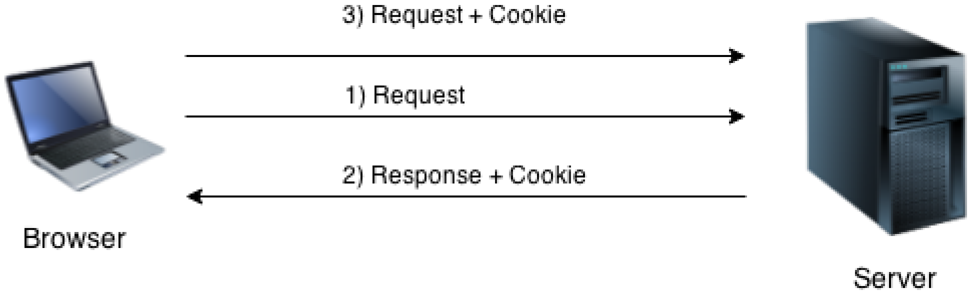
```
import javax.servlet.*;
import javax.servlet.http.*;
public class ColorPostServlet extends HttpServlet {
public void doPost(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
String color = request.getParameter("color");
response.setContentType("text/html");
PrintWriter pw = response.getWriter();
pw.println("<B>The selected color is: ");
pw.println(color);
pw.close();
}
}
```

Cookies in Servlet

A **cookie** is a small piece of information that is persisted between the multiple client requests. A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

How Cookie works

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



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

1. Simplest technique of maintaining the state.
2. Cookies are maintained at client side.

Disadvantage of Cookies

1. It will not work if cookie is disabled from the browser.
2. Only textual information can be set in Cookie object.

Using Cookies

The servlet is invoked when a form on a web page is submitted. The example contains three files as summarized here:

File	Description
• AddCookie.htm	Allows a user to specify a value for the cookie named MyCookie.
• AddCookieServlet.java	Processes the submission of AddCookie.htm.
• GetCookiesServlet.java	Displays cookie values.

The HTML source code for **AddCookie.html** is shown in the following listing. This page contains a text field in which a value can be entered. There is also a submit button on the page. When this button is pressed, the value in the text field is sent to **AddCookieServlet** via an HTTP POST request.

```
<html>
<body>
<center>
<form name="Form1"
method="post"
action="http://localhost:8080/servlets-examples/servlet/AddCookieServlet">
<B>Enter a value for MyCookie:</B>
<input type=textbox name="data" size=25 value="">
<input type=submit value="Submit">
</form>
</body>
</html>
```

The source code for **AddCookieServlet.java** is shown in the following listing. It gets the value of the parameter named “data”. It then creates a **Cookie** object that has the name “MyCookie” and contains the value of the “data” parameter. The cookie is then added to the header of the HTTP response via the **addCookie( )** method. A feedback message is then written to the browser.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class AddCookieServlet extends HttpServlet {
public void doPost(HttpServletRequest request,HttpServletResponse response) throws
ServletException, IOException {

// Get parameter from HTTP request.
String data = request.getParameter("data");
// Create cookie.
Cookie cookie = new Cookie("MyCookie", data);
// Add cookie to HTTP response.
response.addCookie(cookie);
// Write output to browser.
response.setContentType("text/html");
PrintWriter pw = response.getWriter();
pw.println("<B>MyCookie has been set to");
pw.println(data);
pw.close();
}
}
```

The source code for **GetCookiesServlet.java** is shown in the following listing. It invokes the **getCookies( )** method to read any cookies that are included in the HTTP GET request.

The names and values of these cookies are then written to the HTTP response. Observe that the **getName( )** and **getValue( )** methods are called to obtain this information.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class GetCookiesServlet extends HttpServlet {
public void doGet(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
```

```
// Get cookies from header of HTTP request.
Cookie[] cookies = request.getCookies();
// Display these cookies.
response.setContentType("text/html");
PrintWriter pw = response.getWriter();
pw.println("<B>");
for(int i = 0; i < cookies.length; i++) {
String name = cookies[i].getName();
String value = cookies[i].getValue();
pw.println("name = " + name +"; value = " + value);
}
pw.close();
}
}
```

Session Tracking

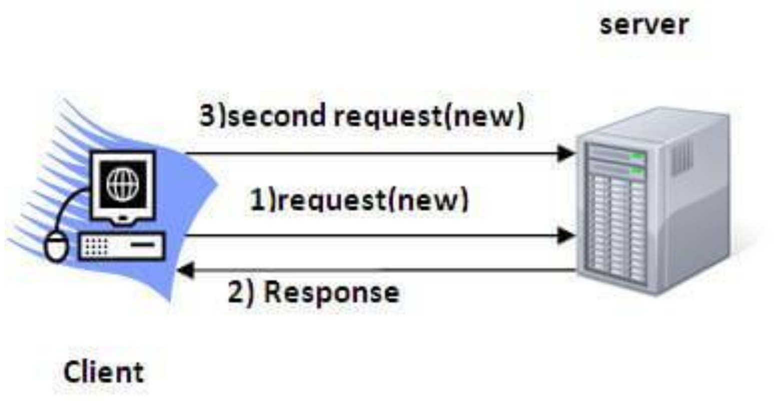
1. **Session Tracking**
2. **Session Tracking Techniques**

Session simply means a particular interval of time.

Session Tracking is a way to maintain state (data) of an user. It is also known as **session management** in servlet.

Http protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.

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



Why use Session Tracking?

It is used to recognize the particular user.

Session methods ,classes & Interfaces

A session can be created via the **getSession( )** method of **HttpServletRequest**. An **HttpSession** object is returned. This object can store a set of bindings that associate names with objects.

The **setAttribute( )**, **getAttribute( )**, **getAttributeNames( )**, and **removeAttribute( )** methods of **HttpSession** manage these bindings. It is important to note that session state is shared among all the servlets that are associated with a particular client.

The following servlet illustrates how to use session state with an program .

The **getSession( )** method gets the current session. A new session is created if one does not already exist.

The **getAttribute( )** method is called to obtain the object that is bound to the name “date”. The object is a **Date** object that encapsulates the date and time when this page was last accessed. (Note: Of course, there is no such binding when the page is first accessed).

A **Date** object encapsulating the current date and time is then created.

The **setAttribute( )** method is called to bind the name “date” to this object.

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

```
public void doGet(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {

// Get the HttpSession object.
HttpSession hs = request.getSession(true);
// Get writer.
response.setContentType("text/html");
PrintWriter pw = response.getWriter();
pw.print("<B>");
// Display date/time of last access.
Date date = (Date)hs.getAttribute("date");
if(date != null) {
pw.print("Last access: " + date + "<br>");
}
// Display current date/time.
date = new Date();
hs.setAttribute("date", date);
pw.println("Current date: " + date);
}
}
```

When you first request this servlet, the browser displays one line with the current date and time information. On subsequent invocations, two lines are displayed. The first line shows the date and time when the servlet was last accessed. The second line shows the current date and time

JSP

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

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

Advantages of JSP over Servlet

There are many advantages of JSP over the Servlet. They are as follows:

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

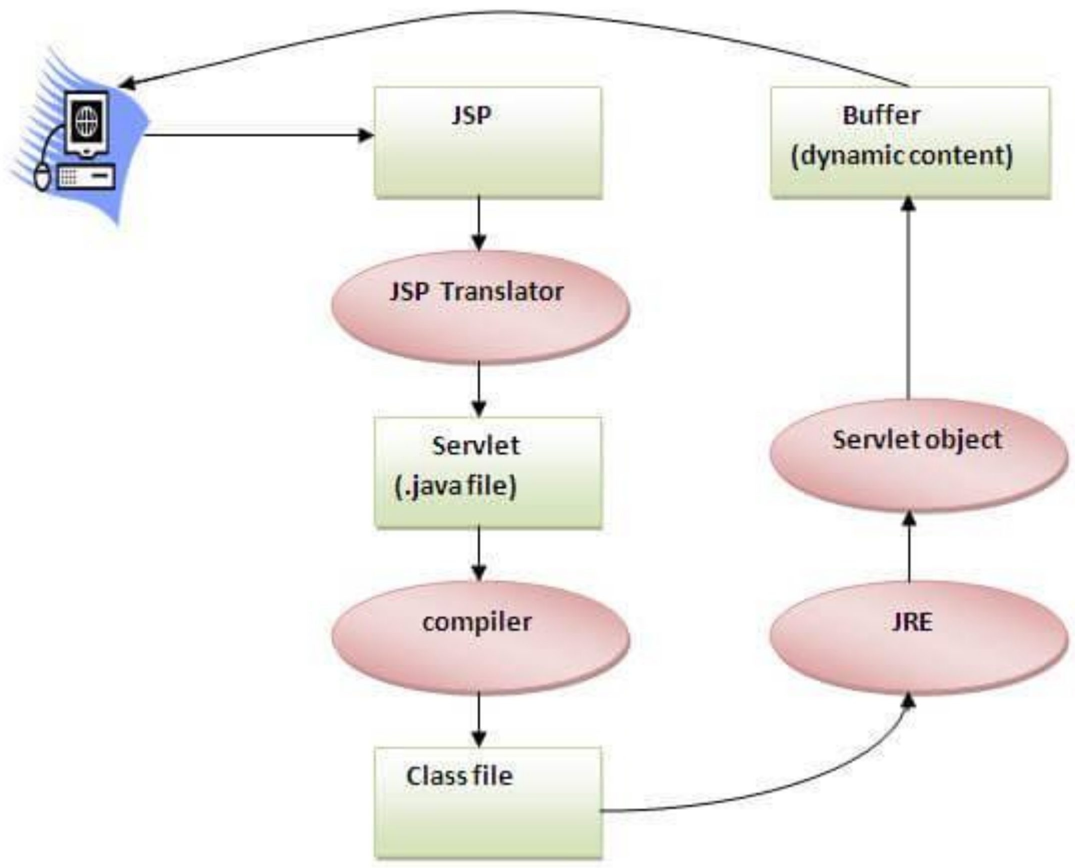
If JSP page is modified, we don't need to recompile and redeploy the project. The Servlet code needs to be updated and recompiled if we have to change the look and feel of the application.

4) Less code than Servlet

In JSP, we can use many tags such as action tags, JSTL, custom tags, etc. that reduces the code. Moreover, we can use EL, implicit objects, etc.

The Lifecycle of a JSP Page

- Translation of JSP Page
- Compilation of JSP Page
- Classloading (the classloader loads class file)
- Instantiation (Object of the Generated Servlet is created).
- Initialization (the container invokes jspInit() method).
- Request processing (the container invokes _jspService() method).
- Destroy (the container invokes jspDestroy() method).



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.

JSP Scriptlet tag (Scripting elements)

In JSP, java code can be written inside the jsp page using the scriptlet tag. Let's see what are the scripting elements first.

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 : <% java source code %>

Example of JSP scriptlet tag

In this example, we are displaying a welcome message.

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

Example of JSP scriptlet tag that prints the user name

In this example, we have created two files index.html and welcome.jsp. The index.html file gets the username from the user and the welcome.jsp file prints the username with the welcome message.

File: index.html

```
<html>
```

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

File: welcome.jsp

```
<html>
<body>
<%
String name=request.getParameter("uname");
out.print("welcome "+name);
%>
</form>
</body>
</html>
```

JSP expression tag

The code placed within **JSP expression tag** is *written to the output stream of the response*. So you need not write out.print() to write data. It is mainly used to print the values of variable or method.

Syntax :

```
<%=  statement %>
```

1. **Example of JSP expression tag, we are simply displaying a welcome message.**

```
<html>
<body>
<%= "welcome to jsp" %>
</body>
</html>
```

2. Example of JSP expression tag that prints current time

To display the current time, we have used the getTime() method of Calendar class. The getTime() is an instance method of Calendar class, so we have called it after getting the instance of Calendar class by the getInstance() method.

index.jsp

```
<html>
<body>
Current Time: <%= java.util.Calendar.getInstance().getTime() %>
</body>
</html>
```

3. Example of JSP expression tag that prints the user name

In this example, we are printing the username using the expression tag. The index.html file gets the username and sends the request to the welcome.jsp file, which displays the username.

File: index.jsp

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

File: welcome.jsp

```
<html>
<body>
<%= "Welcome "+request.getParameter("uname") %>
</body>
</html>
```

JSP Declaration Tag

The **JSP declaration tag** is used *to declare fields and methods*.The code written inside the jsp declaration tag is placed outside the service() method of auto generated servlet.So it doesn't get memory at each request.

Syntax:

```
<%!  field or method declaration %>
```

Difference between JSP Scriptlet tag and Declaration tag

Jsp Scriptlet Tag	**Jsp Declaration Tag**
The jsp scriptlet tag can only declare variables not methods.	The jsp declaration tag can declare variables as well as methods.
The declaration of scriptlet tag is placed inside the _jspService() method.	The declaration of jsp declaration tag is placed outside the _jspService() method.

Example of JSP declaration tag that declares field

In this example of JSP declaration tag, we are declaring the field and printing the value of the declared field using the jsp expression tag.

index.jsp

```
<html>
<body>
<%! int data=50; %>
<%= "Value of the variable is:"+data %>
</body>
</html>
```

Example of JSP declaration tag that declares method

In this example of JSP declaration tag, we are defining the method which returns the cube of given number and calling this method from the jsp expression tag. But we can also use jsp scriptlet tag to call the declared method.

index.jsp

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

```
int cube(int n){
return n*n*n*;
}
%>
<%= "Cube of 3 is:"+cube(3) %>
</body>
</html>
```