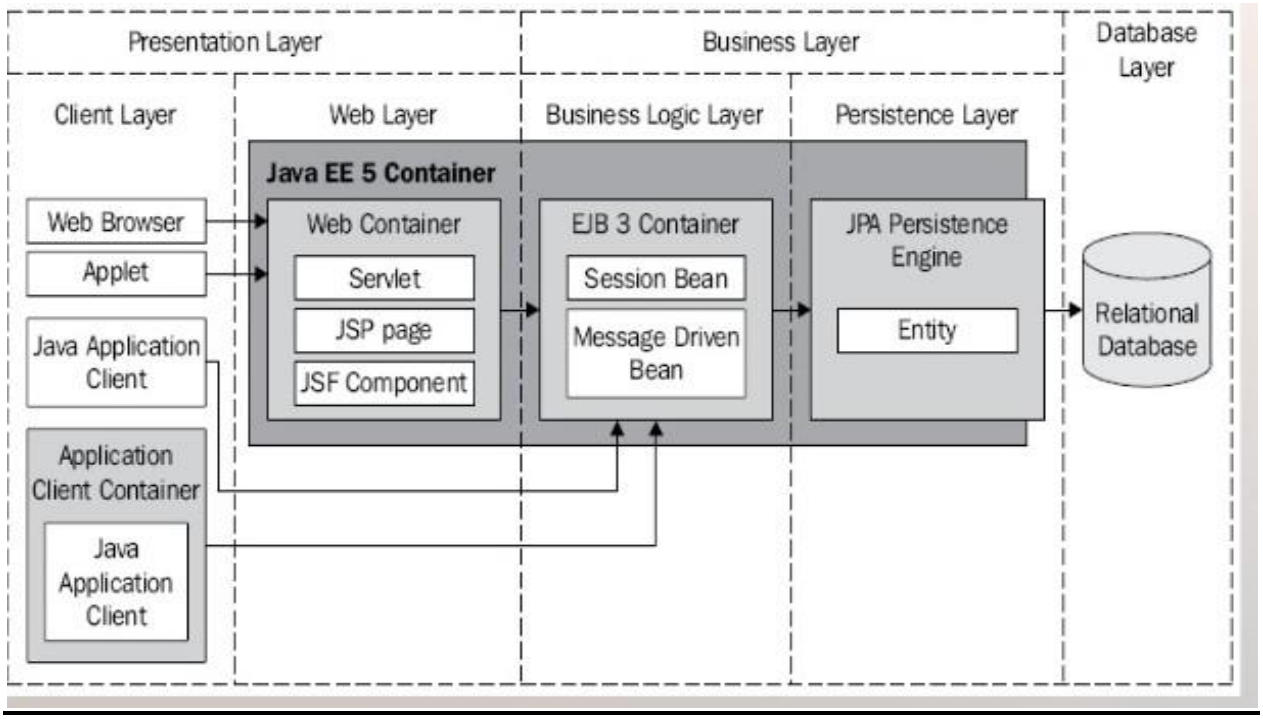
INTRODUCTION TO J2EE

J2EE Introduction

- Short for *Java 2 Platform Enterprise Edition.*
- J2EE is a platform-independent, Java-centric environment from Sun/Oracle for developing, building and deploying Web-based enterprise applications online.
- The J2EE platform consists of a set of services, APIs, and protocols that provide the functionality for developing multi-tiered, Web-based applications.
- The Java EE platform is built on top of the Java SE platform.
- The Java EE platform provides an API and runtime environment for developing and running
  - large-scale
  - multi-tiered
  - scalable
  - reliable
  - secure network applications

A glance at J2EE architecture



Distributed Multi-tiered Applications

- The Java EE platform uses a distributed Multi-tiered application model for enterprise applications.
- Application logic is divided into components according to function

The application components that make up a Java EE application are installed on various machines depending on the tier in the Multi-tiered Java EE environment to which the application component belongs.

Java EE Components

- Java EE applications are made up of components.
- A Java EE component is
  - a self-contained functional software unit that
  - is assembled into a Java EE application with its related classes and files
  - communicates with other components.
- The Java EE specification defines the following Java EE components:
  - Application clients and applets are components that run on the client.
  - Java Servlet, JavaServer Faces, and JavaServer Pages (JSP) technology components are web components that run on the server.
  - EJB components (enterprise beans) are business components that run on the server.
- Java EE components are written in the Java programming language and are compiled in the same way as any program in the language.
- The differences between Java EE components and "standard" Java classes:
  - Java EE components are assembled into a Java EE application
  - they are verified to be well formed and in compliance with the Java EE specification
  - they are deployed to production, where they are run and managed by the Java EE server.
- Client-tier components run on the client machine.
- Web-tier components run on the Java EE server.
- Business-tier components run on the Java EE server.
- Enterprise information system (EIS)-tier software runs on the EIS server.

Introduction to 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

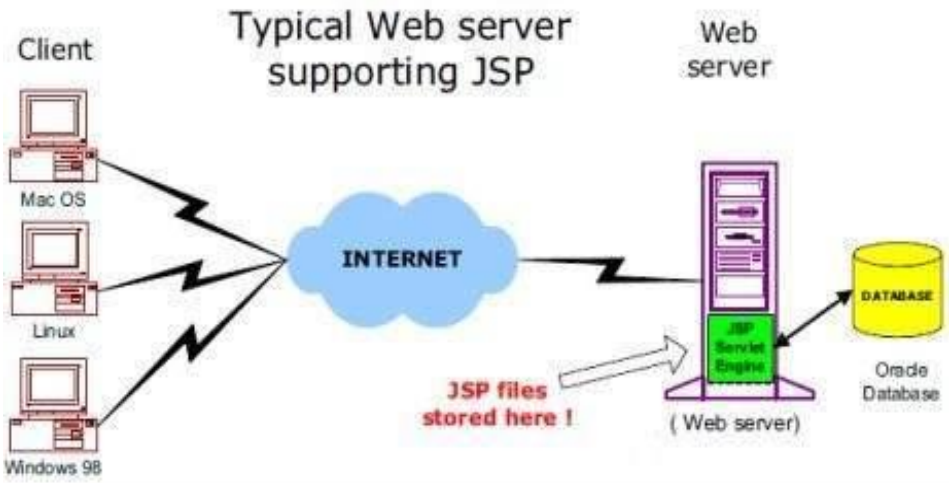
In JSP, we can use many tags such as action tags, JSTL, custom tags, etc. that reduces the code. Moreover, we can use EL, implicit objects, etc.

JSP - Architecture

The web server needs a JSP engine, i.e, a container to process JSP pages. The JSP container is responsible for intercepting requests for JSP pages. This tutorial makes use of Apache which has built-in JSP container to support JSP pages development.

A JSP container works with the Web server to provide the runtime environment and other services a JSP needs. It knows how to understand the special elements that are part of JSPs.

Following diagram shows the position of JSP container and JSP files in a Web application.

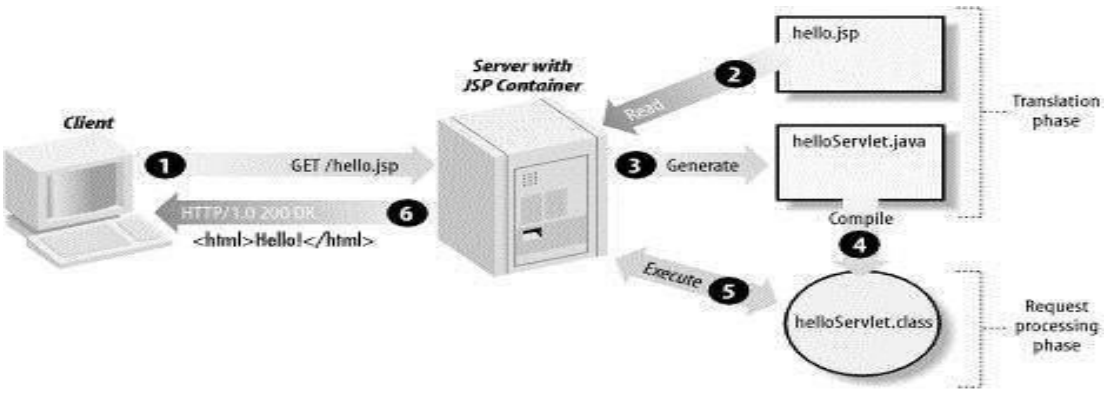


JSP Processing

The following steps explain how the web server creates the Webpage using JSP −

- As with a normal page, your browser sends an HTTP request to the web server.
- The web server recognizes that the HTTP request is for a JSP page and forwards it to a JSP engine. This is done by using the URL or JSP page which ends with **.jsp** instead of **.html**.
- The JSP engine loads the JSP page from disk and converts it into a servlet content. This conversion is very simple in which all template text is converted to println( ) statements and all JSP elements are converted to Java code. This code implements the corresponding dynamic behavior of the page.
- The JSP engine compiles the servlet into an executable class and forwards the original request to a servlet engine.
- A part of the web server called the servlet engine loads the Servlet class and executes it. During execution, the servlet produces an output in HTML format. The output is furthur passed on to the web server by the servlet engine inside an HTTP response.
- The web server forwards the HTTP response to your browser in terms of static HTML content.
- Finally, the web browser handles the dynamically-generated HTML page inside the HTTP response exactly as if it were a static page.

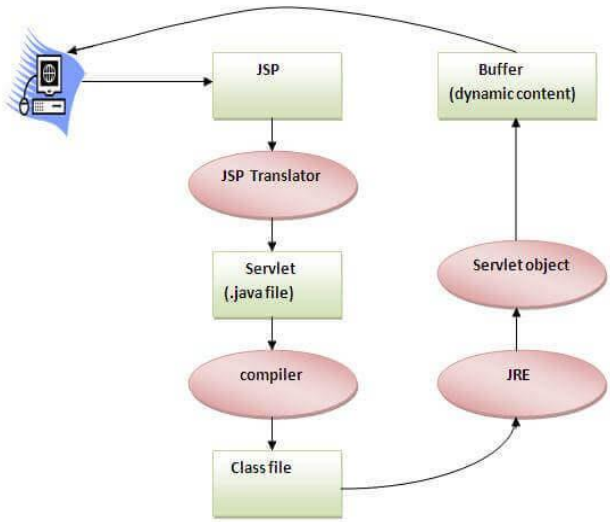
All the above mentioned steps can be seen in the following diagram −



The 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).



As depicted in the above diagram, 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.

EXAMPLE 1:

```
1. <html>
2. <body>
3. <% out.print(2*5); %>
4. </body>
5. </html>
```

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

Elements of JSP

The elements of JSP have been described below −

Scripting elements

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

```
1. <html> <body>
2. <% out.print("welcome to jsp"); %>
3. </body>
4. </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 %>**

Example

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.

```
1. <html>
2. <body>
3. Current Time: <%= java.util.Calendar.getInstance().getTime() %>
4. </body>
5. </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 :

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.

```
1. <html>
2. <body>
3. <%! int data=50; %>
```

4. <%= "Value of the variable is:"+data %>
5. **</body>**
6. **</html>**

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.

JSP Comments

JSP comment marks text or statements that the JSP container should ignore. A JSP comment is useful when you want to hide or "comment out", a part of your JSP page.

Following is the syntax of the JSP comments −

```
<%-- This is JSP comment --%>
```

Following example shows the JSP Comments −

```
<html>
  <head><title>A Comment Test</title></head>

  <body>
    <h2>A Test of Comments</h2>
    <%-- This comment will not be visible in the page source --%>
  </body>
</html>
```

JSP Directives

A JSP directive affects the overall structure of the servlet class. It usually has the following form −

```
<%@ directive attribute="value" %>
```

There are three types of directive tag −

S.No.	Directive & Description
1	**<%@ page ... %>** Defines page-dependent attributes, such as scripting language, error page, and buffering requirements.
2	**<%@ include ... %>** Includes a file during the translation phase.
3	**<%@ taglib ... %>** Declares a tag library, containing custom actions, used in the page

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
- session
- errorPage
- isErrorPage

import Attribute

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

```
1. <html>
2. <body>
3. <%@ page import="java.util.Date" %>
4. Today is: <%= new Date() %>
5. </body>
6. </html>
```

Jsp Include Directive

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

Advantage - 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.

```
1. <html>
2. <body>
3.   <%@ include file="header.html" %>
4.   Today is: <%= java.util.Calendar.getInstance().getTime() %>
5.   </body>
6. </html>
```

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.

```
1. <html>
2. <body>
3.   <%@ taglib uri="http://www.javatpoint.com/tags" prefix="mytag" %>
4.   <mytag:currentDate/>
5.   </body>
6. </html>
```

Servlet

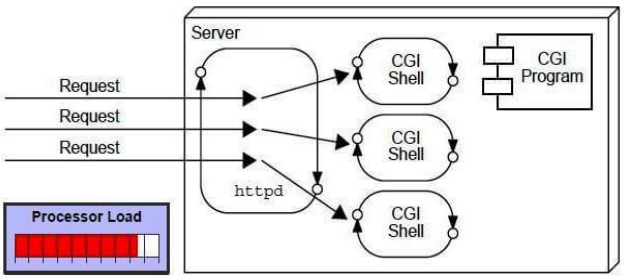
Servlet technology is used to create a web application (resides at server side and generates a dynamic web page). **Servlet** technology is robust and scalable because of java language. Before Servlet, CGI (Common Gateway Interface) scripting language was common as a server-side programming language. However, there were many disadvantages to this technology. There are many interfaces and classes in the Servlet API such as Servlet, GenericServlet, HttpServlet, ServletRequest, ServletResponse, etc.

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.

CGI (Common Gateway Interface)

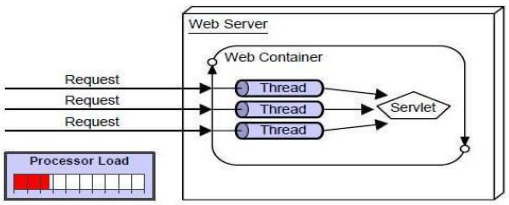
CGI technology enables the web server to call an external program and pass HTTP request information to the external program to process the request. For each request, it starts a new process.



Disadvantages of CGI :

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

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.

Website

Website is a collection of related web pages that may contain text, images, audio and video. The first page of a website is called home page. Each website has specific internet address (URL) that you need to enter in your browser to access a website.

Static Website	Dynamic Website
Prebuilt content is same every time the page is loaded.	Content is generated quickly and changes regularly.
It uses the **HTML** code for developing a website.	It uses the server side languages such as **PHP,SERVLET, JSP, and ASP.NET** etc. for developing a website.

It sends exactly the same response for every request.	It may generate different HTML for each of the request.
The content is only changed when someone publishes and updates the file (sends it to the web server).	The page contains "server-side" code which allows the server to generate the unique content when the page is loaded.

HTTP (Hyper Text Transfer Protocol)

The Hypertext Transfer Protocol (HTTP) is application-level protocol for collaborative, distributed, hypermedia information systems. It is the data communication protocol used to establish communication between client and server.

HTTP is TCP/IP based communication protocol, which is used to deliver the data like image files, query results, HTML files etc on the World Wide Web (WWW) with the default port is TCP 80. It provides the standardized way for computers to communicate with each other.

HTTP Requests

The request sent by the computer to a web server, contains all sorts of potentially interesting information; it is known as HTTP requests. The HTTP client sends the request to the server in the form of request message which includes following information:

- The Request-line
- The analysis of source IP address, proxy and port
- The analysis of destination IP address, protocol, port and host
- The Requested URI (Uniform Resource Identifier)
- The Request method and Content
- The User-Agent header
- The Connection control header

Get vs. Post

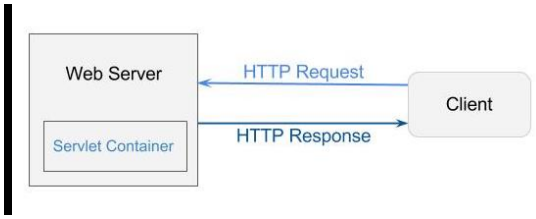
There are many differences between the Get and Post request. Let's see these differences:

GET	POST

1) In case of Get request, only **limited amount of data** can be sent because data is sent in header.	In case of post request, **large amount of data** can be sent because data is sent in body.
2) Get request is **not secured** because data is exposed in URL bar.	Post request is **secured** because data is not exposed in URL bar.
3) Get request **can be bookmarked.**	Post request **cannot be bookmarked.**
4) Get request is **idempotent** . It means second request will be ignored until response of first request is delivered	Post request is **non-idempotent.**
5) Get request is **more efficient** and used more than Post.	Post request is **less efficient** and used less than get.

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 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.

Servlet Interface

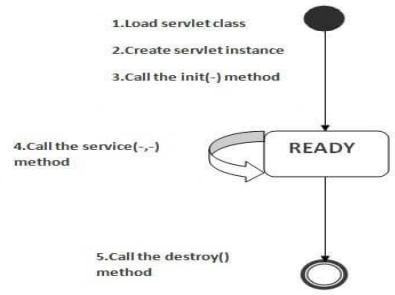
Servlet interface provides commonbehaviorto all the servlets.Servlet interface defines methods that all servlets must implement. Servlet interface needs to be implemented for creating any servlet (either directly or indirectly). It provides 3 life cycle methods that are used to initialize the servlet,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

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.

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:

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.



As displayed in the above diagram, there are three states of a servlet: new, ready and end. The servlet is in new state if servlet instance is created. After invoking the init() method, Servlet comes in the ready state. In the ready state, servlet performs all the tasks. When the web container invokes the destroy() method, it shifts to the end state.

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.

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:

GenericServlet class

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

You may create a generic servlet by inheriting the GenericServlet class and providing the implementation of the service method.

Methods of GenericServlet class

There are many methods in GenericServlet class. They are as follows:

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 String getServletInfo()** returns information about servlet such as writer, copyright, version etc.
5. **public void init()** it is a convenient method for the servlet programmers, now there is no need to call super.init(config)
6. **public String getInitParameter(String name)** returns the parameter value for the given parameter name.
7. **public String getServletName()** returns the name of the servlet object.

Servlet Example by inheriting the GenericServlet class

```
1. import java.io.*;
2. import javax.servlet.*;
3. public class First extends GenericServlet{
4. public void service(ServletRequest req,ServletResponse res)
5. throws IOException,ServletException{
6. res.setContentType("text/html");
7. PrintWriter out=res.getWriter();
8. out.print("<html><body>");
9. out.print("<b>hello generic servlet</b>");
10. out.print("</body></html>");
11. } }
```

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

There are many methods in HttpServlet class. They are as follows:

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.
5. **protected void doDelete(HttpServletRequest req, HttpServletResponse res)** handles the DELETE request. It is invoked by the web container.

JSP - JavaBeans

A JavaBean is a specially constructed Java class written in the Java and coded according to the JavaBeans API specifications.

Following are the unique characteristics that distinguish a JavaBean from other Java classes –

- It provides a default, no-argument constructor.
- It should be serializable and that which can implement the **Serializable** interface.
- It may have a number of properties which can be read or written.
- It may have a number of "**getter**" and "**setter**" methods for the properties.

JavaBeans Properties

A JavaBean property is a named attribute that can be accessed by the user of the object. The attribute can be of any Java data type, including the classes that you define.

A JavaBean property may be **read, write, read only**, or **write only**. JavaBean properties are accessed through two methods in the JavaBean's implementation class –

S.No.	Method & Description
1	get**PropertyName**() For example, if property name is *firstName*, your method name would be **getFirstName()** to read that property. This method is called accessor.
2	set**PropertyName**() For example, if property name is *firstName*, your method name would be **setFirstName()** to write that property. This method is called mutator.

A read-only attribute will have only a **getPropertyName()** method, and a write-only attribute will have only a **setPropertyName()** method.

Accessing JavaBeans

The **useBean** action declares a JavaBean for use in a JSP. Once declared, the bean becomes a scripting variable that can be accessed by both scripting elements and other custom tags used in the JSP. The full syntax for the useBean tag is as follows −

```
<jsp:useBean id = "bean's name" scope = "bean's scope" typeSpec/>
```

Here values for the scope attribute can be a **page, request, session** or **application based** on your requirement. The value of the **id** attribute may be any value as a long as it is a unique name among other **useBean declarations** in the same JSP.

Accessing JavaBeans Properties

Along with **<jsp:useBean...>** action, you can use the **<jsp:getProperty/>** action to access the get methods and the **<jsp:setProperty/>** action to access the set methods.

For example refer : Jsp-javabeans tutorialspont(website)

JSP - Cookies Handling

Cookies are text files stored on the client computer and they are kept for various information tracking purposes. JSP transparently supports HTTP cookies using underlying servlet technology.

There are three steps involved in identifying and returning users −

- Server script sends a set of cookies to the browser. For example, name, age, or identification number, etc.
- Browser stores this information on the local machine for future use.
- When the next time the browser sends any request to the web server then it sends those cookies information to the server and server uses that information to identify the user or may be for some other purpose as well.

The Anatomy of a Cookie

Cookies are usually set in an HTTP header (although JavaScript can also set a cookie directly on a browser). A JSP that sets a cookie might send headers that look something like this –

```
HTTP/1.1 200 OK
Date: Fri, 04 Feb 2000 21:03:38 GMT
Server: Apache/1.3.9 (UNIX) PHP/4.0b3
Set-Cookie: name = xyz; expires = Friday, 04-Feb-07 22:03:38 GMT;
  path = /; domain = tutorialspoint.com
Connection: close
Content-Type: text/html
```

As you can see, the **Set-Cookie header** contains **a name value pair, a GMT date, a path** and **a domain**. The name and value will be URL encoded. The **expires** field is an instruction to the browser to **"forget"** the cookie after the given time and date.

If the browser is configured to store cookies, it will then keep this information until the expiry date. If the user points the browser at any page that matches the path and domain of the cookie, it will resend the cookie to the server. The browser's headers might look something like this −

```
GET / HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.6 (X11; I; Linux 2.2.6-15apmac ppc)
Host: zink.demon.co.uk:1126

Accept: image/gif, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
Cookie: name = xyz
```

A JSP script will then have access to the cookies through the request method ***request.getCookies()*** which returns an array of *Cookie* objects.

Setting Cookies with JSP

Setting cookies with JSP involves three steps −

Step 1: Creating a Cookie object

You call the Cookie constructor with a cookie name and a cookie value, both of which are strings.

```
Cookie cookie = new Cookie("key","value");
```

Step 2: Setting the maximum age

You use **setMaxAge** to specify how long (in seconds) the cookie should be valid. The following code will set up a cookie for 24 hours.

```
cookie.setMaxAge(60*60*24);
```

Step 3: Sending the Cookie into the HTTP response headers

You use **response.addCookie** to add cookies in the HTTP response header as follows

```
response.addCookie(cookie);
```

Reading Cookies with JSP

To read cookies, you need to create an array of *javax.servlet.http.Cookie* objects by calling the **getCookies( )** method of *HttpServletRequest*. Then cycle through the array, and use **getName()** and **getValue()** methods to access each cookie and associated value.

Delete Cookies with JSP

To delete cookies is very simple. If you want to delete a cookie, then you simply need to follow these three steps −

- Read an already existing cookie and store it in Cookie object.
- Set cookie age as zero using the **setMaxAge()** method to delete an existing cookie.
- Add this cookie back into the response header.

JSP - Session Tracking

HTTP is a "stateless" protocol which means each time a client retrieves a Webpage, the client opens a separate connection to the Web server and the server automatically does not keep any record of previous client request.

Cookies

A webserver can assign a unique session ID as a cookie to each web client and for subsequent requests from the client they can be recognized using the received cookie.

This may not be an effective way as the browser at times does not support a cookie. It is not recommended to use this procedure to maintain the sessions.

The session Object

Apart from the above mentioned options, JSP makes use of the servlet provided HttpSession Interface. This interface provides a way to identify a user across.

- a one page request or
- visit to a website or
- store information about that user

By default, JSPs have session tracking enabled and a new HttpSession object is instantiated for each new client automatically. Disabling session tracking requires explicitly turning it off by setting the page directive session attribute to false as follows −

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

The JSP engine exposes the HttpSession object to the JSP author through the implicit **session** object. Since **session** object is already provided to the JSP programmer, the programmer can immediately begin storing and retrieving data from the object without any initialization or **getSession()**.

Deleting Session Data

When you are done with a user's session data, you have several options −

- **Remove a particular attribute** − You can call the ***public void removeAttribute(String name)*** method to delete the value associated with the a particular key.
- **Delete the whole session** − You can call the ***public void invalidate()*** method to discard an entire session.
- **Setting Session timeout** − You can call the ***public void setMaxInactiveInterval(int interval)*** method to set the timeout for a session individually.

- **Log the user out** − The servers that support servlets 2.4, you can call **logout** to log the client out of the Web server and invalidate all sessions belonging to all the users.
- **web.xml Configuration** − If you are using Tomcat, apart from the above mentioned methods, you can configure the session time out in web.xml file as follows.

```
<session-config>
  <session-timeout>15</session-timeout>
</session-config>
```

What is SSL/TLS and HTTPS?

SSL is an acronym for **Secure Sockets Layer**. A type of digital security that allows encrypted communication between a website and a web browser. The technology is currently **deprecated** and has been replaced entirely by TLS.

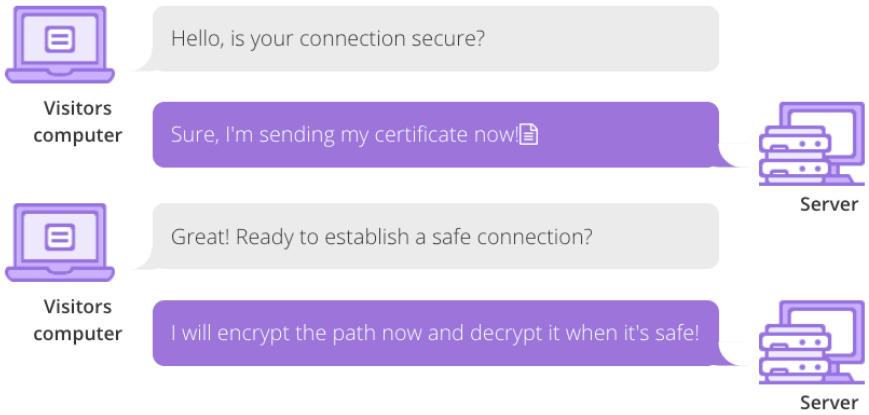
TLS stands for **Transport Layer Security** and it ensures data privacy the same way that SSL does. Since SSL is actually no longer used, this is the correct term that people should start using.

HTTPS is a **secure** extension of HTTP. Websites that install and configure an SSL/TLS certificate can use the HTTPS protocol to establish a secure connection with the server.

- The goal of SSL/TLS is to make it safe and secure to transmit sensitive information including personal data, payment or login information.
- It's an alternative to plain text data transfer in which your connection to a server is unencrypted, and it makes it harder for crooks and hackers to snoop on the connection and steal your data.
- Most people are familiar with SSL/TLS certificates, which are used by webmasters to secure their websites and to provide a secure way for people to carry out transactions.
- You can tell when a website is using one because you'll see a little padlock icon next to the URL in the address bar.

How Do SSL/TLS Certificates Work?

SSL/TLS certificates work by digitally tying a cryptographic key to a company's identifying information. This allows them to encrypt data transfers in such a way that they can't be unscrambled by third parties.



SSL/TLS works by having both a private and a public key, as well as session keys for every unique secure session. When a visitor enters an SSL-secured address into their web browser or navigates through to a secure page, the browser and the web server make a connection.

During the initial connection, the public and private keys will be used to create a session key, which will then be used to encrypt and decrypt the data that's being transferred. This session key will remain valid for a limited time and only be used for that particular session.

You can tell whether a website is using SSL by looking for a padlock icon or a green bar at the top of your browser. You should be able to click on this icon to view the information on who holds the certificate and to manage your SSL settings.

References

1. . https://www.javatpoint.com/
2. https://www.tutorialspoint.com/
