Overview of JSP (Java Server Pages):

The main purpose of the online applications in the Enterprise applications area is to get dynamic responses from the server machine. To design web applications on the server-side we may use Web Technologies like CGI, Servlets, JSP then on.

CGI is essentially a Process-based technology because it had been designed on the idea of C technology. If we deploy any CGI application at the server then for each request CGI container will generate a separate process. In the above context, if we send multiple numbers of requests to an equivalent CGI application then the CGI container has got to generate multiple numbers of processes at the server machine. To handle the multiple numbers of processes at a time server machine has got to consume a greater number of system resources, as a result, the performance of the server-side application will be reduced.

To overcome the above problems, we have to use Thread based technology at the server-side like servlets. In web application development, servlets are excellent at the time of devour the request and process the request but servlets aren't good at the time of generating the dynamic response to the client.

Servlet may be a Thread based technology, if we deploy it at the server then the container will create a separate thread rather than the method for each request from the client. Due to this Thread based technology at the server-side, server-side application performance will be increased. In the case of the servlet, we are unable to separate both presentation logic and business logic. If we perform any modifications on servlets then we must perform recompilation and reloading. If we would like to style web applications by using servlets then we must require excellent knowledge of Java technology.JSP may be a server-side technology provided by Sun Microsystems to style web applications so as to get a dynamic response. The main intention to introduce JSP technology is to scale back java code the maximum amount as possible in web applications. JSP technology may be a server-side technology, it had been designed on the idea of Servlet API and Java API. In web application development, we'll utilize JSP technology to organize the view part or presentation part. JSP technology is extremely good at the time of generating a dynamic response to the client with an excellent look and feel. If we would like to style any web application with JSP technology then it's not required to possess java knowledge. In the case of JSP technology, we are ready to separate presentation logic and business logic because to organize presentation logic we'll use HTML tags, and to organize business logic we will use JSP tags separately. If we perform any modifications on JSP pages then it is not required to perform recompilation and reloading because JSP pages are auto-compiled and auto-loaded.

What is JSP?

The Java Server Pages 1.2 specification provides web developers with a framework to create applications containing dynamic web pages like HTML, DHTML, XHTML, and XML. A JSP page may be a text-based document containing static HTML and dynamic actions that describe the way to process a response to the client in a more powerful and versatile manner. Most of a JSP file is obvious HTML but it also has, interspersed with it, special JSP tags.

The web application is a collection of web resources programs having the capability to generate web pages. To develop these web resources programs, we need web technology. For every request given to the CGI program, one process is heavyweight consume more resources. Due to this CGI applications performance is very poor. For every request given to servlet / JSP pages, one thread will be created. These threads are lightweight to consume fewer resources. Due to this servlet / JSP application gives a very good performance.

JSP stands for Java Server Pages. JSP technology builds on Java. JSP or Java Server Pages was developed by Sun Microsystem. JSP technology is an object-oriented programming language and is predicated on Java Language. JSP is a part of JEE. JSP is a web technology used for developing dynamic websites. JSP is an API introduced by SUN Microsystems which is also used to develop web-based apps. JSP files can be located inside the project folder. The extension name of the JSP file should be .jsp. We no need to configure the JSP file inside the web.xml. We can directly use the JSP file name as a URL in the address bar. We have many advantages compared to servlets. Servlets take many steps to develop where JSP takes fewer steps to save the development time. JSP will divide our project into presentation logic and business logic. JSP will always contain presentation logic where servlets will contain business logic.

Problems with Servlet

In the initial days of Servlets, Sun Microsystems did not attract ASP programmers towards servlet because ASP supports tag-based programming that servlet doesn't support. To work with Servlet strong knowledge is required. It is very difficult for an ASP programmer to learn Java.

To overcome the above problems Sun Microsystems has given a tag-based technology called JSP having all the features of Servlet. So, any programmer can use JSP without having strong knowledge of Java/Servlet. Below are the limitations of Servlet:

1. It is not suitable for non-java programmers as strong knowledge in Java is required to work with Servlets.
2. Placing HTML code in the Servlet program is a complex and error-prone process.
3. Makes the programmer mix up presentation logic (HTML) and business logic (Java code).
4. Any modification in the Servlet program will be reflecting only after recompilation of the Servlet program overloading of Web Application.
5. Doesn't gives implicit object request, response, ServletContext and etc. are not implicit object, they are container created object. So, we need to write additional code to access those objects.
6. ServletConfig in web.xml is mandatory.
7. Servlet is not good for business logic not good for persistence logic.

Why JSP pages are more advantageous than Servlet?

To overcome the problems of Servlets Sun Microsystems has given a tag-based technology called JSP having all the features of Servlet. So, any programmer can use JSP without having strong knowledge of Java/Servlet.

1. It supports tag-based programming.
2. It gives nine implicit objects.
3. It allows us to write separate both business logic and presentation logic.
4. Modifications done in the JSP program will be reflected automatically without reloading the web application.
5. Strong Java knowledge is not required.
6. Suitable for both java and non-java programmers.
7. Configuration of JSP in web.xml file is optional.

8. Allows us to use all the features of servlet because every JSP program internally and equivalent servlet program will be generated.
9. Gives automatic exception handling supports.
10. Every JSP program and equivalent servlet program will be generated internally using the page compilation process. Every server provides one JSP page compiler in the JSP container.
11. Whenever the JSP program is executing it means JSP equivalent servlet is executing internally.

Note: In the initial day's programmer have to use JSP as a complete alternate to Servlet. But now a day's programmers are using both servlet and JSP technology together in web application development.

Advantages of JSP

- JSP pages are faster because they are pre-compiled.
- It is flexibles as it's part of the Sun's J2EE and can be used with other types of Java technologies, such as Servlet.
- It is a 'fly by night' technology as multiple vendors support it because of its specifications.
- It is easy to develop because HTML can be easily maintained with JSP. Relatively minimal programming skills required.
- JSP page code is not visible to the client, only generated HTML is visible.
- It is easy to read, write, and maintain JSP pages.
- It is easy to extend the JSP language by using pluggable, reusable tag libraries available.

Disadvantages of JSP

- It is difficult to trace errors that occurred in JSP pages because JSP pages are translated to servlets and compiled.
- It requires double the disk space to hold JSP pages.
- As JSP pages are to be compiled on the server it requires more time when accessed for the first time.
- The initial compilation produces a noticeable delay when accessing the JSP page for the first time.
- It is difficult to debug the JSP page.

Why JSP?

JSP technology was designed to simplify the process of creating pages by separating web presentations from web content. JSP is a collection of HTML tags and JSP tags and Java code. Every JSP page can be placed in the root folder or WEB-INF. The JSP placed in the root folder is a public resource and this resource can access by the client directly. The JSP placed inside WEB-INF is a private resource, which can't access directly by the client. Client access this resource using public URL defined inside web.xml.

```
<web-app>
<servlet>
 <servlet-name>instanceName</servlet-name>
 <jsp-file>jspfilename</jsp-file>
 <init-param>
 <param-name>name</param-name>
 <param-value>value</param-value>
 </init-param>
 .
 .
 .
</servlet>
<servlet-mapping>
<servlet-name>instancename</servlet-name>
<url-pattern>/publicurl</url-pattern>
</servlet-mapping>
</web-app>
```

Features of JSP

Simplifies the process of Development: It allows the programmer to insert the Java code directly into the JSP file, making the development process easier. Although JSP files are HTML files, they use special tags containing the java source code which provides a dynamic ability.

Portability: The java feature of "write once, run anywhere" is applicable to JSP. JSP is platform-independent making it portable across any platform and therefore multi-platform.

Independency of Layers: There is a clear separation between the presentation and implementation layer.

Creating a Simple JSP page

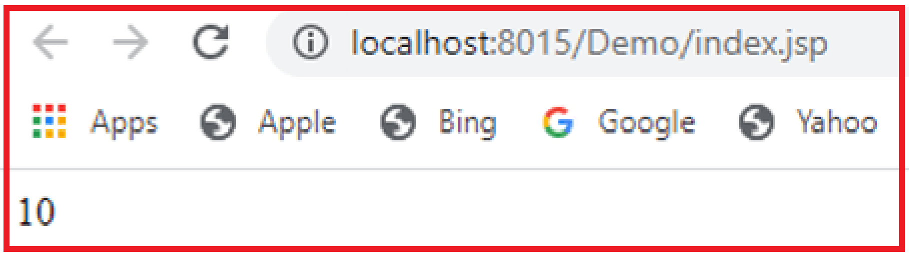
To create your first JSP page, write HTML code, and save it by the .jsp extension. Put the code in a folder and paste it into the web-apps directory in apache tomcat to run the JSP page.

index.jsp

```
<%@   page   language="java"   contentType="text/html;
charset=ISO-8859-1"

 pageEncoding="ISO-8859-1"%>

<!DOCTYPE html>

<html>

<head>

<meta charset="ISO-8859-1">

<title>Insert title here</title>

</head>

<body>

 <%

  out.print(2 * 5);

 %>
```

```
</body>
</html>
```

Output:


```
localhost:8015/Demo/index.jsp
10
```

What is JSP Container?

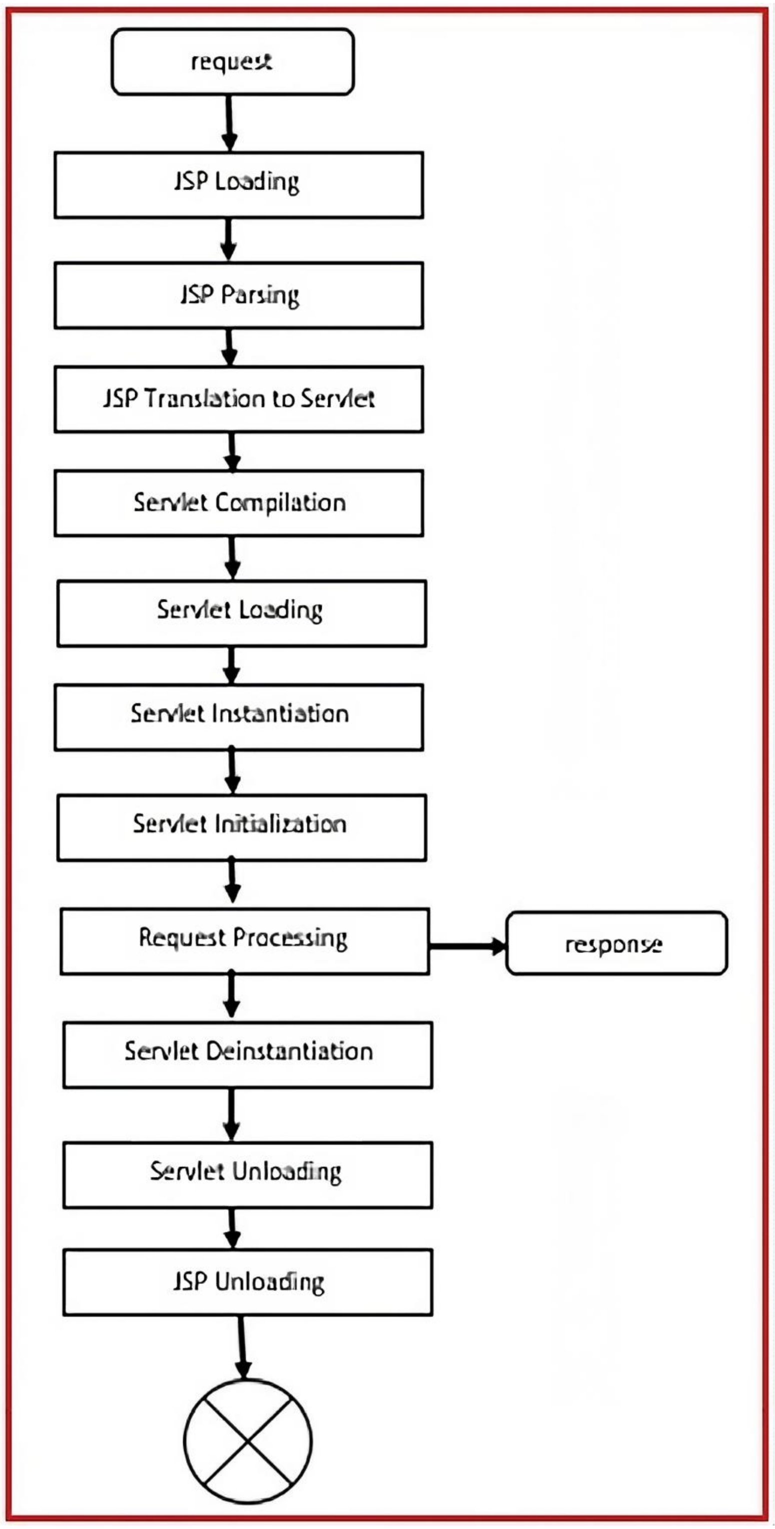
JSP container is a program that provides an implementation for JSP API which is responsible for running the Servlet. If we want to develop any web-based application, we have to use a servlet container, JSP container, HTTP server. Nowadays all these things are integrated into any JEE server.

What is JSP Life Cycle?

JSP Life Cycle is defined as the translation of the JSP Page into the servlet. Because JSP Page always needs to be converted into servlet page first in to process the service requests. Every JSP page ends with .jsp extension. Whenever we were given a request to the server for a .jsp file then first .jsp will be converted into a .java file. Second, .java file is converted into .class file. Third, .class is loaded by container and managed by the container. Basically .jsp to .java and .java to .class conversion is performed by JSP engine. All .class file loading and execution is performed by the container (servlet container). The JSP Life Cycle always starts with the creation of the JSP page and ends with the disintegration of that.

Different Phases of JSP Life Cycle

JSP page life cycle contains the following phases:



When we send a request from the client to the server for a specific JSP page then the container will devour the request, identify the requested JSP pages, and perform the subsequent life cycle actions:

JSP Loading:

In this phase, the container will load the JSP files to the memory from the web application directory structure.

JSP Parsing:

In this phase, the container will check whether all the tags available on the JSP page are in well-formed format or not.

JSP Translation to Servlet:

When JSP receives the first request from the client, the JSP container translates JSP into Servlet. After the JSP Parsing container will translate the loaded JSP page into a specific servlet. While executing a JSP page Tomcat container will provide the translated servlet within the following location at Tomcat Server.

If the JSP file name is first.jsp then Tomcat Server will provide a servlet with name first_jsp. All the translated servlets provided by Tomcat container are by default final. The default superclass for a translated servlet is HttpJspBase. Where JSPPage interface has declared the following methods:

1. public void _JspInit()
2. public void _JspDestroy()

and the HttpJspPage interface has provided the following method:

1. **public void _JspService(HttpServletRequest req, HttpServletResponse res)**

For the above three abstract methods, HttpJspBase class has provided the default implementation but the _JspService(_,_) method would be overridden in the first_jsp class with the content that we provided in first.jsp file.

Servlet Compilation

After successful translation of servlet container will compile servlet java file and generates the respective .class file. Here, the source program is compiled and generates byte code or .class file.

Servlet Loading

In this phase, the container will load the translated servlet class byte code to the memory. Here, the compiled .class file is loaded into JVM.

Servlet Instantiation

In this phase, the container will load the translated servlet class byte code to the memory. Here, creates an object of the loaded class.

Servlet Initialization

In this phase, the container will access the _JspInit() method to initialize the servlet. Here, JSP is initialized by calling JspInit() method.

Creating request and response objects

After the servlet is initialized successfully, the container will create a thread to access the _JspService(_,_) method, for this container has to create HttpServletRequest and HttpServletResponse.

Generating Dynamic Response:

After getting request and response objects container will access the _JspService(_,_) method, by executing its content and then the container will generate some response on the response object.

Dispatching dynamic response to the client:

When container generated thread reached the ending point of the _JspService(_,_) method then that thread is going to be in a Dead state, with this container will dispatch dynamic response to the client through the Response Format.

Destroying request and response objects

When the dynamic response reached to client protocol will terminate its virtual socket connection, with this container will destroy request and response objects.

Servlet Deinstantiation

After destroying request and response objects container is going to be in a waiting state depends on the container, then the container identifies no further request for an equivalent resource then the container will destroy the servlet object, for this container will execute the _JspDestroy() method.

Servlet and JSP Unloading:

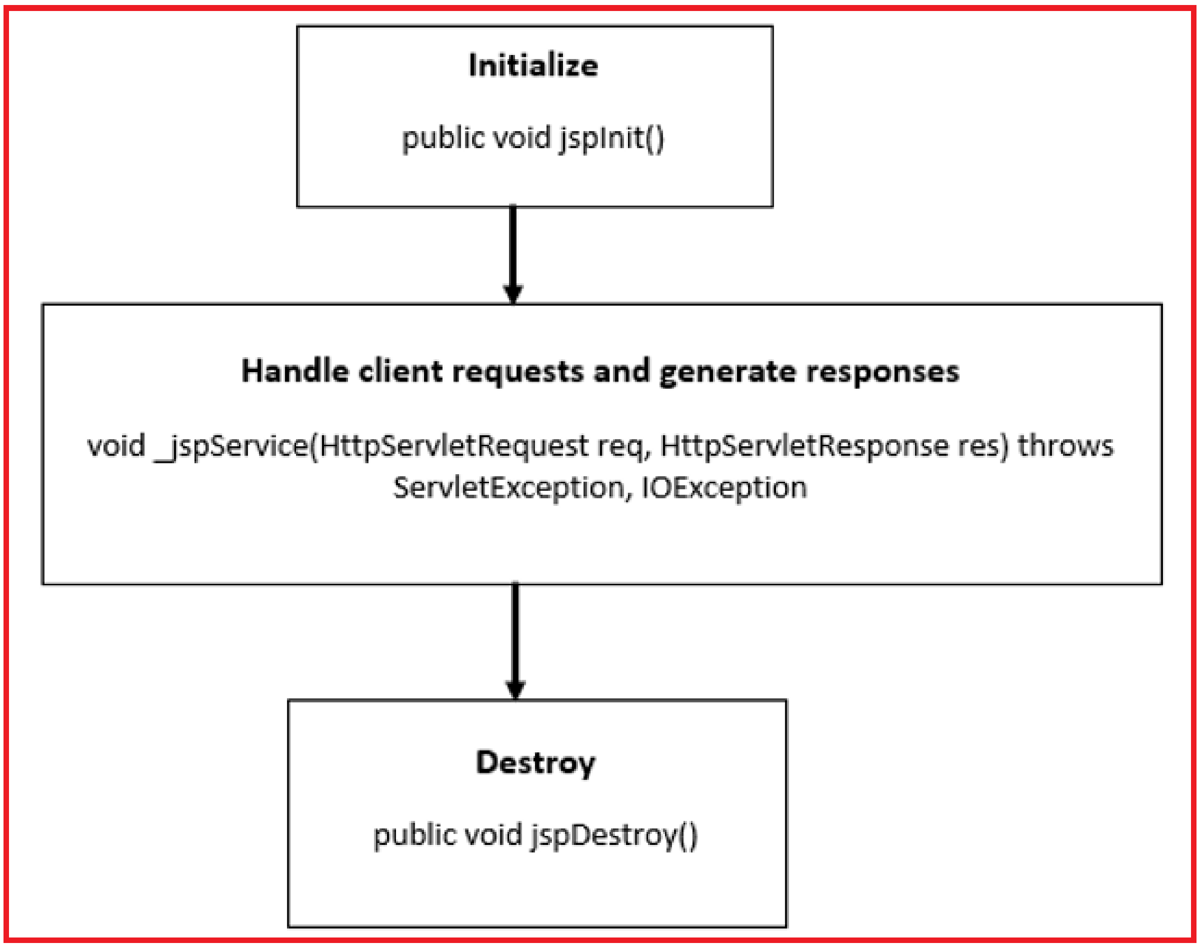
After the servlet de-instantiation container will eliminate the translated servlet byte code and Jsp code from memory.

Flow of Execution

First, the container loads .jsp file from memory. Then container verifies the syntax of JSP tags placed in the .jsp file. Using the JSP page compiler JSP program will be converted into an equivalent servlet class. The java compiler (javac) compiles the JES class source file. Then container loads JES class and creates the object. Container then calls jspInit() method through init(ServletConfig) method to initialize ServletConfig object with JES class object. Then it creates one set of requests, a response object from the current request. The container then calls the _jspService(_,_) method of JES through service(_,_) to process the request. The output generated by _jspService(_,_) method goes to the browser as a response. Once the response is delivered to the request, the response object will be destroyed. If there is no further request to JSP or if the web application stopped/reloaded the container performs servlet de-instantiation. When this is about to happen container calls jspDestroy() through the destroy() method. JVM then unloads the loaded JES class. At last container unloads the loaded .jsp file.

Different Methods of JSP Life Cycle

The lifecycle of JSP is controlled by three methods which are automatically called, when a JSP is requested and when the JSP terminates normally. These are: jspInit(), _jspService() and jspDestroy()


Initialize
public void jspInit()
Handle client requests and generate responses
void _jspService(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
Destroy
public void jspDestroy()

jspInit() Method:

jspInit() method is similar to the init() method in a Java Servlet and an applet. It is called first when the JSP is requested and is employed to initialize objects and variables that are used throughout the lifetime of the JSP.

_jspService() Method:

_jspService() method is automatically called and retrieves a connection to HTTP. It will call the doGet or doPost() method of servlet created.

jspDestroy() Method:

jspDestroy() method is similar to the destroy() method in Servlet. The destroy() method is automatically called when the JSP terminates normally. It isn't called by JSP when it terminates. It is used for cleanup where resources used during the execution of the JSP are released.

Difference between Web Server, Application Server, and Web Container

1. **Web Server**
   - Web Servers are responsible for serving static content e.g., HTML over HTTP protocol.
   - Apache and IIS are two popular web servers. Apache is used everywhere including Java world but IIS is most popular in Microsoft ASP .NET world.
   - It is expected from a web server to provide HTTP protocol level service.
   - We need a web server like Apache HTTPD if we are serving static web pages.

2. **Application Server**
   - Application Server is responsible for serving dynamic content, managing the EJB pool, facilitating the distributed transactions, facilitating application lookup over JNDI, application security, and others.
   - From the Java EE perspective couple of popular application servers are IBM Websphere, Oracle WebLogic, glassfish, and Redhat's Joss.
   - Application Server is supposed to provide more powerful and dynamic Web service and business level service via EJB (Enterprise Java Beans).
   - If we have a Java EE application using EJB, distributed transaction, messaging and other fancy features then we need a full-fledged application server like JBoss, WebSphere, or Oracle's WebLogic.

3. **Web Container**
   - Web Containers are only responsible for generating HTML by executing JSP and Servlet on Server Side.
   - In Web Containers, Apache Tomcat and Jetty are two of the most popular Servlet engines in the java web world.
   - Essential Services like Database connection pooling is not only provided by application server but also by Web Containers like Tomcat.
   - If we have a Java application with just JSP and servlet to generate dynamic content then you need web containers like Tomcat or Jetty.

JSP Processing

Before a JSP page is sent to a browser, the server must process all the JSP elements it contains. This processing is performed by a web container, which can be either a native part of a web server or a separate product attached to the web server. The web container turns the JSP page into a Java servlet, then executes the servlet.

Converting the JSP page into a servlet (known as the JSP page implementation class) and compiling the servlet take place in the translation phase. The web container initiates the translation phase for a JSP page automatically when the first request for the page is received. The translation phase takes a bit of time, of course, so users may notice a slight delay the first time they request a JSP page. The translation phase can also be initiated explicitly, to avoid hitting the first user with the delay. This is referred to as precompilation.

The web container is also responsible for invoking the JSP page implementation class to process each request and generate responses. This is called the request processing phase. The two phases are illustrated in Figure 1-2.

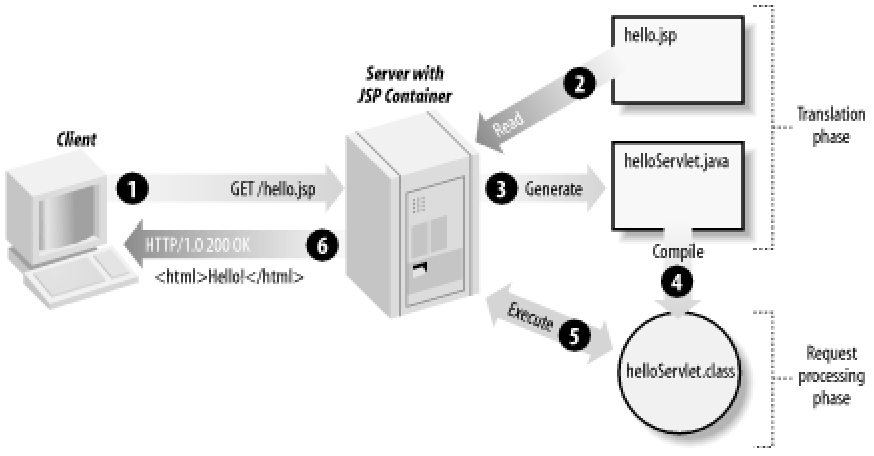


Figure 1-2. JSP page translation and processing phases

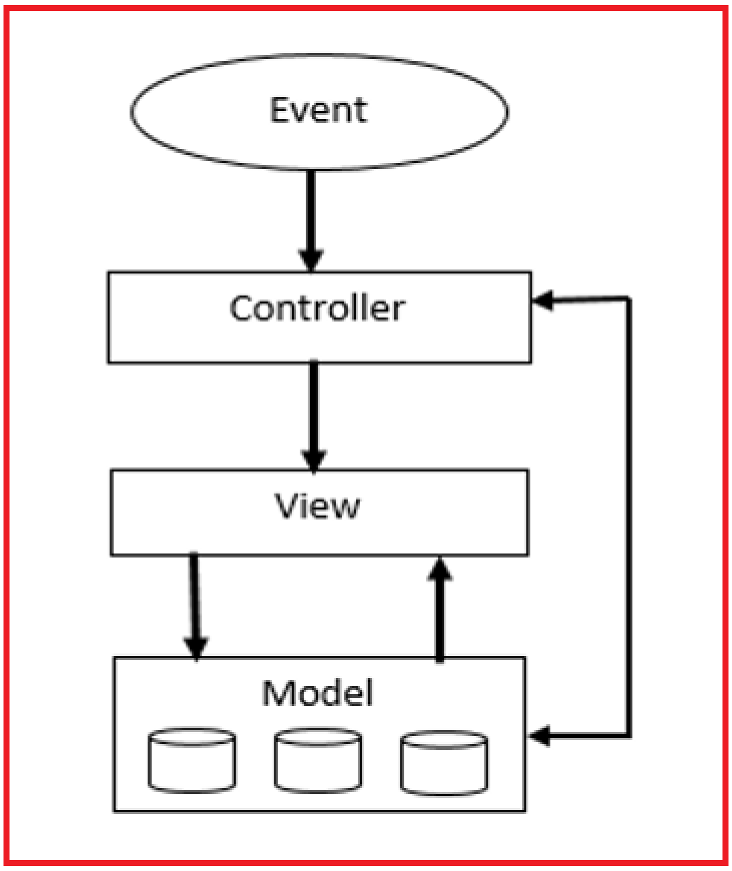
As long as the JSP page remains unchanged, the translation phase is skipped. When the page is modified, it goes through the translation phase again.

What is MVC?

MVC stands for Model-View-Controller. It is a design pattern used for developing web applications. It is a layout pattern used to isolate the information, presentation logic, business logic. Web application Logic is divided into three logics:

1. Presentation Logic
2. Business Logic
3. Persistent Logic

Each one is called Model or Tier. MVC is popular because it isolates the appliance logic from the interface layer and supports the separation of concerns. Here the Controller receives all requests for the appliance then works with the Model to organize any data needed by the View. The View then uses the information prepared by the Controller to get a final presentable response. The MVC abstraction is often graphically represented as follows.



What is the Model?

It is a module or tier which contains persistent logic or database operation. The model uses JDBC, Hibernate, JPA. The Model segment compares each one among the knowledge-related rationale that the client works with. This can speak to either the information that is being exchanged between the View and Controller segments or some other business-related information.

What is a Controller?

The controller controls the flow of execution of web applications. In web application development servlet is called a controller. Controller receive request/input from the client and uses other modules to perform operations. The controller contains Business Logic. Controllers set about as an interface amongst Model and consider segments to process all the business rationale and approaching solicitations, control information utilizing the Model part and cooperate with the Views to render the last yield.

What is View?

The view is used for generating input and output. It contains presentation logic. This is developed using HTML or JSP. The View part is utilized for all the UI rationale of the application.

Advantages of MVC

- The main advantage of using MVC in a Web Application is it makes complex applications easy to manage with divisions of Model, View, and Controllers.
- MVC provides a strong routing mechanism with a Front Controller pattern.
- MVC works well for development with large teams with multiple web developers and designers working simultaneously.
- In MVC, separating the model from View makes the web application more robust and easier to maintain.
- In MVC, separating the Controller from the Model allows configurable mapping of user actions on the Controller to application functions on the Model.

Example of MVC in JSP Application

In this example, we are showing how to use MVC architecture in JSP. In this example, we are creating an example in which servlet as a controller, JSP as a view component, Java Bean class as a model. In this example, we have created 6 pages.

1. index.jsp
2. ControllerServlet.java
3. LoginBean.java
4. login-success.jsp
5. login-error.jsp
6. web.xml

Where index.jsp helps us to get the input from the user. ControllerServlet.java acts as a controller, login-success.jsp and login-error.jsp file acts as a view component. Login-success.jsp page will show the "Welcome" message if the user will successfully login whereas login-error.jsp page will show the error message and returns you back to the index.jsp page. LoginBean.java acts as a Model Layer and a web.xml file is used to mapping the servlet.

What are JSP Directives?

Directives are basically wont to configure the code that's generated by the container during a Servlet. JSP directives are JSP components that are used to give the instructions to the JSP compiler. JSP directives will simplify writing JSP files. These tags are used to give directions to the JSP page compiler. In web applications, JSP Directives can be used to define present JSP page characteristics, to include the target resource content into the present JSP page, and to make available user-defined tag library into this JSP page. All the JSP directives are going to be resolved at the time of translating the JSP page to the servlet. The majority of JSP Directives will not give a direct effect to response generation.

To provide JSP Directives in JSP pages we have to use the following syntaxes:

JSP-Based Syntax: <%@Directive_name[attribute-list]%>

Example: < %@page import="java.io.*"%>

XML-Based Syntax: <jsp:directive.directiveName[attribute-list]%/>

Example: <jsp:directive.page import="java.io.*"/>

Types of Directives in JSP:

There are three types of Directives in JSP technology:

1. Page Directives
2. Include Directives
3. Taglib Directives

Page Directives in JSP

In JSP technology, page directives are often wont to define this JSP page characteristics wish to define import statements, specify particular superclass to the translated servlet, to specify metadata about present JSP pages, and so on. The JSP page directives define the properties for the entire JSP page by using its different attributes and set values of the attributes as per requirements. It is useful to provide global information for the JSP page. Basically, Page directives are used to give the instructions to the JSP compiler like what package to import, what language we are writing, etc. Page directives are basically used for supplying compile-time information to the container for generating a servlet. The page directive will take the info within the sort of (key, value) pair. Whenever we use page directive as a part of the JSP program that statement must be the first statement. The scope of the page directive is applicable to the current JSP page only. JSP program allows only one-page directive at a time.

Syntax: `<%@ page attributeName=”values” %>`

The space between the tag <%@ and %> before the page (directive name) and after values of the last attribute, is optional, you can leave the space or not.

Include Directives in JSP

Include Directive are often wont to include the content of the target resource into this JSP page. The include directive is employed to incorporate a file during the interpretation phase. This directive tells the container to merge the content of other external files with the present JSP during the interpretation phase. You may code include directives anywhere on your JSP page. It is used to include the code of destination program/file to the code of source JSP program’s JES class. This tag is given for code inclusion, not for output inclusion. This tag is given for code inclusion, not for output inclusion. Here, a separate JES class will not be generated for the destination file but the code of the destination file will be included in the JES class of the source file. This method does not use the rd.include(_,_) method internally. The include directive is employed to statically insert the contents of a resource into the present JSP. This enables a user to reuse the code without duplicating it and includes the contents of the required file at the interpretation time. This directive has just one attribute called a file that specifies the name of the file to be included. One JSP page can have more than one include directives.

Syntax: `<%@ include file=”–”%>`

JSP Taglib Directives

The main purpose of Taglib Directives is to make available user-defined tag library into the present JSP pages. The JSP API allows you to define custom JSP tags that appear as if HTML or XML tags and a tag library may be a set of user-defined tags that implement custom behavior. The taglib directives declare that your JSP page uses a set of custom tags, identifies the location of the library, and provides a means for identifying the custom tags in your JSP page.

Syntax: `<%@ taglib uri="—" prefix="—"%>`

Where URI attribute can be used to specify the name and location of user-defined tag library and prefix attribute can be used to define prefix names for the custom tags.

What are JSP actions?

The servlet compartment gives much implicit usefulness to facilitate the improvement of the applications. Software engineers can utilize these capacities in JSP applications. The JSP Actions labels empower the software engineer to utilize these capacities. The JSP Actions are XML labels that can be utilized on the JSP page. These labels inside utilized functionalities of the servlet-programming interface to perform tasks straightforwardly at the execution stage or solicitation handling stage.

In JSP innovation, by utilizing scripting components we can give java code inside the JSP pages, yet the fundamental of JSP innovation isn't to permit java code inside JSP pages. To dispose of java code from JSP pages we need to wipe out scripting components, to kill scripting components from JSP pages we need to give an option, for example, JSP Actions. If there should be an occurrence of JSP Actions, we will characterize a scripting tag on the JSP page and we will give a square of java code w.r.t. scripting tag. At the point when the compartment experiences the scripting label the holder will execute individual java code, by this, an activity will be performed called JSP Action.

Types of JSP Actions

In JSP technology there are two types of actions:

1. Standard Actions
2. Custom Actions

Standard Actions

Standard activities are implicit labels of JSP. It very well may be characterized by the JSP innovation to play out a specific activity. JSP innovation has given all the standard activities as a bunch of predefined labels called Action Tags. These are essentially used to pass runtime data to the compartment. As a part of JSP we have the following standard actions:

The <jsp:include> Action

In JSP pages, <jsp:include> activity tag can be utilized to incorporate the substance of the objective asset into the present JSP page. <jsp:include> activity tag can be utilized to incorporate static assets where the incessant updates are accessible. It will be settled at the hour of solicitation handling. It functions as a sub-daily practice; the Java servlet briefly passes the solicitation and reaction to the predetermined JSP/Servlet. Control is then returned back to the current JSP page. This tag is utilized for preparing a customer demand by a source JSP page by including other JSP pages and static assets like HTML's. One source JSP can incorporate the number of worker side assets lastly, we get the reaction of source JSP. This tag is given to incorporate the yield objective web asset program to yield source JSP program by rd.include(_,_) inside. It produces a separate JES class for the objective JSP page and remembers that yield for source JSP by utilizing rd.include(_,_). It performs yield incorporation.

Syntax: `<jsp:include page=" " flush=" "/>`

Attributes:

Page: It helps to include the relative URL of the page.

Flush: It is a Boolean attribute that determines whether the included resource has its buffer flushed before it is included.

JSP Custom Actions

In JSP innovation, by utilizing JSP mandates we can characterize present JSP page qualities, we can't utilize JSP orders to perform activities in the JSP pages. To perform activities in the event that we use scripting components, at that point we need to give Java code inside the JSP pages. The primary subject of JSP innovation isn't to permit Java code inside the JSP pages, to dispense with Java code from JSP pages we need to wipe out scripting components, we need to utilize custom activities. On the off chance that we need to take out scripting components from JSP pages, we need to utilize activities. Custom activities are JSP activities that could be set up by the designers according to their application prerequisites. In JSP innovation, standard activities will be spoken to as a bunch of predefined labels are called Action Tags. In JSP Technology, standard activities will be spoken to as a bunch of predefined labels are called Action labels. Additionally, all the custom activities will be spoken to as a bunch of clients characterized labels are called Custom Tags.

Syntax: `<prefix_Name : tag_Name>—-Body—-</prefix_Name>`

If we want to design custom tags in our JSP applications then we have to use the following 3 elements:

JSP page with Taglib directives: It is used to make available the TLD files into the present JSP page on the basis of custom tags prefix names.

TLD (Tag Library Descriptor) file: This tag provides the mapping between custom tag names and respective TagHandler classes.

TagHandler class: It is a normal Java class that is able to provide the basic functionality for the custom tags.

What are JSP Implicit Objects?

Implicit Objects are a set of Java objects which the JSP Container makes available to developers on each page. These objects can be accessed as built-in variables via scripting elements. It can also be accessed programmatically by JavaBeans and Servlets. These are created automatically for you within the service method.

In J2SE applications, for every java developer, it is a common requirement to display data on the command prompt. To perform this operation every time we have to prepare PrintStream object with command prompt location as target location:

```
Printstream ps = new PrintStream("C:\Program
files\…..\cmd.exe");

ps.println("Hello");
```

In Java applications, the PrintStream object is a frequent requirement so that java technology has provided that PrintStream object as a predefined object in the form of out variable in System class.

```
public static final PrintStream out;
```

Similarly, in web applications, all the web developers may require some objects like request, response, config, context, session, and so on are a frequent requirement, to get these objects we have to write some piece of java code. Once the above-specified objects are a frequent requirement in web applications, JSP technology has provided them as predefined support in the form of JSP Implicit Objects in order to reduce the burden on the developers.

Implicit objects in JSP need not be declared or instantiated by the JSP author. They are automatically instantiated by the container which is accessed by using standard variables. Hence, they are called implicit objects. These are parsed by the container and inserted into the generated servlet code. They are only available within the JSP service method and not in any declaration.

Types of Implicit Objects in JSP

For every JSP file when the JSP compiler generates the servlet program it will create the following 9 implicit objects inside the _jspService() method.

1. Request
2. Response
3. PageContext
4. Session
5. Application
6. Config
7. Out
8. Page
9. Exception

Hence all these objects are created and available inside the _jspService() method we can use all these implicit objects directly inside the JSP scriptlets.

Request Object in JSP

It is an instance of javax.servlet.http.HttpServletRequest object. This implicit object is used to process the request sent by the client. It is the HttpServletRequest object associated with the request. When a client requests a page the JSP engine creates a new object to represent that request.

Following are methods of request object:

1. getCookies(): It returns an array containing all of the cookie objects the client sent with this request.
2. getAttributeNames(): It returns an enumeration containing the names of the attributes available to this request.
3. getHeaderNames(): It returns an enumeration of all the header names it contains.
4. getParameterNames(): It returns a String object containing the names of the parameters contained in this request.
5. getSession(): It returns the current session associated with this request.
6. getLocale(): It returns the preferred Locale that the client will accept content in.

7. getInputStream(): It helps us to retrieve the body of the request as binary data using a ServletInputstream.
8. getAuthType(): It returns the name of the authentication scheme used to protect the servlet.
9. getContentType(): It returns the MIME type of the body of the request.
10. getContextPath(): It returns the portion of the request URI.
11. getMethod(): It returns the name of the HTTP method with which this request was made.
12. getPathInfo(): It returns extra path information associated with the URL.
13. getProtocol(): It returns the name and version of the protocol the request uses.
14. getQueryString(): It returns the query string that is contained in the requested URL.
15. getRequestedSessionId(): It returns the session ID specified by the client.
16. getServletPath(): It returns the part of the request's URL that calls the JSP.
17. isSecure(): It returns True or False indicating whether the request was made using a secure channel or not.
18. getContentLength(): It returns the length of the request body and made available by the input stream.
19. getServerPort(): It returns the port number on which the request was received.
20. getRemoteAddr(): It returns the IP address of the client that sent the request.
21.

Response Object in JSP:

This is the HttpServletResponse object associated with the response to the client. This implicit object is used to process the request and send the response to the client. We can send content-type as well as error reports and text. The object is by default available to scriptlets and expressions to get response-related information and to set response-related information. If a JSP wants to redirect a request from this server to another server then it can use the response object and sendRedirect method call.

Following are the methods of response object:

1. encodeRedirectURL(String url): It is used to encode the specified URL for use in the sendRedirect method.
2. containsHeader(String name): It returns True or False indicating whether the named response header has already been set.

3. isCommitted(): It returns True or False indicating if the response has been committed.
4. addCookie(): It is used to add the specified cookie to the response.
5. addHeader(String name, String value): It is used to add a response header with the given name and value.
6. flushBuffer(): It is used to force any content in the buffer to be written to the client.
7. reset(): It is used to clear the data that exists in the buffer as well as the status code and headers.
8. sendError(int sc): It is used to send an error response to the client using the specified status code and clearing the buffer.
9. sendRedirect(String location): It is used to send a temporary redirect response to the client using the specified redirect location URL.
10. setBufferSize(int size): It is used to set the preferred buffer size for the body of the response.
11. setContentType(int len): It is used to set the length of the content body on the response.
12. setContentType(String type): It is used to set the content type of the response being sent to the client.
13. setHeader(String name, String value): It is used to set a response header with the given name and value.
14. setLocale(Locale loc): It is used to set the locale of the response.
15. setStatus(int sc): It is used to set the status code for the response.

PageContext Implicit Object in JSP:

pageContext is an implicit object in JSP Technology, it can be used to get all the JSP implicit objects even in a Non-JSP environment. It is created for pageContext class which is used for creating all the implicit variables like a session, ServletConfig, out, etc… While preparing Tag handler classes in custom tags container will provide pageContext object as part of its life cycle, from this we are able to get all the implicit objects. In JSP applications, by using the pageContext object we are able to perform some operations with the attributes in the JSP scopes page, request, session, and application like adding an attribute, removing an attribute, getting an attribute, and finding an attribute. This encapsulates the use of server-specific features like higher performance JSPWriters. The pageContext reference points to pageContext class sub-class object.

Following are the methods of pageContext object:

1. setAttribute(String name, Object value, int scope): It is used to set an attribute onto a particular scope. Where scope may be page, request, session, or application.
2. getAttribute(String name): It is used to get an attribute from page scope.
3. getAttribute(String name, int scope): If we want to get an attribute value from the specified scope we have to use this method.
4. removeAttribute(String name, int scope): It is used to remove an attribute from a particular scope.
5. findAttribute(String name): To find an attribute value from page scope, request scope, session scope, and application scope we have to use this method.

Session Object in JSP:

This is the HttpSession object associated with the request. a session is an implicit object which is created for HttpSession class using which we can store and access data. The session reference point HttpSession implementation object. By default, this reference is available to scriptlet, expression tags. These tags can access session related info and they can manage session scope attributes. By using this reference these tags can delete session objects by calling session.invalidate() and these reference tags can set session timeout.

Following are the methods of the session object:

1. getAttribute(String name): It returns the object bound with the specified name in this session.
2. getAttributeNames(): It returns an Enumeration of String objects containing the names of all the objects bound to this session.
3. getCreationTime(): It returns the time when this session was created.
4. getId(): It returns a string containing the unique identifier assigned to this session.
5. getLastAccessedTime(): It returns the last time the client sent a request associated with this session.

6. getMaxInactiveInterval(): It returns the maximum time interval that the servlet container will keep this session open between client accesses.
7. invalidate(): It invalidates the session and unbinds any objects bound to it.
8. isNew(): It returns True if the client does not know about the session or if the client chooses not to join the session.
9. removeAttribute(String name): It removes the object bound with the specified name from this session.
10. setAttribute(String name, Object value): It binds an object to this session using the specified name.
11. setMaxInactiveInterval(int interval): It specifies the time between client requests before the servlet container will invalidate this session.

Application Object in JSP

This is the ServletContext object associated with the application context. Application is an implicit object which is created for ServletContext class and used to access the data of the ServletContaxt object same as in servlets. The application reference point ServletContext implementation object. By using this reference, a JSP can access application-related information such as managing application scope attributes, getting the application init parameters, etc.

Following are the methods of application object:

1. setAttribute(String Key, Object Value): This method is used to set a hit counter variable and to reset the same variable.
2. getAttribute(String Key): This method is used to read the current value of the hit counter and increase it by one and again set it for future use whenever a user accesses your page.

Implicit Config Object in JSP:

This is the ServletContext object associated with the page. This implicit object is created for ServletConfig class and used to access the data of the ServletConfig object same as in servlets. The config reference points ServletConfig implementation object. By using this config reference a JSP page can get initialization parameters of JSP configuration, get initialization parameters names, get servlet names, and can get servlet context reference.

Following is the only method of config object:

1. getServletName(): This method returns the servlet name which is the string contained in the <servlet-name> element defined in the web.xml configuration file.
2. getServletContext(): It returns a reference to the Servlet Context.

Out Object in JSP:

This is the PrintWriter object used to send output to the client. The out-reference points JSPWriter subclass object. By using the object, we can print HTML tags and plain text data on the browser.

Methods of out object

1. out.print(dataType dt): It is used to print a data type value.
2. out.println(dataType dt): It is used to print a data type value then terminate the line with a new line character.
3. out.flush(): It is used to flush the stream.

What is a Session?

A session is a time duration; it will start from the starting point of the client conversation with the server and will terminate at the ending point of the client's conversation with the server. The data which we transferred from client to server through multiple numbers of requests during a particular session then that data is called State of the Session.

In general, in web applications, a container will prepare a request object similarly to represent a particular user we have to prepare a separate session. In this context, to keep track of all the session objects at the server machine we need to set explicit mechanisms called Session Tracking Mechanisms.

Session Management in Java

Session Management is used to recognize the particular user. It is a way to maintain the state (data) of the user about a series of requests from the same user (that is, requests originating from the same browser) across the same period of time. Each time the user requests to the server, the server always treats the request as the new request.

What is Session Tracking?

In our applications sometimes we need to carry or transfer the data between multiple forms. But in form-based applications, we are using HTTP protocol which is a stateless protocol which means it remembers only the current conversation. For example, if we leave the first form and enter it into the second form, we will forget about the previous form data. But if we want to overcome this limitation and carry the data between multiple forms, we can use any one of the following methodologies:

1. Hidden variables
2. Cookies
3. Sessions
4. A session with URL re-writing or encoding URL

Need for Session Tracking in JSP

Whenever a client makes a request, then for each request a new connection is established to the webserver because HTTP is a stateless protocol. It is very difficult for servers to identify that the requests are coming from the same user or not because every time a new connection is established. In this case, the server does not keep track of the information of the previous request if every time a new request is made.

To overcome this problem, for every request a unique id should be possessed which helps to maintain a complete conversation between the client and server.

Session Tracking Techniques

Hidden Variables

If we want to use variables, we have to use hidden fields given by HTML. Hidden fields must be specified as a part of <form> and we must specify name and value attributes. Hidden fields not displayed to the client.

Syntax: `<input type="hidden" name="xxx" value="yyy"/>`

Advantages:

1. Hidden variables consume less memory.
2. Once we close the browser the data of hidden variables will be cleared automatically.

Disadvantages:

1. If we want to carry a huge amount of data then it will be complicated using hidden variables.
2. Hidden variables are not recommended to carry any sensitive data.

Cookies

Cookies are also used to transfer the data between multiple form-based applications. Simply a cookie is a small piece of information sent by the server to any client. Every cookie will store the data in the form of key-value pair. The server will add the cookie to the response header by using the Set-Cookie header and send it to the client. Cookies will be stored in the browser by using the domain name of the website. The browser can contain cookies from multiple domains. The client needs to use HeaderCookie to get all the cookies available in the browser.

Syntax: `Cookie myCookie = new Cookie(‶SessionID″, ‶SessionValue″)`

Advantage:

1. Compared to hidden variables cookies are better to transfer the data because once we store the cookie, we can get the data on any page of the website directly.

Disadvantage:

1. If the client disables the cookies in the browser we can work with cookies.
2. For security reasons, it is not recommended to use cookies.
3. If we want to a huge amount of data between server and client it will be complicated.

Session Object in JSP

Creating a session object means creating an object for a class that is implementing javax.servlet.HttpSession interface. The session object is also used to transfer the data between multiple form-based applications. In the case of servlets, we have to write the code for creating the session object. But in JSP by default session object is enabled whenever the client sends the request to the server internally. So, we can write the JSP program directly to get Session information. In the JSP session is an implicit object. But if we want to handle session object in JSP explicitly then we have to write the following page re-directive in the JSP program:

```
<%@ page session=″false|true″%>
```

Advantage: This technique is more reliable as each user has its own unique sessionID which is helpful to access the session data stored at the server-side.

How to get Session Object?

When the client sends the request to the server for the first time then the server creates a session object and now the server creates a unique ID that is associated with this session object. Then the server will create a cookie with the name JSESSIONID which holds the session object as a value and send it to the client. If we want to request the server to create a session object, we have to use the following methods of request object:

HttpSession getSession(true)

HttpSession getSession(false)

Session objects are used to store the data. The session can support to stores of any kind of data. To work with the session object and to store the data or to get the data we have to use some set of methods.

Methods of Session Object

1. getAttribute(String name): It gives the object bound with the specified name in this session.
2. getAttributeNames(): It gives an Enumeration of String objects containing the names of all the objects bound to this session.
3. getCreationTime(): It returns the time when this session was created.
4. getId(): It returns a string containing the unique identifier assigned to this session.
5. getLastAccessedTime(): It returns the last time the client sent a request associated with this session.
6. getMaxInactiveInterval(): It returns the maximum time interval that the servlet container will keep this session open between client accesses.
7. invalidate(): It invalidates the session and unbinds any objects bound to it.
8. isNew(): It returns true if the client does not yet know about the session or if the client chooses not to join the session.
9. removeAttribute(String name): It removes the object bound with the specified name from this session.
10. setAttribute(String name, Object value): It binds an object to this session using the specified name.

11. setMaxInactiveInterval(int interval): It specifies the time between client requests before the servlet container will invalidate this session.

When session object is deleted?

1. When we close the server.
2. When we call session.invalidate() in our program generally session.invalidate() is called as a part of logout pages.
3. When we call session.setMaxInactiveInterval(int) in our program.
4. By changing the session timeout in web.xml of the server.