MODULE II DYNAMIC WEB PAGES AND JAVA SERVLETS 9

Overview of HTML5 – CSS- Servlets- Introduction; Servlet Lifecycle; sessions;session tracking using hidden fields, user authentication, URL rewriting andcookies; Inter servlet Communication; Java Server Pages (JSP) – Introduction toJSP tags; JSP life cycle; Directives; Custom JSP tags; Java Server Facestechnology – Introduction to page navigation; tags, life cycle and architecture.

Overview of HTML5:

HTML5 is the next major revision of the HTML standard superseding HTML 4.01, XHTML 1.0, and XHTML 1.1. HTML5 is a standard for structuring and presenting content on the World Wide Web.

HTML5 is a cooperation between the World Wide Web Consortium (W3C) and the Web Hypertext Application Technology Working Group (WHATWG).

The new standard incorporates features like video playback and drag-and-drop that have been previously dependent on third-party browser plug-ins such as Adobe Flash, Microsoft Silverlight, and Google Gears.

Browser Support

The latest versions of Apple Safari, Google Chrome, Mozilla Firefox, and Opera all support many HTML5 features and Internet Explorer 9.0 will also have support for some HTML5 functionality.

The mobile web browsers that come pre-installed on iPhones, iPads, and Android phones all have excellent support for HTML5.

New Features

HTML5 introduces a number of new elements and attributes that can help you in building modern websites. Here is a set of some of the most prominent features introduced in HTML5.

- New Semantic Elements − These are like <header>, <footer>, and <section>.
- Forms 2.0 − Improvements to HTML web forms where new attributes have been introduced for <input> tag.
- Persistent Local Storage − To achieve without resorting to third-party plugins.
- WebSocket − A next-generation bidirectional communication technology for web applications.
- Server-Sent Events − HTML5 introduces events which flow from web server to the web browsers and they are called Server-Sent Events (SSE).
- Canvas − This supports a two-dimensional drawing surface that you can program with JavaScript.

- Audio & Video − You can embed audio or video on your webpages without resorting to third-party plugins.
- Geolocation − Now visitors can choose to share their physical location with your web application.
- Microdata − This lets you create your own vocabularies beyond HTML5 and extend your web pages with custom semantics.
- Drag and drop − Drag and drop the items from one location to another location on the same webpage.

Backward Compatibility

HTML5 is designed, as much as possible, to be backward compatible with existing web browsers. Its new features have been built on existing features and allow you to provide fallback content for older browsers.

New Added Elements in HTML 5:

- <article>: The <article> tag is used to represent an article. More specifically, the content within the <article> tag is independent from the other content of the site (even though it can be related).
- <aside>: The <aside> tag is used to describe the main object of the web page in a shorter way like a highlighter. It basically identifies the content that is related to the primary content of the web page but does not constitute the main intent of the primary page. The <aside> tag contains mainly author information, links, related content and so on.
- <figcaption>: The <figcaption> tag in HTML is used to set a caption to the figure element in a document.
- <figure>: The <figure> tag in HTML is used to add self-contained content like illustrations, diagrams, photos or codes listing in a document. It is related to main flow, but it can be used in any position of a document and the figure goes with the flow of the document and if it is removed it should not affect the flow of the document.
- <header>: It contains the section heading as well as other content, such as a navigation links, table of contents, etc.
- <footer>: The <footer> tag in HTML is used to define a footer of HTML document. This section contains the footer information (author information, copyright information, carriers etc.). The footer tag is used within body tag. The <footer> tag is new in the HTML 5. The footer elements require a start tag as well as an end tag.
- <main>: Delineates the main content of the body of a document or web app.
- <mark>: The <mark> tag in HTML is used to define the marked text. It is used to highlight the part of the text in the paragraph.

- <nav>: The <nav> tag is used to declaring the navigational section in HTML documents. Websites typically have sections dedicated to navigational links, which enables user to navigate the site. These links can be placed inside a nav tag.
- <section>: It demarcates a thematic grouping of content.
- <details>: The <details> tag is used for the content/information which is initially hidden but could be displayed if the user wishes to see it. This tag is used to create interactive widget which user can open or close it. The content of details tag is visible when open the set attributes.
- <summary>: The <summary> tag in HTML is used to define a summary for the <details> element. The <summary> element is used along with the <details> element and provides a summary visible to the user. When the summary is clicked by the user, the content placed inside the <details> element becomes visible which was previously hidden. The <summary> tag was added in HTML 5. The <summary> tag requires both starting and ending tag.
- <time>: The <time> tag is used to display the human-readable data/time. It can also be used to encode dates and times in a machine-readable form. The main advantage for users is that they can offer to add birthday reminders or scheduled events in their calendars and search engines can produce smarter search results.
- <bdi>: The <bdi> tag refers to the Bi-Directional Isolation. It differentiates a text from other text that may be formatted in different direction. This tag is used when a user generated text with an unknown direction.
- <wbr>: The <wbr> tag in HTML stands for word break opportunity and is used to define the position within the text which is treated as a line break by the browser. It is mostly used when the used word is too long and there are chances that the browser may break lines at the wrong place for fitting the text.
- <datalist>: The <datalist> tag is used to provide autocomplete feature in the HTML files. It can be used with input tag, so that users can easily fill the data in the forms using select the data.
- <keygen>: The <keygen> tag in HTML is used to specify a key-pair generator field in a form. The purpose of <keygen> element is to provide a secure way to authenticate users. When a form is submitted then two keys are generated, private key and public key. The private key stored locally, and the public key is sent to the server. The public key is used to generate client certificate to authenticate user in future.
- <output>: The <output> tag in HTML is used to represent the result of a calculation performed by the client-side script such as JavaScript.
- <progress>: It is used to represent the progress of a task. It also defines how much work is done and how much is left to download a task. It is not used to represent the disk space or relevant query.

- <svg>: It is the Scalable Vector Graphics.
- <canvas>: The <canvas> tag in HTML is used to draw graphics on web page using JavaScript. It can be used to draw paths, boxes, texts, gradient and adding images. By default, it does not contain border and text.
- <audio>: It defines the music or audio content.
- <embed>: Defines containers for external applications (usually a video player).
- <source>: It defines the sources for <video> and <audio>.
- <track>: It defines the tracks for <video> and <audio>.
- <video>: It defines the video content.

Advantages:

- All browsers supported.
- More device friendly.
- Easy to use and implement.
- HTML 5 in integration with CSS, JavaScript, etc. can help build beautiful websites.

Disadvantages:

- Long codes have to be written which is time consuming.
- Only modern browsers support it.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Simple HTML5 Program</title>
</head>
<body>
<h1>Hello, World!</h1>
</body>
</html>
```

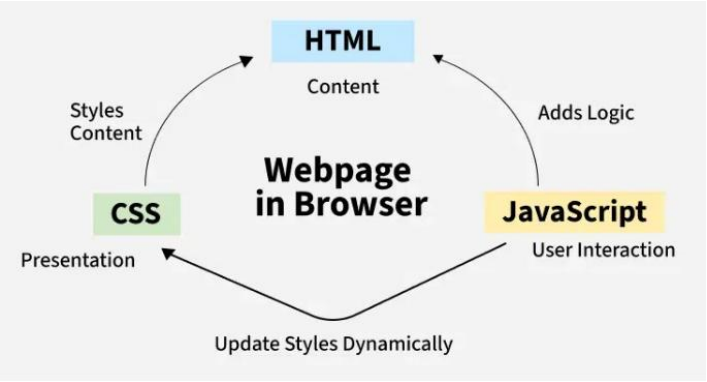
Output:

Hello, World!

CSS

CSS stands for Cascading Style Sheets. It is a stylesheet language used to style and enhance website presentation.

- CSS is one of the main three components of a webpage along with HTML and JavaScript.
  - HTML adds Structure to a web page.
  - JavaScript adds logic to it and CSS makes it visually appealing or stylish. It controls the layout of a web page i.e. how HTML elements will be displayed on a webpage.
- CSS was released (in 1996), 3 years after HTML (in 1993). The main idea behind its use is, it allows the separation of content (HTML) from presentation (CSS). This makes websites easier to maintain and more flexible.



Role of CSS In Webpage

There are three different ways to add CSS styles to an HTML document:

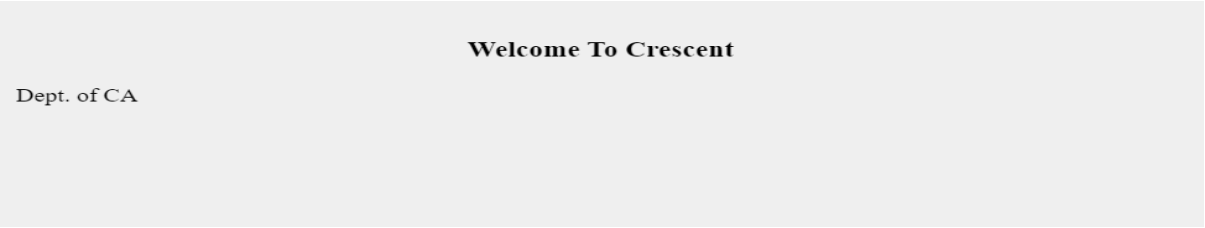
1. Inline CSS

Use the style attribute on the HTML element to add styles to the web page. It is used for small projects.

```
<!-- File name: index.html -->
<html>
<body>
<!-- Using Inline CSS -->
<h3 style="text-align: center;">
        Welcome To Crescent
        </h3>

<p>Dept. of CA</p>
</body>
</html>
```

Output:



2. Internal CSS

Place the CSS styles within a <style> tag inside the HTML file, usually inside the <head> section.

```
<!-- File name: index.html -->
<html>
<head>
        <!-- Using Internal CSS -->
<style>
      h3 {
         color: green;
      }
</style>
</head>
<body>
<!-- CSS is applied here -->
<h3>Welcome To Crescent</h3>

<p>Dept. of CA</p>
</body>
</html>
```

Output:

Welcome To Crescent

Dept. of CA

3. External CSS

Create a separate CSS file with a .css extension and link this file to your HTML file using the <link> tag. It consider the best practice to add CSS into HTML File.

```
<!-- File name: index.html -->
<html>
<head>
        <!-- Importing External CSS -->
<link rel="stylesheet" href="style.css" />
</head>
<body>
<p>Hai Shanthi</p>
</body>
</html>
```

```
 /* Write CSS Here *//* External CSS */

/* File name: style.css */

p {
```

```
        text-align: center;
}
```

Output:

Hai Shanthi

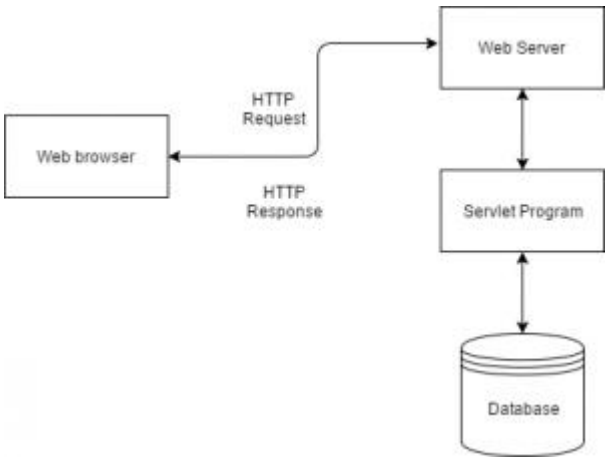
Servlets-Introduction

- Servlets are the Java programs that run on the Java-enabled web server or application server.
- They are used to handle the request obtained from the webserver, process the request, produce the response, then send a response back to the webserver.

Properties of Servlets are as follows:

- Servlets work on the server-side.
- Servlets are capable of handling complex requests obtained from the webserver.

Servlet Architecture



Execution of Servlets basically involves six basic steps:

- The clients send the request to the webserver.
- The web server receives the request.
- The web server passes the request to the corresponding servlet.

- The servlet processes the request and generates the response in the form of output.
- The servlet sends the response back to the webserver.
- The web server sends the response back to the client and the client browser displays it on the screen.

Advantages of Servlet

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

Servlets API's:

Servlets are build from two packages:

- javax.servlet(Basic)
- javax.servlet.http(Advance)

Various classes and interfaces present in these packages are:

Component	Type	Package
Servlet	Interface	javax.servlet.*
ServletRequest	Interface	javax.servlet.*
ServletResponse	Interface	javax.servlet.*
GenericServlet	Class	javax.servlet.*
HttpServlet	Class	javax.servlet.http.*
HttpServletRequest	Interface	javax.servlet.http.*
HttpServletResponse	Interface	javax.servlet.http.*
Filter	Interface	javax.servlet.*
ServletConfig	Interface	javax.servlet.*

The Servlet Container

Servlet container, also known as Servlet engine is an integrated set of objects that provide a run time environment for Java Servlet components.

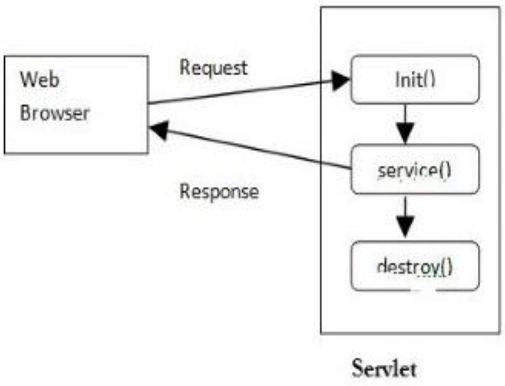
Services provided by the Servlet container:

1. Network Services: Loads a Servlet class. The loading may be from a local file system, a remote file system or other network services. The Servlet container provides the network services over which the request and response are sent.
2. Decode and Encode MIME-based messages: Provides the service of decoding and encoding MIME-based messages.
3. Manage Servlet container: Manages the lifecycle of a Servlet.
4. Resource management Manages the static and dynamic resources, such as HTML files, Servlets, and JSP pages.
5. Security Service: Handles authorization and authentication of resource access.
6. Session Management: Maintains a session by appending a session ID to the URL path.

Servlet Life Cycle

The web container maintains the life cycle of a servlet instance. The life cycle of the servlet is as follows:

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) Servlet class is loaded

The classloader is responsible to load the servlet class. The servlet class is loaded when the first request for the servlet is received by the web container.

2) Servlet instance is created

The web container creates the instance of a servlet after loading the servlet class. The servlet instance is created only once in the servlet life cycle.

3) init method is invoked

The web container calls the init method only once after creating the servlet instance. The init method is used to initialize the servlet. It is the life cycle method of the javax.servlet.Servlet interface. Syntax of the init method is given below:

public void init(ServletConfig config) throws ServletException

4) service method is invoked

The web container calls the service method each time when request for the servlet is received. If servlet is not initialized, it follows the first three steps as described above then calls the service method. If servlet is initialized, it calls the service method. Notice that servlet is initialized only once. The syntax of the service method of the Servlet interface is given below:

public void service(ServletRequest request, ServletResponse response)

5) destroy method is invoked

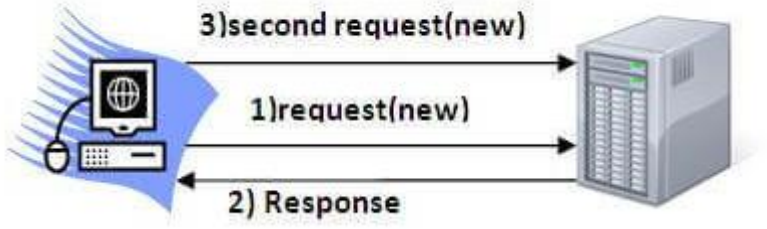
The web container calls the destroy method before removing the servlet instance from the service. It gives the servlet an opportunity to clean up any resource for example memory, thread etc. The syntax of the destroy method of the Servlet interface is given below:

public void destroy()

Session Tracking in Servlets

- **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.



Session Tracking Techniques

The techniques used in Session tracking:

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

1. Hidden Form Field

Hidden form field can also be used to store session information for a particular client. In case of hidden form field a hidden field is used to store client state. In this case user information is stored in hidden field value and retrieved from another servlet.

Example: <input type="hidden" name="uname" value="SSS>

Advantage of Hidden Form Field

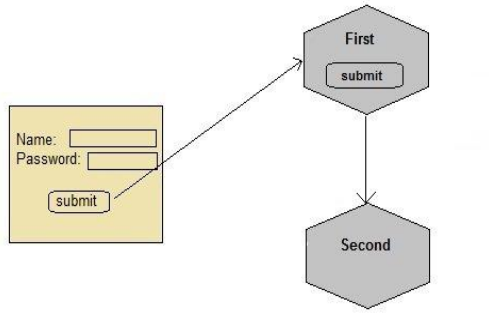
- It will always work whether cookie is disabled or not.

Disadvantage of Hidden Form Field:

- It is maintained at server side.
- Extra form submission is required on each pages.
- Only textual information can be used.

Example of using Hidden Form Field

In this example, we are storing the name of the user in a hidden textfield and getting that value from another servlet.



hidden field in **First** Servlet, populated the value of user, and sent it to the **Second** Servlet, now Second servlet also has the user information. Similarly we will have to keep sending this information, wherever we need this, using hidden fields.

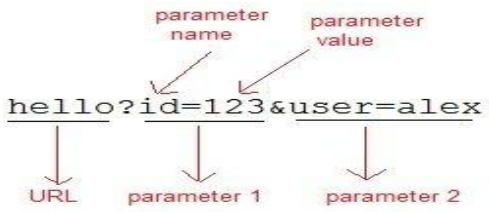
2. User Authentication

- **Session Management**: After successful authentication (e.g., via username/password), a session ID or a token is created and stored on the server-side.

- **Session IDs in Cookies**: The server sends a session ID to the client, typically via a cookie. This ID is then used to identify the user for subsequent requests.
- **Session Timeout**: Implementing a timeout for sessions is important for security. After a certain period of inactivity, the session should expire.
- **Security Considerations**: Ensure to use secure cookies (set with the Secure and HttpOnly flags) to prevent session hijacking.

3. URL Rewriting

if the client has disabled cookies in the browser then session management using cookie wont work. In that case **URL Rewriting** can be used as a backup. **URL rewriting** will always work. In URL rewriting, a token(parameter) is added at the end of the URL. The token consist of name/value pair separated by an equal(=) sign.



When the User clicks on the URL having parameters, the request goes to the **Web Container** with extra bit of information at the end of URL. The **Web Container** will fetch the extra part of the requested URL and use it for session management.

The getParameter() method is used to get the parameter value at the server side.

Advantage of URL Rewriting

- It will always work whether cookie is disabled or not (browser independent).
- Extra form submission is not required on each pages.

Disadvantage of URL Rewriting

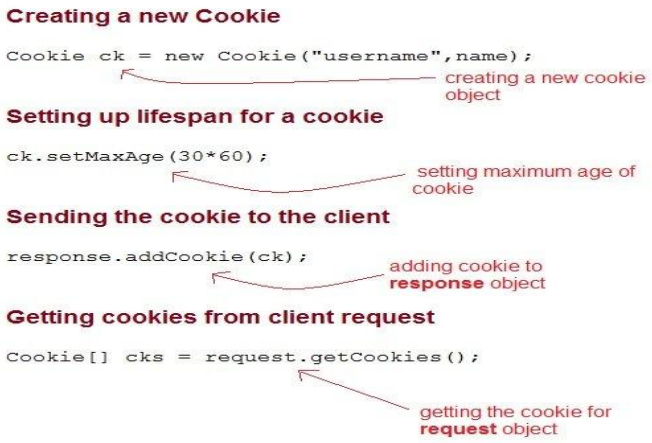
- It will work only with links.
- It can send Only textual information.

4. Cookies in Servlet

Cookies are small pieces of information that are sent in response from the web server to the client. **Cookies** are the simplest technique used for storing client state.

Cookies are stored on client's computer. They have a lifespan and are destroyed by the client browser at the end of that lifespan.

Cookies are created using **Cookie** class present in Servlet API. Cookies are added to **response** object using the addCookie() method. This method sends cookie information over the HTTP response stream. getCookies() method is used to access the cookies that are added to response object.



Types of Cookies

There are two types of cookies. They are as following:

- Session
- Persistent

1) Session cookies:

The session cookies do not have any expiration time. It is present in the browser memory. When the web browser is closed then the cookies are destroyed automatically.

2) Persistent Cookies:

The Persistent cookies have an expiration time. It is stored in the hard drive of the user and it is destroyed based on the expiry time.

Advantage of Cookies

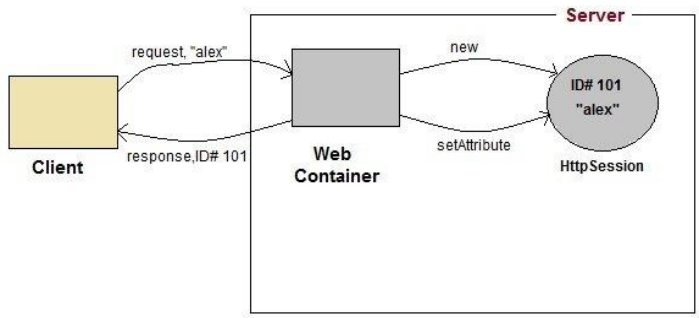
- Simplest technique of maintaining the state.
- Cookies are maintained at client side.

Disadvantage of Cookies

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

5. HttpSession

HttpSession object is used to store entire session with a specific client. We can store, retrieve and remove attribute from **HttpSession** object. Any servlet can have access to **HttpSession** object throughout the getSession() method of the **HttpServletRequest** object.

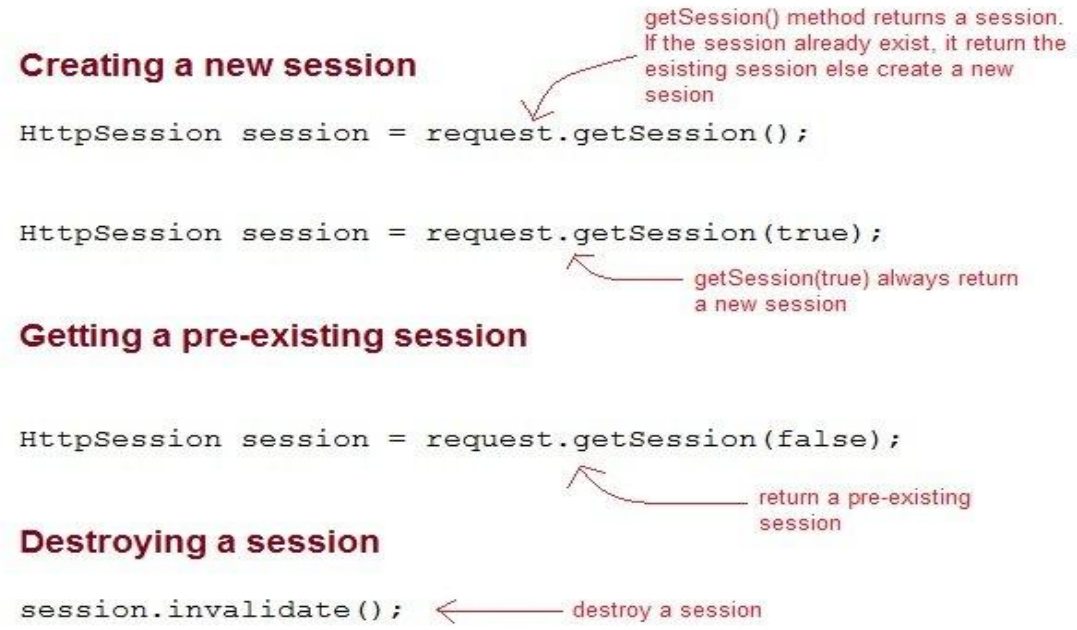


1. On client's first request, the **Web Container** generates a unique session ID and gives it back to the client with response. This is a temporary session created by web container.
2. The client sends back the session ID with each request. Making it easier for the web container to identify where the request is coming from.
3. The **Web Container** uses this ID, finds the matching session with the ID and associates the session with the request.

Some Important Methods of Servlet HttpSession

Methods	Description

long getCreationTime()	returns the time when the session was created, measured in milliseconds since midnight January 1, 1970 GMT.
String getId()	returns a string containing the unique identifier assigned to the session.
long getLastAccessedTime()	returns the last time the client sent a request associated with the session
int getMaxInactiveInterval()	returns the maximum time interval, in seconds.
void invalidate()	destroy the session
boolean isNew()	returns true if the session is new else false
void setMaxInactiveInterval(int interval)	Specifies the time, in seconds,after servlet container will invalidate the session.



Inter-Servlet Communication

Inter Servlet Communication is a communication between servlets. Servlets talking to each other. There are many ways to communicate between servlets, including

- Request Dispatching
- HTTP Redirect
- Servlet Chaining
- HTTP request (using sockets or the URLConnection class)
- Shared session, request, or application objects (beans)
- Direct method invocation (deprecated)

- Shared static or instance variables (deprecated)

Basically, interServlet communication is achieved through servlet chaining, which is a process in which you pass the output of one servlet as the input to another. These servlets should be running in the same server.

e.g.
ServletContext.getRequestDispatcher(HttpRequest,HttpResponse).forward("NextServlet") ;
You can pass in the current request and response object from the latest form submission to the next servlet/JSP. You can modify these objects and pass them so that the next servlet/JSP can use the results of this servlet.

There are some Servlet engine-specific configurations for servlet chaining.

Servlets can also call public functions of other servlets running in the same server. This can be done by obtaining a handle to the desired servlet through the ServletContext Object by passing it the servlet name (this object can return any servlets running in the server). And then calling the function on the returned Servlet object.

e.g. TestServlet test=
(TestServlet)getServletConfig().getServletContext().getServlet("OtherServlet");
otherServletDetails= Test.getServletDetails();

You must be careful when you call another servlet's methods. If the servlet that you want to call implements the SingleThreadModel interface, your call could conflict with the servlet's single-threaded nature. (The server cannot intervene and make sure your call happens when the servlet is not interacting with another client.) In this case, your servlet should make an HTTP request to the other servlet instead of direct calls.

Servlets could also invoke other servlets programmatically by sending an HTTP request. This could be done by opening a URL connection to the desired Servlet.

Java Server Pages (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

1) Extension to Servlet

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.soft Joins Growing List of Major Companies Pulling Out of Russia

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.

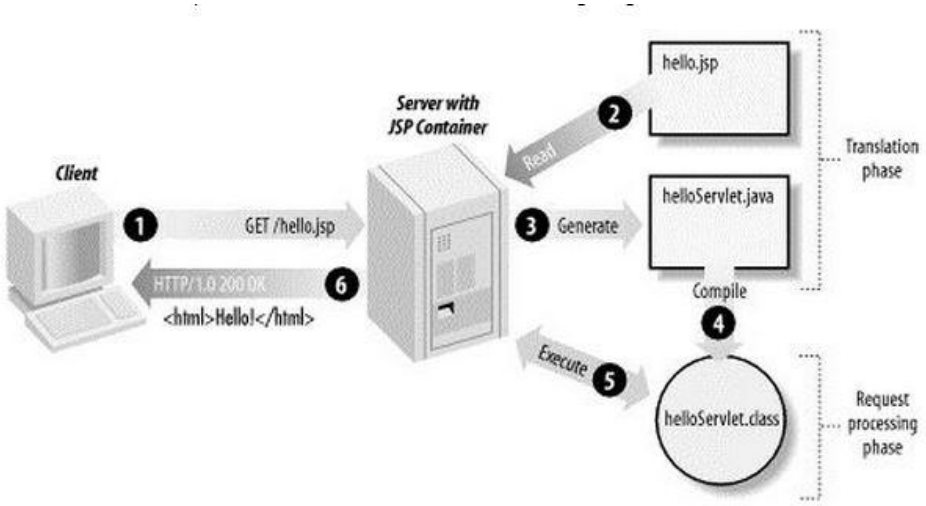
JSP Processing

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

- 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.



JSP Tags/Elements

JSP Tag	Brief Description	Tag Syntax
Directive	Specifies translation time instructions to the JSP engine.	<%@ directives %>
Declaration	Declaration Declares and defines methods and variables.	<%! variable declaration & method definition %>

Scriptlet	Allows the developer to write free-form Java code in a JSP page.	<% some Java code %>
Expression	Used as a shortcut to print values in the output HTML of a JSP page.	<%= an Expression %>
Action	Provides request-time instructions to the JSP engine.	<jsp:actionName />
Comment	Used for documentation and for commenting out parts of JSP code.	<%– any Text –%>

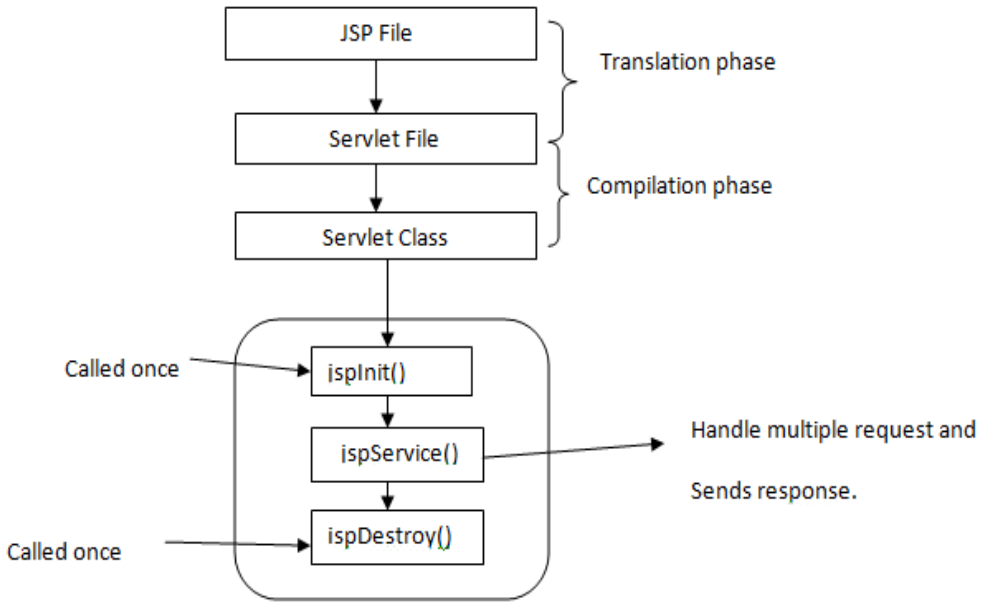
JSP Life Cycle

A Java Server Page life cycle is defined as the process that started with its creation which later translated to a servlet and afterward servlet lifecycle comes into play. This is how the process goes on until its destruction.

Following steps are involved in the JSP life cycle:

- Translation of JSP page to Servlet
- Compilation of JSP page(Compilation of JSP into test.java)
- Classloading (test.java to test.class)
- Instantiation(Object of the generated Servlet is created)
- Initialization(jspInit() method is invoked by the container)
- Request processing(_jspService()is invoked by the container)
- JSP Cleanup (jspDestroy() method is invoked by the container)

JSP Life Cycle



- **Translation of JSP page to Servlet :**

This is the first step of the JSP life cycle. This translation phase deals with the Syntactic correctness of JSP. Here test.jsp file is translated to test.java.

- **Compilation of JSP page :**

Here the generated java servlet file (test.java) is compiled to a class file (test.class).

- **Classloading :**

Servlet class which has been loaded from the JSP source is now loaded into the container.

- **Instantiation :**

Here an instance of the class is generated. The container manages one or more instances by providing responses to requests.

- **Initialization :**

jspInit() method is called only once during the life cycle immediately after the generation of Servlet instance from JSP.

- **Request processing :**

_jspService() method is used to serve the raised requests by JSP. It takes request and response objects as parameters. This method cannot be overridden.

➢ **JSP Cleanup :**

In order to remove the JSP from the use by the container or to destroy the method for servlets jspDestroy()method is used. This method is called once, if you need to perform any cleanup task like closing open files, releasing database connections jspDestroy() can be overridden.

JSP Directives

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

There are three types of directives:

- page directive
- include directive
- taglib directive

JSP page directive

language	Which language the file uses.	<%@ page language = "java" %>
extends	Superclass used by the JSP engine for the translated Servlet.	<%@ page extends = "com.taglib..." %>
import	Import all the classes in a java package into the current JSP page. This allows the JSP page to use other java classes. The following packages are implicitly imported. java.lang.* javax.servlet.* javax.servlet.jsp.* javax.servlet.http.*	<%@ page import = "java.util.*" %>
session	Does the page make use of sessions. By default all JSP pages have session data available. There are performance benefits to switching session to false.	Default is set to true.
buffer	Controls the use of buffered output for a JSP page. Default is 8kb	<%@ page buffer = "none" %>
autoFlush	Flush output buffer when full.	<%@ page autoFlush = "true" %>
isThreadSafe	Can the generated Servlet deal with multiple requests? If true a new thread is started so requests are handled simultaneously.	
info	Developer uses info attribute to add information/document for a page. Typically used to add author, version, copyright and date info.	<%@ page info = "CS@SIUCS Department. Programming Distributed Apps. " %>
errorPage	Different page to deal with errors. Must be URL to error page.	<%@ page errorPage = "/error/error.jsp" %>
IsErrorPage	This flag is set to true to make a JSP page a special Error Page. This page has access to the implicit object exception (see later).	
contentType	Set the mime type and character set of the JSP.	

Jsp Include Directive

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

Advantage of Include directive: Code Reusability

Syntax: <%@ include file="resourceName" %>

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: *<%@ taglib uri="uri of the tag library" prefix="prefix of tag library" %>*

CUSTOM JSP Tags

Custom tags are user-defined tags. They eliminates the possibility of scriptlet tag and separates the business logic from the JSP page. The same business logic can be used many times by the use of custom tag.

Advantages of Custom Tags

1. Eliminates the need of scriptlet tag
2. Separation of business logic from JSP
3. Re-usability

Syntax to use custom tag

There are two ways to use the custom tag. They are given below:

<prefix:tagname attr1=value1....attrn=valuen />

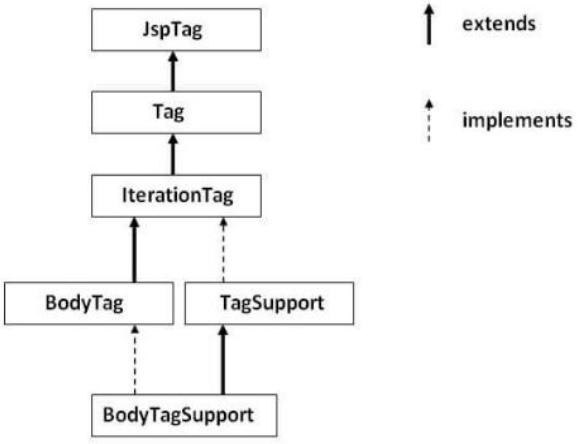
<prefix:tagname attr1=value1....attrn=valuen >

body code

</prefix:tagname>

JSP Custom Tag API

The javax.servlet.jsp.tagext package contains classes and interfaces for JSP custom tag API. The JspTag is the root interface in the Custom Tag hierarchy.



JspTag interface: The JspTag is the root interface for all the interfaces and classes used in custom tag. It is a marker interface.

Tag interface: The Tag interface is the sub interface of JspTag interface. It provides methods to perform action at the start and end of the tag.

Fields of Tag interface

There are four fields defined in the Tag interface. They are:

Field Name	Description
public static int EVAL_BODY_INCLUDE	it evaluates the body content.
public static int EVAL_PAGE	it evaluates the JSP page content after the custom tag.
public static int SKIP_BODY	it skips the body content of the tag.
public static int SKIP_PAGE	it skips the JSP page content after the custom tag.

The methods of the Tag interface are as follows:

Method Name	Description
public void setPageContext(PageContext pc)	it sets the given PageContext object.
public void setParent(Tag t)	it sets the parent of the tag handler.
public Tag getParent()	it returns the parent tag handler object.
public int doStartTag()throws JspException	it is invoked by the JSP page implementation object. The JSP programmer should override this method and define the business logic to be performed at the start of the tag.
public int doEndTag()throws JspException	it is invoked by the JSP page implementation object. The JSP programmer should override this method and define the business logic to be performed at the end of the tag.
public void release()	it is invoked by the JSP page implementation object to release

	the state.

IterationTag interface: The IterationTag interface is the sub interface of the Tag interface. It provides an additional method to reevaluate the body.

Field of IterationTag interface

There is only one field defined in the IterationTag interface.

- **public static int EVAL_BODY_AGAIN** it reevaluates the body content.

Method of Tag interface

There is only one method defined in the IterationTag interface.

- **public int doAfterBody()throws JspException** it is invoked by the JSP page implementation object after the evaluation of the body. If this method returns EVAL_BODY_INCLUDE, body content will be reevaluated, if it returns SKIP_BODY, no more body cotent will be evaluated.

TagSupport class: The TagSupport class implements the IterationTag interface. It acts as the base class for new Tag Handlers. It provides some additional methods also.

Java Server Faces

It is a server side component based user interface framework. It is used to develop web applications. It provides a well-defined programming model and consists of rich API and tag libraries. The latest version JSF 2 uses Facelets as its default templating system. It is written in Java.

The JSF API provides components (inputText, commandButton etc) and helps to manage their states. It also provides server-side validation, data conversion, defining page navigation, provides extensibility, supports for internationalization, accessibility etc.

The JSF Tag libraries are used to add components on the web pages and connect components with objects on the server. It also contains tag handlers that implements the component tag.

Benefits of JavaServer Faces

1) It provides clean and clear separation between behavior and presentation of web application. You can write business logic and user interface separately.

2) JavaServer Faces API?s are layered directly on top of the Servlet API. Which enables several various application use cases, such as using different presentation technologies, creating your own custom components directly from the component classes.

3) Including of Facelets technology in JavaServer Faces 2.0, provides massive advantages to it. Facelets is now the preferred presentation technology for building JavaServer Faces based web applications.

JSF Features

Latest version of JSF 2.2 provides the following features.

- Component Based Framework
- Implements Facelets Technology
- Integration with Expression Language
- Support HTML5
- Ease and Rapid web Development.
- Support Internationalization
- Bean Annotations
- Default Exception Handling
- Templating
- Inbuilt AJAX Support
- Security

Component Based Framework

JSF is a server side component-based framework. It provides inbuilt components to build web application. You can use HTML5, Facelets tags to create web pages.

Facelets Technology

Facelets is an open source Web template system. It is a default view handler technology for JavaServer Faces (JSF). The language requires valid input XML documents to work. Facelets supports all of the JSF UI components and focuses completely on building the view for a JSF application.

Expression Language

Expression Language provides an important mechanism for creating the user interface (web pages) to communicate with the application logic (managed beans). The EL represents a union of the expression languages offered by JavaServer Faces technology.

HTML 5

HTML5 is the new standard for writing web pages. JavaServer Faces version 2.2 offers an easy way for including new attributes of HTML 5 to JSF components and provides HTML5 friendly markup.

Ease and Rapid web Development.

JSF provides rich set of inbuilt tools and libraries so that you can easily and rapidly develop we application.

Support Internationalization

JSF supports internationalization for creating World Class web application. You can create applications in the different-different languages. With the help of JSF you can make the application adaptable to various languages and regions.

Bean Annotations

JSF provides annotations facility in which you can perform validation related tasks in Managed Bean. It is good because you can validate your data in bean rather than in HTML validation.

Exception Handling

JSF provide default Exception handling so you can develop exception and bug free web application.

Templating

Introducing template in new version of JSF provides reusablity of components. In JSF application, you can create new template, reuse template and treat it as component for application.

AJAX Support

JSF provides inbuilt AJAX support. So, you can render application request to server side without refreshing the web page. JSF also support partial rendering by using AJAX.

Security

JSF provides implicit protection against this when state is saved on the server and no stateless views are used, since a post-back must then contain a valid javax.faces.ViewState hidden parameter. Contrary to earlier versions, this value seems sufficiently random in modern JSF implementations. Note that stateless views and saving state on the client does not have this implicit protection.

JSF - Page Navigation

Navigation rules are those rules provided by JSF Framework that describes which view is to be shown when a button or a link is clicked.

Navigation rules can be defined in JSF configuration file named faces-config.xml. They can be defined in managed beans.

Navigation rules can contain conditions based on which the resulted view can be shown. JSF 2.0 provides implicit navigation as well in which there is no need to define navigation rules as such.

Implicit Navigation

JSF 2.0 provides **auto view page resolver** mechanism named **implicit navigation**. In this case, you only need to put view name in action attribute and JSF will search the correct **view** page automatically in the deployed application.

Auto Navigation in JSF Page

Set view name in action attribute of any JSF UI Component.

```
<h:form>
<h3>Using JSF outcome</h3>
<h:commandButtonaction="page2"value="Page2"/>
</h:form>
```

Here, when **Page2** button is clicked, JSF will resolve the view name, **page2** as page2.xhtml extension, and find the corresponding view file **page2.xhtml** in the current directory.

Auto Navigation in Managed Bean

Define a method in managed bean to return a view name.

```
@ManagedBean(name ="navigationController", eager =true)
@RequestScoped

publicclassNavigationControllerimplementsSerializable{
publicString moveToPage1(){
return"page1";
}
}
```

Get view name in action attribute of any JSF UI Component using managed bean.

```
<h:form>
<h3> Using Managed Bean</h3>
<h:commandButton action = "#{navigationController.moveToPage1}"
  value = "Page1" /glt;
</h:form>
```

Here, when **Page1** button is clicked, JSF will resolve the view name, **page1** as page1.xhtml extension, and find the corresponding view file **page1.xhtml** in the current directory.

Conditional Navigation

Using managed bean, we can very easily control the navigation. Look at the following code in a managed bean.

Resolving Navigation Based on from-action

JSF provides navigation resolution option even if managed bean different methods returns the same view name.

Forward vs Redirect

JSF by default performs a server page forward while navigating to another page and the URL of the application does not change.

To enable the page redirection, append **faces-redirect=true** at the end of the view name.

JSF - Basic Tags

JSF provides a standard HTML tag library. For these tags you need to use the following namespaces of URI in html node.

<html
xmlns = "http://www.w3.org/1999/xhtml"
xmlns:h = "http://java.sun.com/jsf/html">

S.No	Tag & Description
1	h:inputText Renders a HTML input of type="text", text box.
2	h:inputSecret Renders a HTML input of type="password", text box.
3	h:inputTextarea Renders a HTML textarea field.
4	h:inputHidden Renders a HTML input of type="hidden".
5	h:selectBooleanCheckbox Renders a single HTML check box.
6	h:selectManyCheckbox Renders a group of HTML check boxes.
7	h:selectOneRadio

	Renders a single HTML radio button.
8	h:selectOneListbox Renders a HTML single list box.
9	h:selectManyListbox Renders a HTML multiple list box.
10	h:selectOneMenu Renders a HTML combo box.
11	h:outputText Renders a HTML text.
12	h:outputFormat Renders a HTML text. It accepts parameters.
13	h:graphicImage Renders an image.
14	h:outputStylesheet Includes a CSS style sheet in HTML output.
15	h:outputScript Includes a script in HTML output.
16	h:commandButton Renders a HTML input of type="submit" button.
17	h:Link Renders a HTML anchor.
18	h:commandLink Renders a HTML anchor.
19	h:outputLink Renders a HTML anchor.
20	h:panelGrid Renders an HTML Table in form of grid.
21	h:message Renders message for a JSF UI Component.
22	h:messages Renders all message for JSF UI Components.
23	f:param Pass parameters to JSF UI Component.

24	f:attribute Pass attribute to a JSF UI Component.
25	f:setPropertyActionListener Sets value of a managed bean's property

JSF - Facelets Tags

JSF provides special tags to create common layout for a web application called facelets tags.

These tags provide flexibility to manage common parts of multiple pages at one place.

For these tags, you need to use the following namespaces of URI in html node.

<html

xmlns = "http://www.w3.org/1999/xhtml"

xmlns:ui = "http://java.sun.com/jsf/facelets">

Following are important Facelets Tags in JSF 2.0.

S.No	Tag & Description
1	Templates We'll demonstrate how to use templates using the following tags • <ui:insert> • <ui:define> • <ui:include> • <ui:composition>
2	Parameters We'll demonstrate how to pass parameters to a template file using the following tag • <ui:param>
3	Custom We'll demonstrate how to create custom tags
4	Remove We'll demonstrate capability to remove JSF code from generated HTML page

JSF - Convertor Tags

JSF provides inbuilt convertors to convert its UI component's data to object used in a managed bean and vice versa. For example, these tags can convert a text into date object and can validate the format of input as well.

For these tags, you need to use the following namespaces of URI in html node.

```
<html
xmlns="http://www.w3.org/1999/xhtml"
```

```
xmlns:f="http://java.sun.com/jsf/core">
```

Following are important *Convertor Tags* in JSF 2.0 −

S.No	Tag & Description
1	f:convertNumber Converts a String into a Number of desired format
2	f:convertDateTime Converts a String into a Date of desired format
3	Custom Convertor Creating a custom convertor

JSF - Validator Tags

JSF provides inbuilt validators to validate its UI components. These tags can validate the length of the field, the type of input which can be a custom object. For these tags you need to use the following namespaces of URI in html node.

<html

xmlns = "http://www.w3.org/1999/xhtml"

xmlns:f = "http://java.sun.com/jsf/core">

Following are important Validator Tags in JSF 2.0−

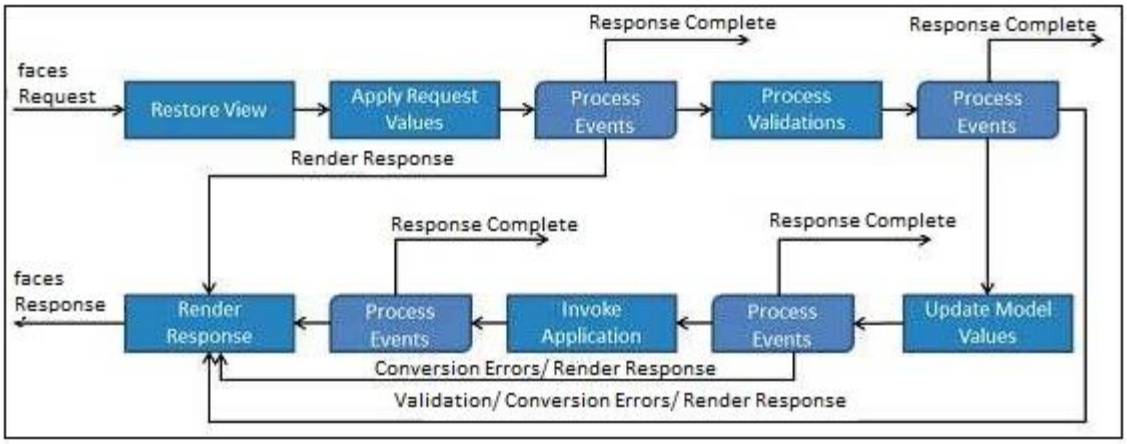
S.No	Tag & Description
1	f:validateLength Validates the length of a string
2	f:validateLongRange Validates the range of a numeric value
3	f:validateDoubleRange Validates the range of a float value
4	f:validateRegex Validates JSF component with a given regular expression
5	Custom Validator Creates a custom validator

JSF - Life Cycle

JSF application life cycle consists of six phases which are as follows −

- Restore view phase
- Apply request values phase; process events

- Process validations phase; process events
- Update model values phase; process events
- Invoke application phase; process events
- Render response phase



Phase 1: Restore view

JSF begins the restore view phase as soon as a link or a button is clicked and JSF receives a request.

During this phase, JSF builds the view, wires event handlers and validators to UI components and saves the view in the FacesContext instance. The FacesContext instance will now contain all the information required to process a request.

Phase 2: Apply request values

After the component tree is created/restored, each component in the component tree uses the decode method to extract its new value from the request parameters. Component stores this value. If the conversion fails, an error message is generated and queued on FacesContext. This message will be displayed during the render response phase, along with any validation errors.

If any decode methods event listeners called renderResponse on the current FacesContext instance, the JSF moves to the render response phase.

Phase 3: Process validation

During this phase, JSF processes all validators registered on the component tree. It examines the component attribute rules for the validation and compares these rules to the local value stored for the component.

If the local value is invalid, JSF adds an error message to the FacesContext instance, and the life cycle advances to the render response phase and displays the same page again with the error message.

Phase 4: Update model values

After the JSF checks that the data is valid, it walks over the component tree and sets the corresponding server-side object properties to the components' local values. JSF will update the bean properties corresponding to the input component's value attribute.

If any updateModels methods called renderResponse on the current FacesContext instance, JSF moves to the render response phase.

Phase 5: Invoke application

During this phase, JSF handles any application-level events, such as submitting a form/linking to another page.

Phase 6: Render response

During this phase, JSF asks container/application server to render the page if the application is using JSP pages. For initial request, the components represented on the page will be added to the component tree as JSP container executes the page. If this is not an initial request, the component tree is already built so components need not be added again. In either case, the components will render themselves as the JSP container/Application server traverses the tags in the page.

After the content of the view is rendered, the response state is saved so that subsequent requests can access it and it is available to the restore view phase.

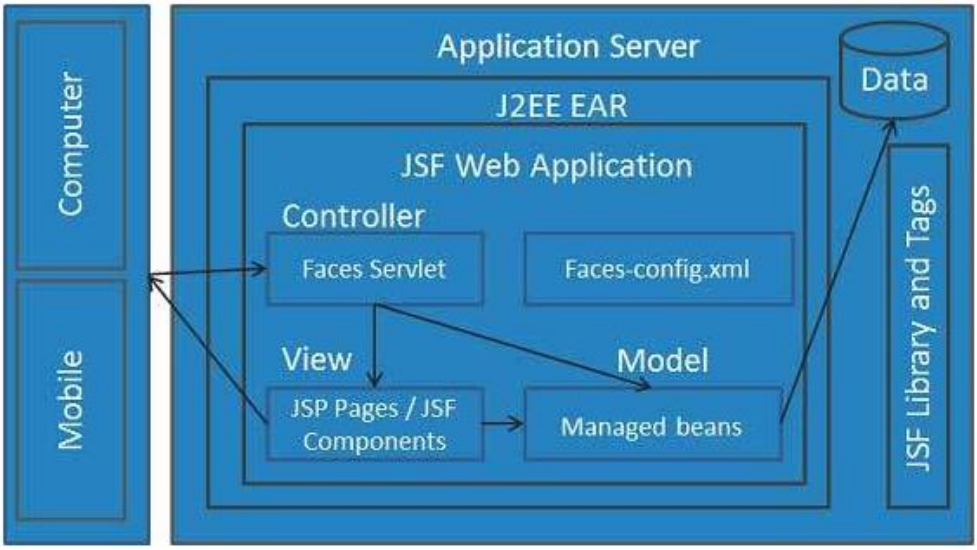
JSF - Architecture

JSF technology is a framework for developing, building server-side User Interface Components and using them in a web application. JSF technology is based on the Model View Controller (MVC) architecture for separating logic from presentation.

JSF application is similar to any other Java technology-based web application; it runs in a Java servlet container, and contains −

- JavaBeans components as models containing application-specific functionality and data
- A custom tag library for representing event handlers and validators
- A custom tag library for rendering UI components
- UI components represented as stateful objects on the server

- Server-side helper classes
- Validators, event handlers, and navigation handlers
- Application configuration resource file for configuring application resources



There are controllers which can be used to perform user actions. UI can be created by web page authors and the business logic can be utilized by managed beans.

JSF provides several mechanisms for rendering an individual component. It is upto the web page designer to pick the desired representation, and the application developer doesn't need to know which mechanism was used to render a JSF UI component.