**JSP (JavaServer Pages):**

Purpose:

- JSP is a technology used to create dynamic web pages with Java code embedded in HTML-like syntax.

- It allows developers to write Java code directly within the HTML markup, making it easier to generate dynamic content.

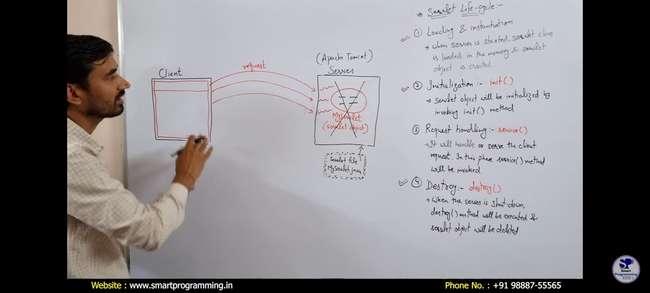
1) Define tag handlers by implementing the javax.servlet.jsp.tagext.Tag interface or extending its subclasses.

2) Declare tag libraries in the TLD (Tag Library Descriptor) file and map them to tag handler classes.

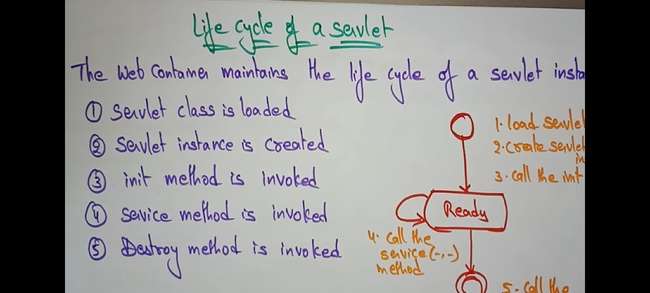
3) Use the custom tags in JSP pages by referencing the tag libraries and specifying the tag names and attributes

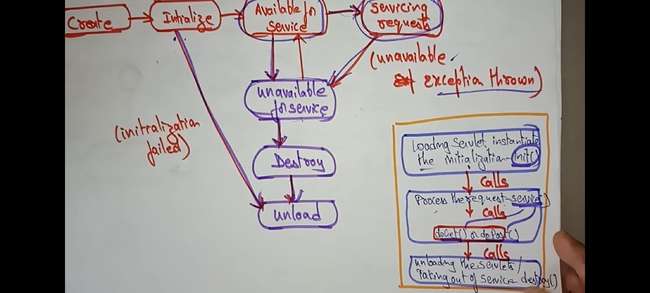
**Life cycle of Servlet**

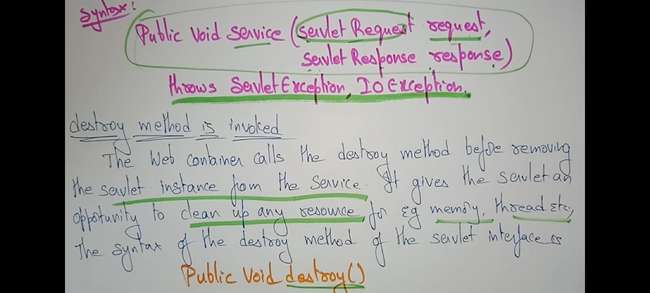
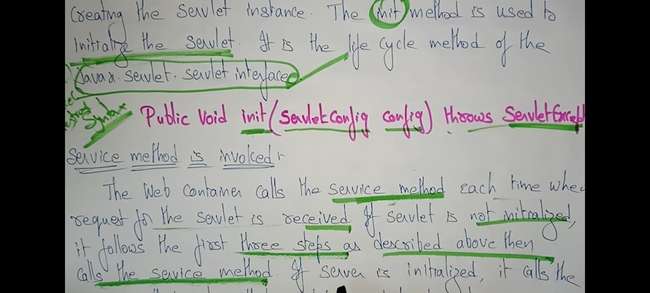
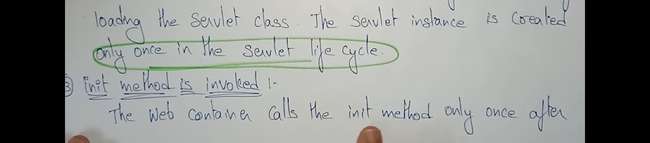
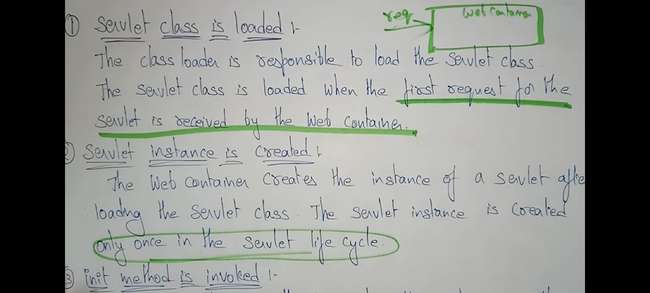
1) [https://youtu.be/6RhdX-d\_pbY?si=eE7psP54hwoOHqgS](https://youtu.be/6RhdX-d_pbY?si=eE7psP54hwoOHqgS" \o "https://youtu.be/6RhdX-d_pbY?si=eE7psP54hwoOHqgS)



2) [https://youtu.be/SpOWu3NHFIw?si=AetsYPvLkEvMVZ7t](https://youtu.be/SpOWu3NHFIw?si=AetsYPvLkEvMVZ7t" \o "https://youtu.be/SpOWu3NHFIw?si=AetsYPvLkEvMVZ7t)

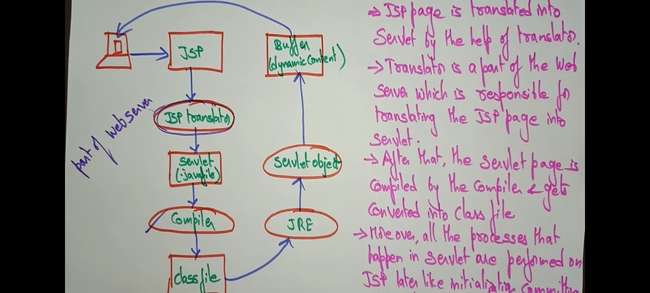
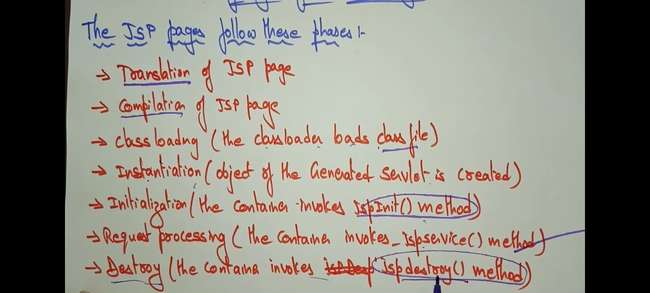






**JSP Life Cycle**

[https://youtu.be/KR5xHmgF1fo?si=cmfwyETrgfnagpqb](https://youtu.be/KR5xHmgF1fo?si=cmfwyETrgfnagpqb" \o "https://youtu.be/KR5xHmgF1fo?si=cmfwyETrgfnagpqb)



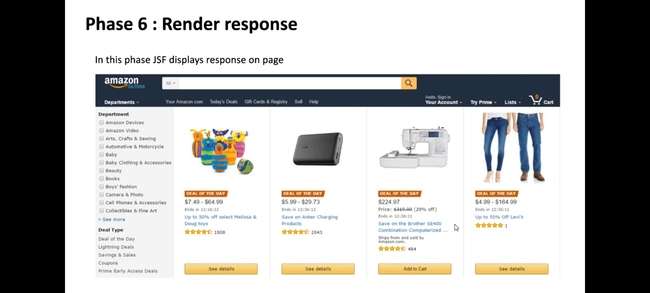
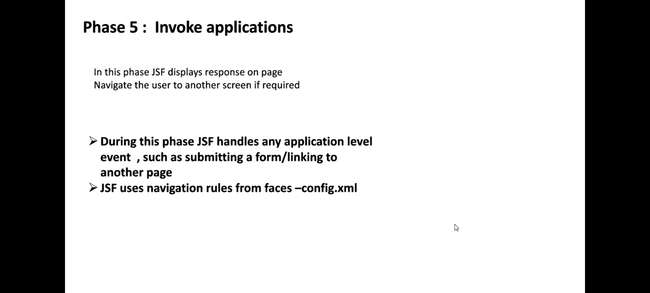
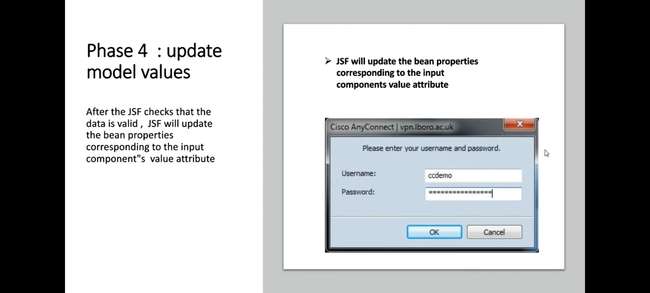
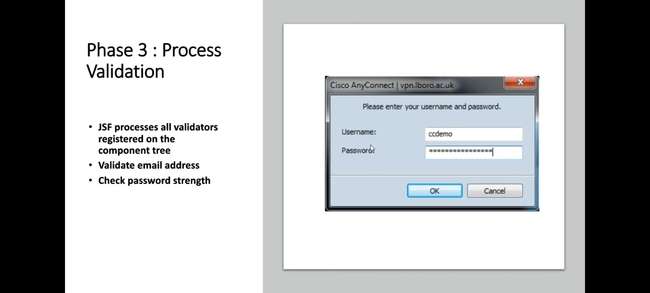
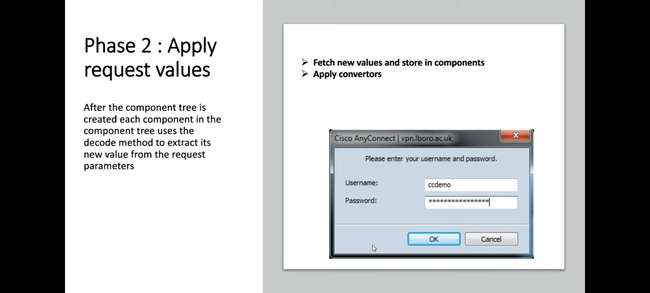
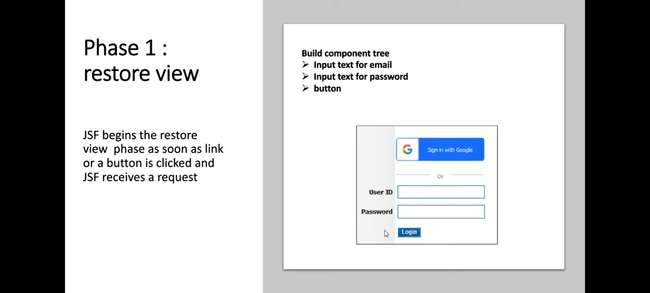
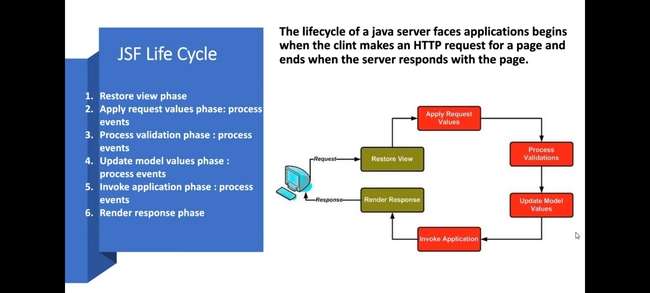
**Cookie in Servlet**

[https://youtu.be/LILHZhIV9sM?si=MObdPdUm0Jxuw2SM](https://youtu.be/LILHZhIV9sM?si=MObdPdUm0Jxuw2SM" \o "https://youtu.be/LILHZhIV9sM?si=MObdPdUm0Jxuw2SM)

**JSF Lifecycle**

[https://youtu.be/mZ83Idd\_jFc?si=exl2sMoBGFJ4v8Mm](https://youtu.be/mZ83Idd_jFc?si=exl2sMoBGFJ4v8Mm" \o "https://youtu.be/mZ83Idd_jFc?si=exl2sMoBGFJ4v8Mm)





**Servlet (bhamstra jwd)**

**Difference between CGI and Servlet**

**What is CGI?**

[https://www.boardinfinity.com/blog/understanding-servlets-in-java/amp/](https://www.boardinfinity.com/blog/understanding-servlets-in-java/amp/" \o "https://www.boardinfinity.com/blog/understanding-servlets-in-java/amp/) . (All que in one link)





આજે આપણે બધા ડાયનેમિક વેબ પેજીસ બનાવવાની જરૂરિયાતથી વાકેફ છીએ એટલે કે જેઓ સમય અનુસાર સાઇટની સામગ્રી બદલવાની ક્ષમતા ધરાવે છે અથવા ક્લાયન્ટ દ્વારા મળેલી વિનંતી અનુસાર સામગ્રીઓ જનરેટ કરવામાં સક્ષમ છે.

જો તમને જાવામાં કોડિંગ ગમે છે, તો તમને એ જાણીને આનંદ થશે કે જાવાનો ઉપયોગ કરીને ડાયનેમિક વેબ પેજીસ જનરેટ કરવાની એક રીત પણ અસ્તિત્વમાં છે અને તે રીત છે Java Servlet.

સર્વલેટ્સ એ જાવા પ્રોગ્રામ્સ છે જે જાવા વેબ સર્વર અથવા એપ્લિકેશન સર્વર પર ચાલે છે.

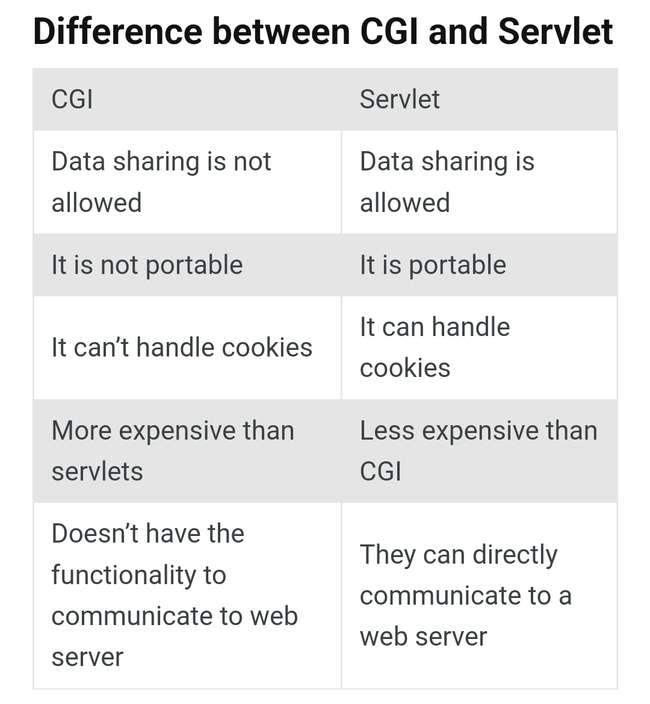
તેઓનો ઉપયોગ વેબસર્વરમાંથી મેળવેલી વિનંતીને હેન્ડલ કરવા, વિનંતી પર પ્રક્રિયા કરવા, પ્રતિભાવ ઉત્પન્ન કરવા, પછી વેબસર્વરને પ્રતિસાદ મોકલવા માટે થાય છે.

સર્વલેટ્સના ગુણધર્મો નીચે મુજબ છે:

* સર્વલેટ્સ સર્વર બાજુ પર કામ કરે છે.
* સર્વલેટ્સ વેબસર્વરમાંથી મેળવેલ જટિલ વિનંતીઓને હેન્ડલ કરવામાં સક્ષમ છે.
* Execution of Servlets basically involves six basic steps:

1. The clients send the request to the webserver.
2. The web server receives the request.
3. The web server passes the request to the corresponding servlet.
4. The servlet processes the request and generates the response in the form of output.
5. The servlet sends the response back to the webserver.
6. The web server sends the response back to the client and the client browser displays it on the screen.

**Servlet over CGI**

Advantages of a Java Servlet

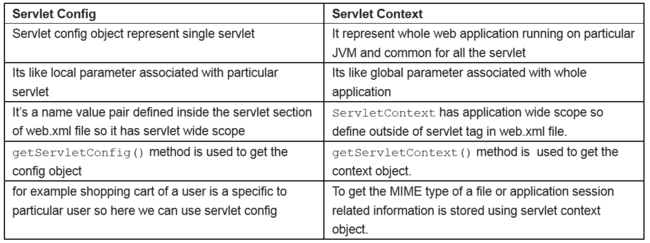
1. In Servlet, new processes are not created. Therefore, it is faster than CGI.

2. Servlets generally are platform-independent.

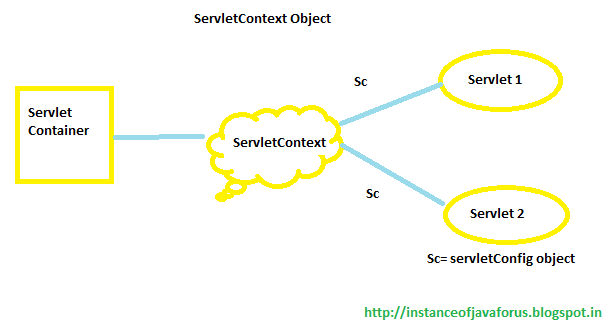
3. These are the server-side component. Therefore, the servlet inherits the security provided by the Web server.

**Servlet Context**

[https://pin.it/4LVTvTSUr](https://pin.it/4LVTvTSUr" \o "https://pin.it/4LVTvTSUr)



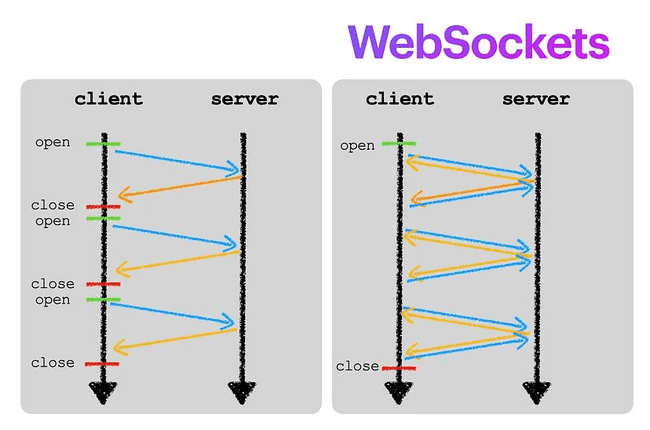
* Each web application contains its own resources in a separate environment.
* This environment called ad Web Application context or ServletContext.
* A Servlet context contains zero or more number of servlets.
* ServletContext is an interface. this interface implemented by the container by the container provider.
* The implemented class allows all servlets in the web application to communicate with the container.
* We can store common data into this application and also we can share that data even after request response objects deletion in the memory.



**Web Socket**

[https://medium.com/swlh/how-to-build-a-websocket-applications-using-java-486b3e394139](https://medium.com/swlh/how-to-build-a-websocket-applications-using-java-486b3e394139" \o "https://medium.com/swlh/how-to-build-a-websocket-applications-using-java-486b3e394139) .

[https://www.javatpoint.com/what-is-websocket](https://www.javatpoint.com/what-is-websocket" \o "https://www.javatpoint.com/what-is-websocket)

* WebSocket is a thin, lightweight layer above TCP.
* Websocket allows creating a communication channel between a client and a server.
* In particular, a communication channel that uses the WebSocket protocol as a communication protocol.
* The WebSocket protocol is compatible with the HTTP protocol
* 

Traditional HTTP Communication vs Web Socket

|  |  |
| --- | --- |
| Traditional HTTP Communication | Web Socket |
| 1) client connect server  2) client send request for resources  3) server response  4) close communication channel | 1) keep connection open  2) both client - server can make req - res |

* Web sockets are defined as a two-way communication between the servers and the clients,

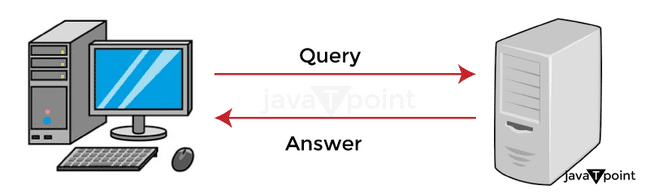
which mean both the parties, communicate and exchange data at the same time.

* It allows two-way data transfer in which we can simultaneously transfer data from client to server and vice versa.
* The client could be any web browser, and

the server could be any backend system.

* WebSocket API allow two-way data flow simultaneously, which provides a quick way to transfer the data.
* Thus, WebSocket allows a web application to communicate with the WebSocket server without interruption to provide real-time data.
* The below image explains how WebSocket transfers the data:
* 

**How does it work?**

* વેબસોકેટ કેવી રીતે કાર્ય કરે છે તે સમજવા માટે, સૌ પ્રથમ, આપણે એ સમજવાની જરૂર છે કે વેબસાઇટ HTTP પ્રોટોકોલ પર કેવી રીતે કાર્ય કરે છે અને વેબસોકેટનો ઉપયોગ કર્યા વિના વેબસાઇટ ડેટાને ઍક્સેસ કરે છે.
* સામાન્ય રીતે, વેબ પેજીસ HTTP પ્રોટોકોલ પર HTTP કનેક્શન બનાવીને સર્વ કરવામાં આવે છે. અહીં, ક્લાયંટની વિનંતી મુજબ HTTP પ્રોટોકોલ દ્વારા ડેટા આપવામાં આવે છે.
* દર વખતે જ્યારે ક્લાયંટ સર્વરને વિનંતી કરે છે, ત્યારે દરેક ક્રિયા માટે ચોક્કસ પ્રતિસાદ હોય છે, અને સર્વર તે મુજબ પ્રતિભાવ મોકલે છે.
* 
* વેબસોકેટ પ્રોટોકોલ HTTP પ્રોટોકોલ કરતા અલગ રીતે કામ કરે છે.
* તે ડાયનેમિક કોલ-અપ પદ્ધતિનો ઉપયોગ કરીને રીઅલ-ટાઇમમાં ડેટા ટ્રાન્સફર કરી શકે છે.
* અમારે ફક્ત વેબસોકેટ પ્રોટોકોલનો ઉપયોગ કરીને ક્લાયંટથી સર્વર સાથે કનેક્શન સ્થાપિત કરવાની જરૂર છે. વેબસોકેટ પ્રોટોકોલ ક્લાયંટને હેન્ડશેક ટ્રાન્સમિટ કરે છે.
* તેમાં જરૂરી ડેટા ટ્રાન્સમિશન માહિતીને ઓળખવા માટે તમામ જરૂરી માહિતી શામેલ છે.
* એકવાર કનેક્શન સ્થાપિત થઈ જાય, ચેનલ ખુલ્લી હોય છે, અને હેન્ડશેક પછી તે ખુલ્લી રહે છે જેથી સતત સંચાર થઈ શકે. આમ, સર્વર વિનંતી કર્યા વિના ક્લાયંટને ડેટા મોકલી શકે છે.
* આથી, જ્યારે પણ સર્વર નવો ડેટા મેળવે છે, ત્યારે તે તેને વધુ વિનંતી કર્યા વગર તે જ ચેનલ પરના ક્લાયન્ટને ટ્રાન્સફર કરે છે.

**Custom Tags in JSP**

[https://www.javatpoint.com/custom-tags](https://www.javatpoint.com/custom-tags" \o "https://www.javatpoint.com/custom-tags)

Overview + advantage:

* Custom tags are user-defined tags.
* They eliminates the possibility of scriptlet tag
* separates the business logic from the JSP page.
* The same business logic can be used many times by the use of custom tag.
* Syntax to use custom tag
  1. <prefix:tagname attr1=value1....attrn=valuen />
  2. <prefix:tagname attr1=value1....attrn=valuen >

body code

</prefix:tagname>

* The following three components are required to develop custom tags :
  1. Tag Handler
  2. TLD(Tag Library Descriptor) file
  3. Taglib directive in jsp file

1) Tag handler

* It is a container-managed object created by a JSP container at runtime.
* A Java class is used to implement the processing logic for the tag.
* It may contain certain properties corresponding to the attributes or body of the tag.
* All tag handlers have a pageContext property for the JSP page where the tag is located

2) TLD

* Declare tag library in TLD file and map them to take handler class
* It is a file saved with a .tld extension

that contains a set of related tags

mapped to their respective tag handlers

* along with their description such as the name of the tag, attributes of the tag, etc

3) Taglib directive in JSP page

* Taglib directive is used to access a particular tag library within a JSP.
* It specifies the URI of the tag library and its prefix.
* Syntax:

<%@taglib uri=” ” prefix=” “%>

**Create a custom tag in JSP to extract the part of a given string.**

1. Define the Tag Handler Class:
   * Create Java class extend

java.servlet.jsp.tagext.SimpleTagSupporter

1. Declare Tag Attributes:
2. Extract Substring Logic:
   * Write logic inside 'doTag()'
   * Use 'substring()' to extract substring.
3. Register the Tag in the Tag Library Descriptor (TLD):
   * Create or update TLD file to declare custom tag
   * define tag name, class, attribute name
4. Use the Custom Tag in JSP:
   * Import custom tag library at top of your JSP page
   * use it by specifying prefix and attribute in JSP code.

**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:
  + 1. page directive
    2. include directive
    3. taglib directive
* Syntax of JSP Directive

<%@ directive attribute="value" %>

**JSP Page Directive**

* The page directive defines attributes that apply to an entire JSP page.
* Syntax of JSP page directive

<%@ page attribute="value" %>

* Attributes of JSP page directive
  + import
  + contentType
  + extends
  + info
  + buffer
  + language
  + isELIgnored
  + isThreadSafe
  + autoFlush
  + session
  + pageEncoding
  + errorPage
  + isErrorPage

**JSF customer validator**

[https://www.naukri.com/code360/library/jsf-custom-validator](https://www.naukri.com/code360/library/jsf-custom-validator" \o "https://www.naukri.com/code360/library/jsf-custom-validator)

* Creating a custom validator in JSP involves writing a Java class to perform validation logic and integrating it into your JSP page.
* Here's a simplified explanation:
  1. Create a Validator Class:
  2. Integrate Validator into JSP:
  3. Use Validator with Form Inputs:
  4. Handle Validation Errors:
  5. Test Validator Thoroughly:

Example:

Let's create a simple JSP page with a form that validates a user's email address:

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title>Email Validation</title>  <script>  function validateForm() {  var email = document.getElementById("email").value;  var emailPattern = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;  if (!emailPattern.test(email)) {  document.getElementById("emailError").innerHTML = "Invalid email address";  return false;  }  return true;  }  </script>  </head>  <body>  <h2>Email Validation</h2>  <form action="submit.jsp" method="post" onsubmit="return validateForm()">  Email: <input type="text" id="email" name="email">  <span id="emailError" style="color: red;"></span><br>  <input type="submit" value="Submit">  </form>  </body>  </html> |

* In this example:
* The JavaScript function validateForm() checks if the email input matches the specified pattern.
* If the email is invalid, an error message is displayed next to the input field.
* The form submission is prevented if the email is invalid.
* This is a basic example of client-side validation in JSP.

**JSP ACTION TAG**  (openai)

* In JSP, the <jsp:action> tag is used to include dynamic content or execute Java code within a JSP page.
* The <jsp:action> tag in JSP allows you to perform actions or execute Java code directly within your JSP page.
* It's like a mini-program that you can insert anywhere in your JSP code to achieve specific tasks.
* Key Points:

1. Dynamic Content:
2. Execution of Java Code:
3. Integration with JavaBeans:
4. Common Actions:

|  |
| --- |
| Example:  <jsp:include page="header.jsp" />  <jsp:action>  <%  String userName = "John";  out.println("Welcome, " + userName + "!");  %>  </jsp:action>  <jsp:forward page="footer.jsp" /> |

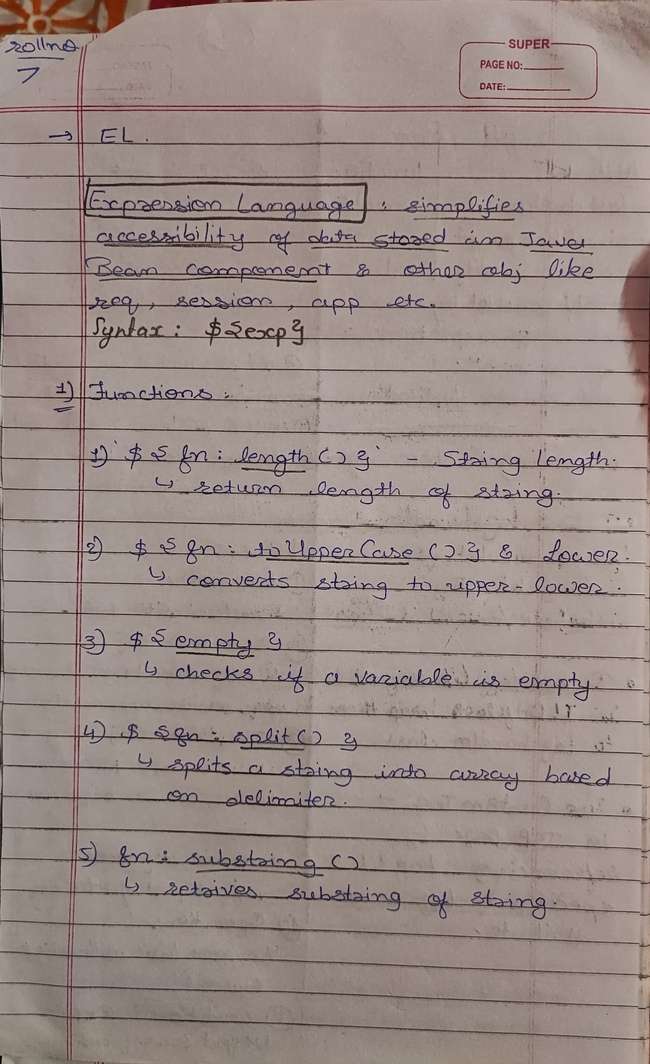
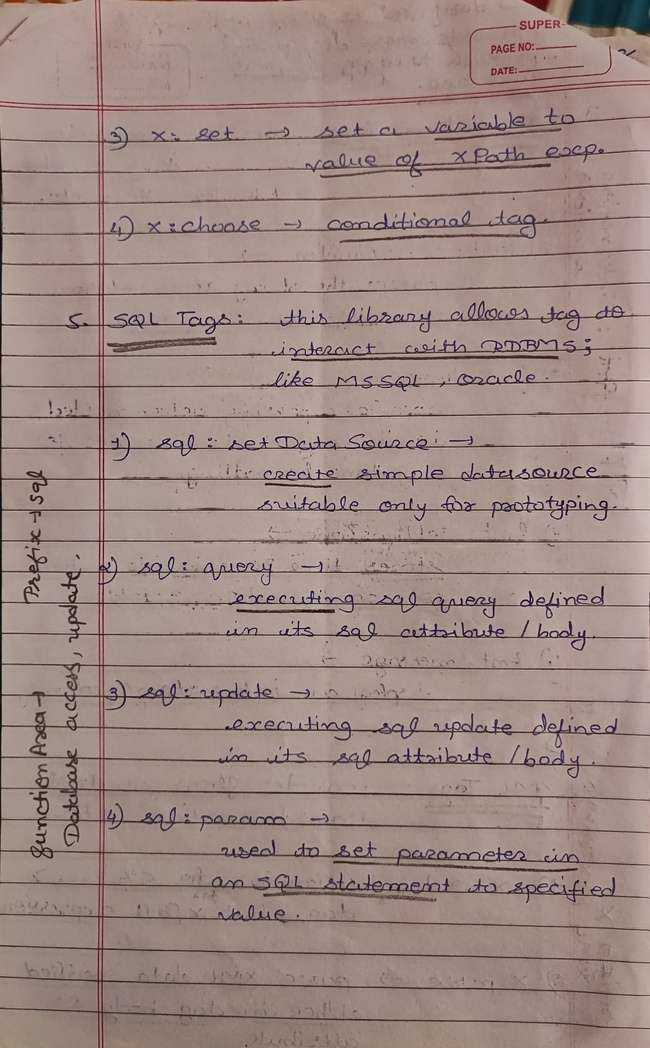
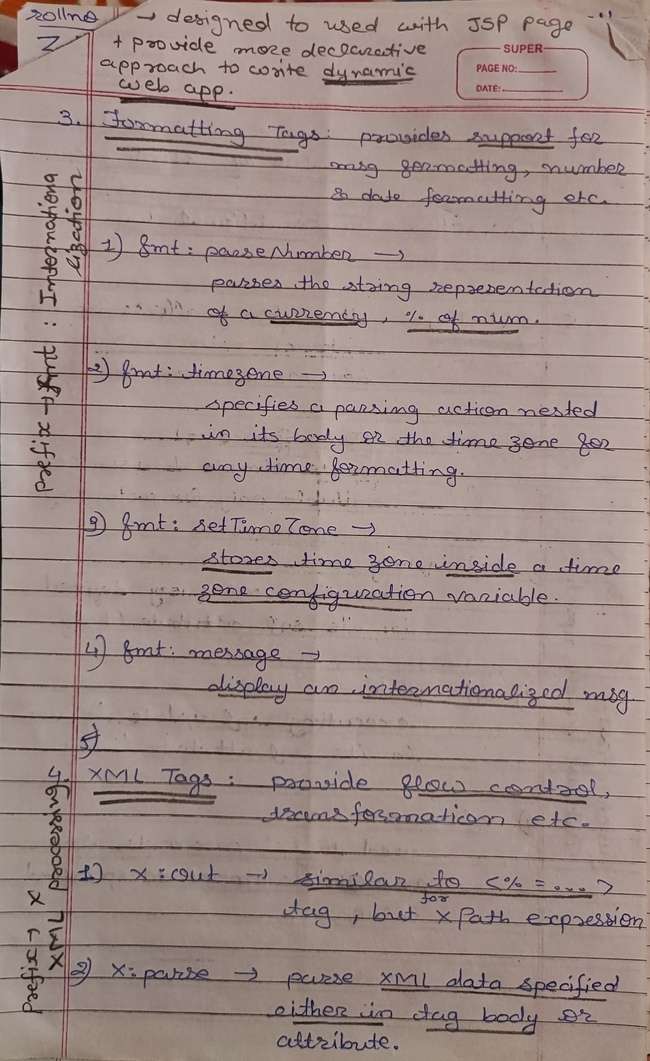
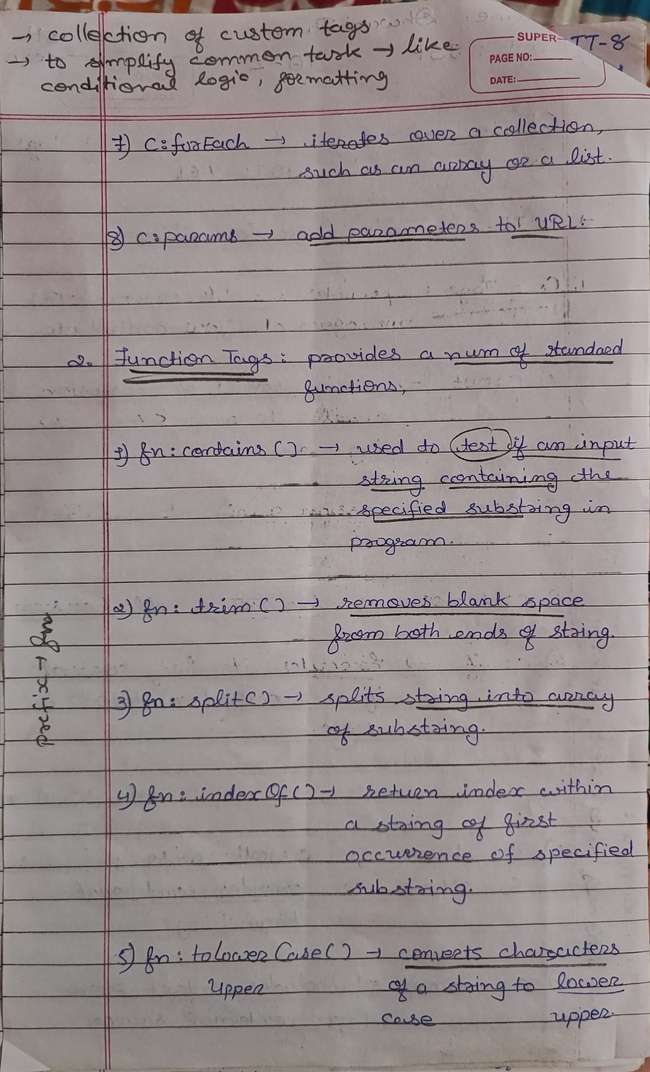
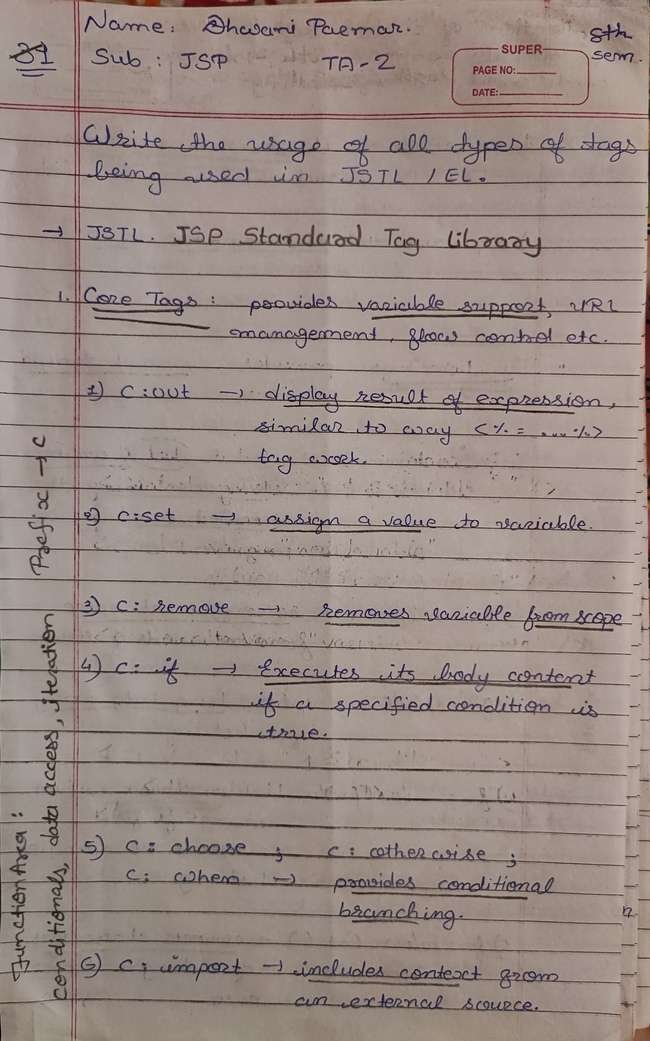
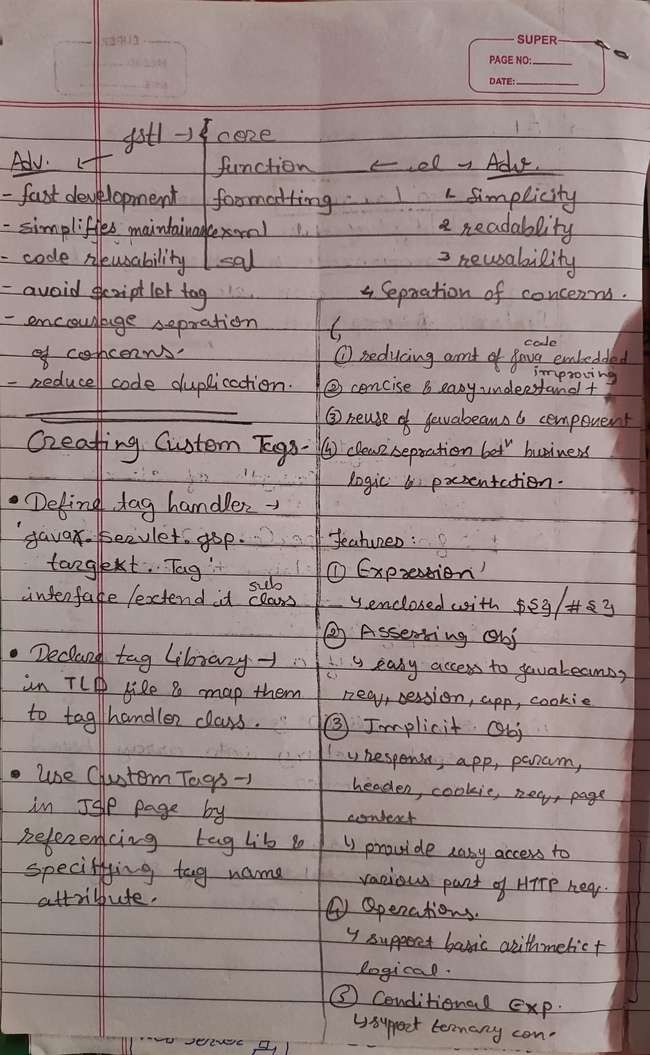
**JSP Scripting tag**

* In JSP, scripting tags are used to embed Java code directly within the HTML content of a JSP page.
* Scripting tags allow you to write Java code directly within your JSP page.
* They provide a way to add dynamic behavior to your web application and interact with data or perform calculations.
* Types of Scripting Tags:

1. Declaration (<%! %>)
2. Scriptlet (<% %>)
3. Expression (<%= %>)

|  |
| --- |
| Example:  <!DOCTYPE html>  <html>  <head>  <title>Scripting Tags Example</title>  </head>  <body>  <%-- Declaration --%>  <%!  int num1 = 10;  int num2 = 20;  %>  <%-- Scriptlet --%>  <%  int sum = num1 + num2;  %>  <%-- Expression --%>  <p>The sum of <%= num1 %> and <%= num2 %> is <%= sum %>.</p>  </body>  </html> |

**JSTL / EL**



**Event handling in JSF**

[https://www.tutorialspoint.com/jsf/jsf\_event\_handling.htm](https://www.tutorialspoint.com/jsf/jsf_event_handling.htm" \o "https://www.tutorialspoint.com/jsf/jsf_event_handling.htm)

* When a user clicks a JSF button or link or changes any value in the text field,

JSF UI component fires an event,

which will be handled by the application code.

* To handle such an event, an event handler is to be registered in the application code or managed bean.
* When a UI component checks that a user event has occured,

it creates an instance of the corresponding event class and adds it to an event list.

* Then, Component fires the event,
* i.e., checks the list of listeners for that event and calls the event notification method on each listener or handler.
* JSF also provide system level event handlers, which can be used to perform some tasks when the application starts or is stopping.

Following are some important Event Handler in JSF 2.0 −

S.No Event Handlers & Description

* 1 **valueChangeListener**
  + - Value change events get fired when the user make changes in input components.
* 2 **actionListener**
  + - Action events get fired when the user clicks a button or link component.
* 3 **Application Events**
  + - Events firing during JSF lifecycle: PostConstructApplicationEvent, PreDestroyApplicationEvent , PreRenderViewEvent.

**Servlet Filters**

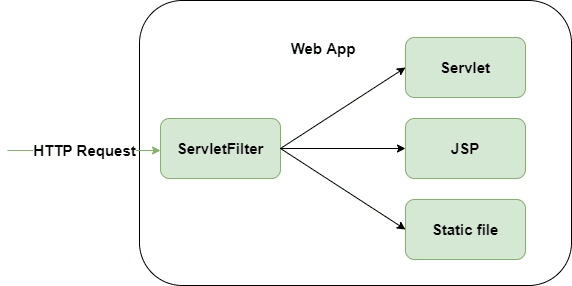
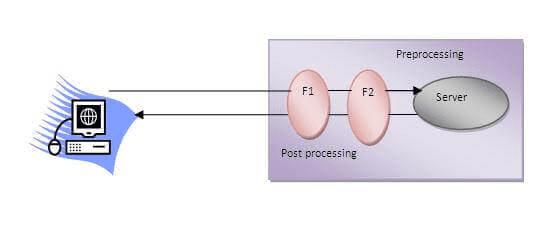
**[https://www.javatpoint.com/servlet-filter](https://www.javatpoint.com/servlet-filter" \o "https://www.javatpoint.com/servlet-filter)**

* filter is an object that is invoked at the preprocessing and postprocessing of a request.
* It is mainly used to perform filtering tasks such as conversion, logging, compression, encryption and decryption, input validation etc.
* The servlet filter is pluggable,
* i.e. its entry is defined in the web.xml file,

if we remove the entry of filter from the web.xml file,

filter will be removed automatically and

we don't need to change the servlet.

* So maintenance cost will be less.
* 
* Or
* 
* Unlike Servlet, One filter doesn't have dependency on another filter.

**Usage of Filter**

* ADVERTISEMENT
* recording all incoming requests
* logs the IP addresses of the computers from which the requests originate
* conversion
* data compression
* encryption and decryption
* input validation etc.

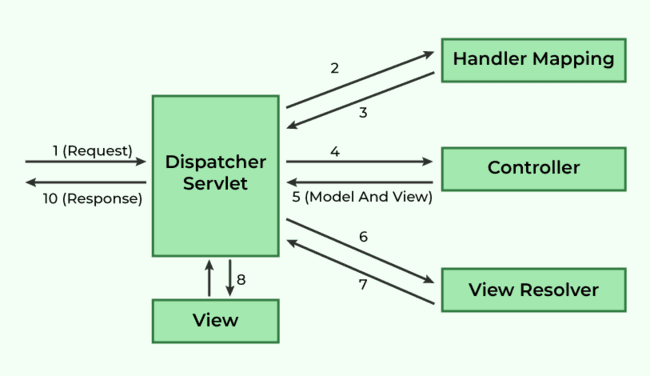
Servlet Listner

**Spring MVC**

[https://youtube.com/playlist?list=PL0zysOflRCelAb51IrexpUSeB0dpr9p6g&si=hPc0Cu93IfNGf6wj](https://youtube.com/playlist?list=PL0zysOflRCelAb51IrexpUSeB0dpr9p6g&si=hPc0Cu93IfNGf6wj" \o "https://youtube.com/playlist?list=PL0zysOflRCelAb51IrexpUSeB0dpr9p6g&si=hPc0Cu93IfNGf6wj)

[https://www.geeksforgeeks.org/spring-mvc-framework/](https://www.geeksforgeeks.org/spring-mvc-framework/" \o "https://www.geeksforgeeks.org/spring-mvc-framework/)

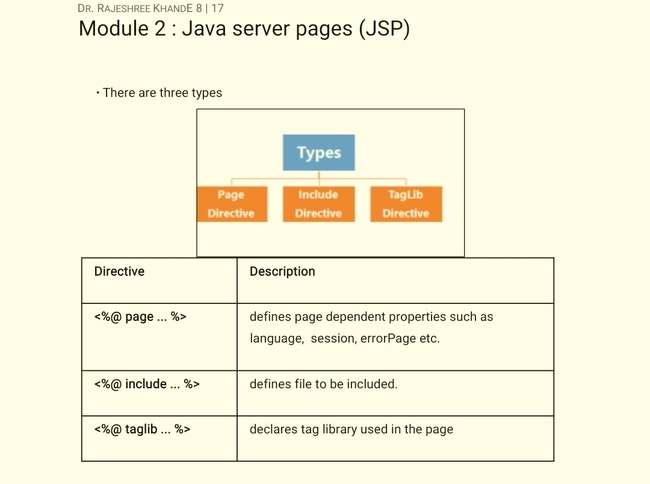
* Spring MVC Framework follows the Model-View-Controller architectural design pattern
* which works around the Front Controller
* i.e. the Dispatcher Servlet.
* The Dispatcher Servlet handles and dispatches all the incoming HTTP requests to the appropriate controller.
* It uses @Controller and @RequestMapping as default request handlers.
* The **@Controller** annotation => defines that a particular class is a controller.
* **@RequestMapping** annotation => maps web requests to Spring Controller methods.
* The terms model, view, and controller are as follows:
* **Model**: The Model encapsulates the application data.
* **View**: View renders the model data and generates HTML output that the client’s browser can interpret.
* **Controller**: The Controller processes the user requests and passes them to the view for rendering.



* Spring MVC Framework works as follows:

1. All the incoming requests are intercepted / stop by the DispatcherServlet that works as the front controller.
2. The DispatcherServlet then gets an entry of handler mapping from the XML file and forwards the request to the controller.
3. The object of ModelAndView is returned by the controller.
4. The DispatcherServlet checks the entry of the view resolver in the XML file and invokes the appropriate view component.

**JSP TYPES**



**server push feature of servlet**

* Server push is a feature of servlets that enables the server to send data to the client without the client explicitly requesting it.
* Server push, also known as HTTP/2 server push or HTTP/2 server-side push,
* is a feature that allows a servlet to proactively send data to the client without waiting for the client to request it.
* This is particularly useful for real-time updates, notifications, or streaming data scenarios.
* Key points

1. Proactive Data Transmission
2. Efficient Resource Loading
3. Push Promises
4. Use Cases

|  |
| --- |
| @WebServlet("/PushServlet")  public class PushServlet extends HttpServlet {  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  // Create HTTP/2 push builder  PushBuilder pushBuilder = request.newPushBuilder();    // Push resources to the client  if (pushBuilder != null) {  pushBuilder.path("style.css").push();  pushBuilder.path("script.js").push();  pushBuilder.path("image.jpg").push();  }    // Send response to the client  response.setContentType("text/html");  PrintWriter out = response.getWriter();  out.println("<html><head><title>Server Push Example</title></head><body>");  out.println("<h1>Welcome to Server Push Example</h1>");  out.println("</body></html>");  }  }  Simple : |

**Web Filtering**

[https://www.wallarm.com/what/web-filtering](https://www.wallarm.com/what/web-filtering" \o "https://www.wallarm.com/what/web-filtering)

* A web filtering software is used to control which websites an end user is able to access.
* It compares the content and its source to a set of rules in order to decide whether or not to display it.
* Internet content such as spyware, viruses, and material deemed improper for the workplace can be blocked by implementing a web filtering policy for an organization.
* Filtering unsuitable content from the internet can help employees operate more efficiently, limit legal exposure, and safeguard a company's network from outside attacks.

|  |
| --- |
| import javax.servlet.\*;  import javax.servlet.annotation.WebFilter;  import java.io.IOException;  @WebFilter("/\*")  public class LoggingFilter implements Filter {  public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {  // Log details of the incoming request  System.out.println("Request received for: " + request.getLocalAddr() + " - " + ((HttpServletRequest) request).getRequestURI());  // Pass the request/response to the next filter or servlet in the chain  chain.doFilter(request, response);  }  public void init(FilterConfig config) throws ServletException {  // Initialization logic, if needed  }  public void destroy() {  // Cleanup resources, if needed  }  } |