Servlet technology

---------------------------------

Tech stack?

* Desigining a web application involves the tasks like displaying the data sent from the server, handling client side operations, handling server side operations and handling database operations.
* To ensure the smooth execution of these operations, we use a combination of programming languages, tools and frameworks, called a tech stack.
* The tech stack, is visualized as two parts.

1. front end: refers to the technologies used for interaction with the user.
2. back end: refers to the technologies used on the server for processing the user inputs and interacting with the database.

* some of the tech stacks are,

MERN(MonogDB-Express.js-React JS-Node.js)

MEAN(MongoDB-Express.js-Angular-Node.js)

LAMP(Linux-Apache-MySQL-PHP)

MongoDB:

* MongoDB is an open source document oriented database written in C++.
* Data in MongoDB is stored in documents, with in a collection, as a set of name-value pairs.

Express.js:

* Express.js is a lightweight application development framework.
* Express.js is used for back-end development.

Angular:

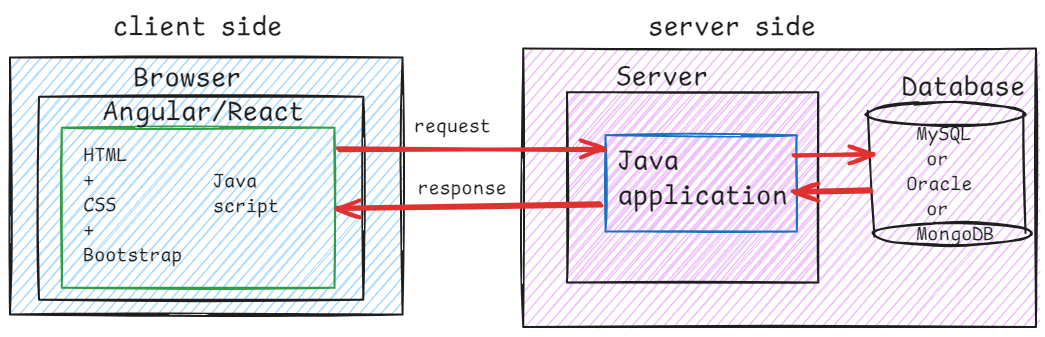
* Angular is an open source Javascript framework developed by Google, for building both mobile and web applications.
* It is a good framework for rapid front-end development.

Node.js:

* Node.js is a server-side, open source Javascript execution environment.
* It is built on Google chrome’s V8 Javascript runtime.

React JS:

* React is an open source Javascript library/framework developed by facebook, for building both mobile and web applications.
* It is a good framework for rapid front-end development.
* The MEAN / MERN stack, is good for developing client-side application development, not for server-side application development.
* client-side application runs on the user’s machine or mobile phone, which will communicate with the server to fetch the required information.
* server-side application contains the business logic and validations for processing the data sent by the client.
* For server-sde application development, we use the technologies like Java, Kotlin, .net, Python, etc..

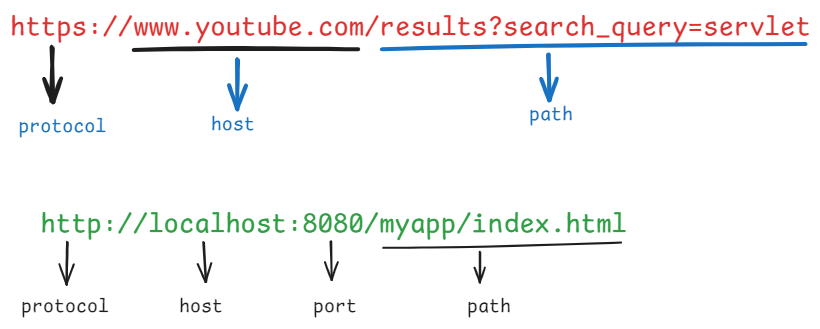


What is URL?

* Uniform Resource Locator
* The URL uniquely identifies a particular resource on the internet.

protocol://host:port/path

for ex:



HTTP (Hyper Text Transfer Protocol):

* If any client wants to communicate with any server, both the client and the server must agree on some predefined rules of communication, and these rules are called protocols.
* HTTP is a protocol in web communication.
* The web browser and the web server communicate with each other with HTTP protocol.
* HTTP is a stateless and connection less protocol.
* The client establishes a connection with the server and sends the request. Over the same connection, server sends the response. Once the response is delivered, the connection is terminated.

MIME(Multipurpose Internet Mail Extension):

* MIME is a standard way to indicate the format of response.
* server inserts the MIME type/content type into response, before sending it to the client.
* client uses this content type and selects appropriate application to show the data to the user.
* a MIME type has a format type/sub-type.

For ex:

text/html

text/plain

text/css

image/jpeg

image/png

image/gif

application/json

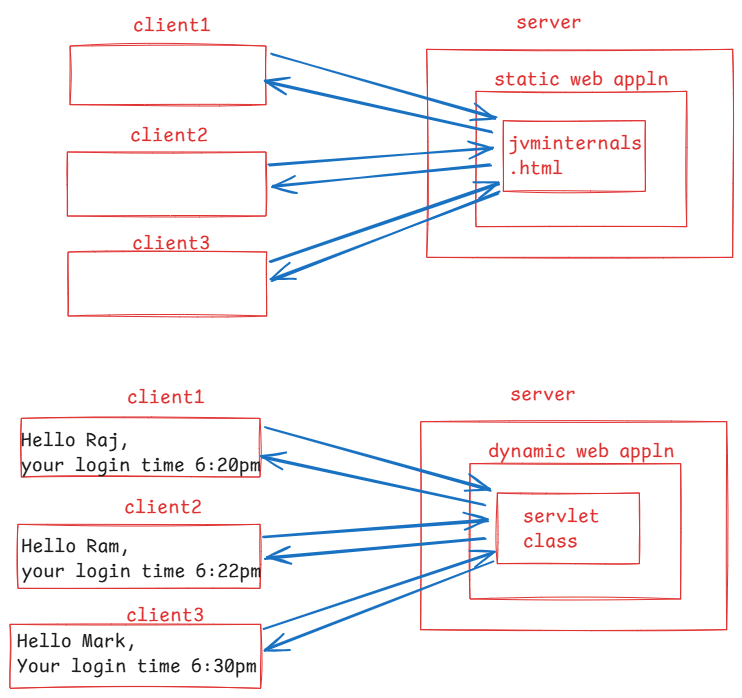
application/xml

Types of web applications:

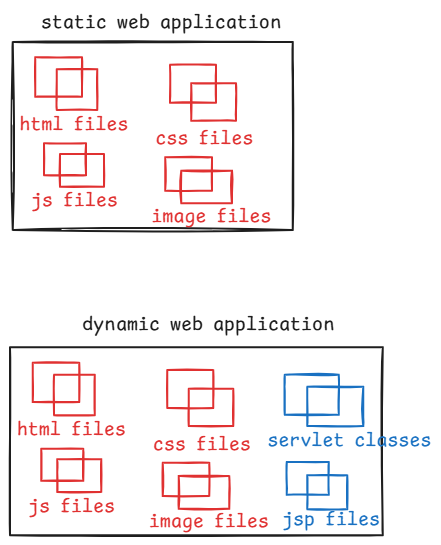
--------------------------

1. static web application
2. dyanamic web application.

* static web application runs on a server and provided static web pages to the users.
* dynamic web application runs on a server and provided either static/dynamic web pages to the users.
* static web applications delivers the same pages as a response to the clients.
* dynamic web applications delivers dynamic pages as a reponse to the clients.



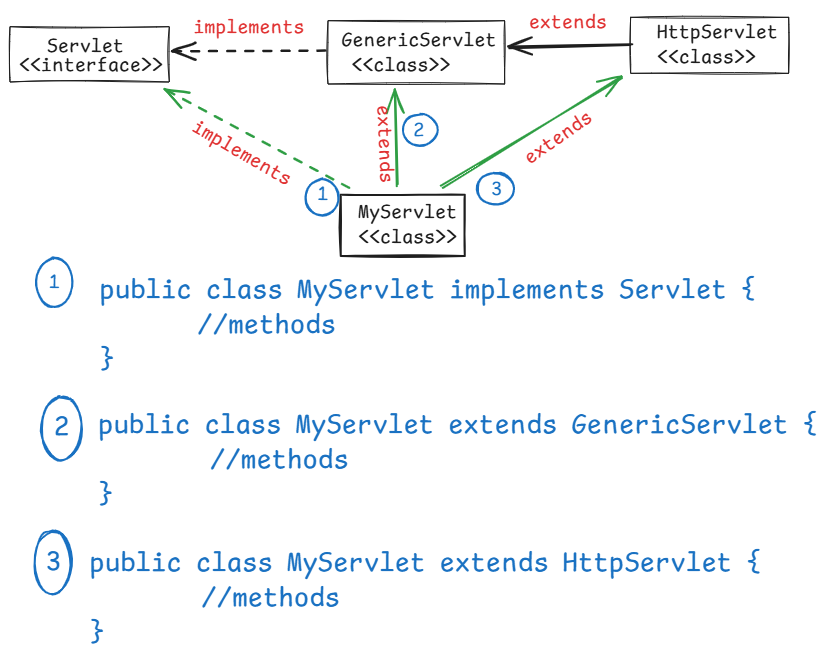
* To develop a static web application, we need the client-side web technologies like HTML, CSS, Javascript, Bootstrap,etc.
* To develop a dynamic web application, we need to use both the client-side web technologies and also, the server-side web technologies like servlet or JSP or ASP.net or PHP, etc.



Definition of a servlet:

* A servlet is a Java class, which executes on a server, reads the input from the user, executes some application logic and generates the data and finally sends the data as a response to the user.

How can we create a servlet class?



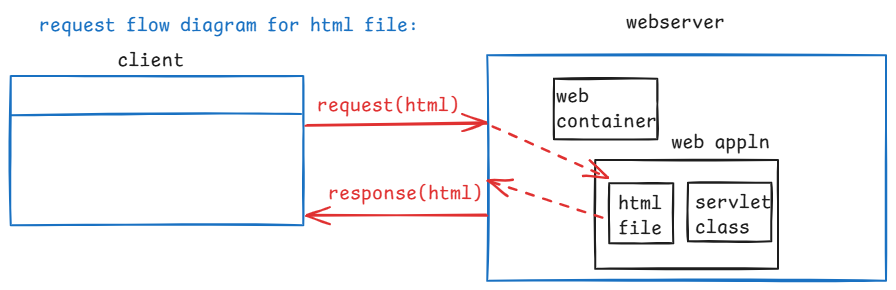
* SERVLET API has provided two packages.

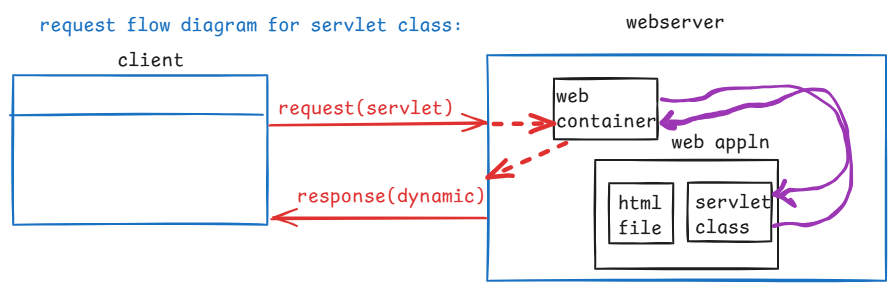
1. javax.servlet package
2. javax.servlet.http package

* If our servlet class implements Servlet interface then it has to override all the abstract methods(5).
* If our servlet class extends GenericServlet abstract class then it has to override one abstract method(1).
* If our servlet class extends HttpServlet abstract class then it has to override the methods of HttpServlet class based on the requirements.
* Actually HttpServlet class is an abstract class but it does not have any abstract methods.

Q) can we create an abstract class without abstract methods?

A) Yes. Suppose, if your class is not a fully implemented class, which means, the class has defined methods, but they are not upto the mark then you can declare your class as an abstract class.

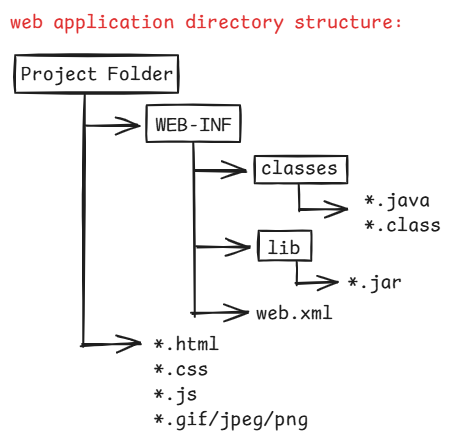




* If a request comes from the client to a static page like html/image then the webserver itself will handle that request and provides that html file or image to the client.
* If a request comes from the client to a servlet class then the webserver will forward that request to another component called web container/servlet container. This container will execute the servlet class and returns a servlet response to the web server.
* Finally, the webserver will send the response to the client.

who is this web container?

* It is a component available in a web server application and it is responsible for executing servlet classes/JSP files in a server.



Installing Tomcat server:

1. visit <https://tomcat.apache.org/>
2. under download, click Tomcat 10, then click the link 10.1.3
3. click 32/64-bit windows service installer
4. apache-tomcat-10.1.30.exe file is downloaded
5. click on the .exe file, follow the next buttons,

enter shutdown port: 8085

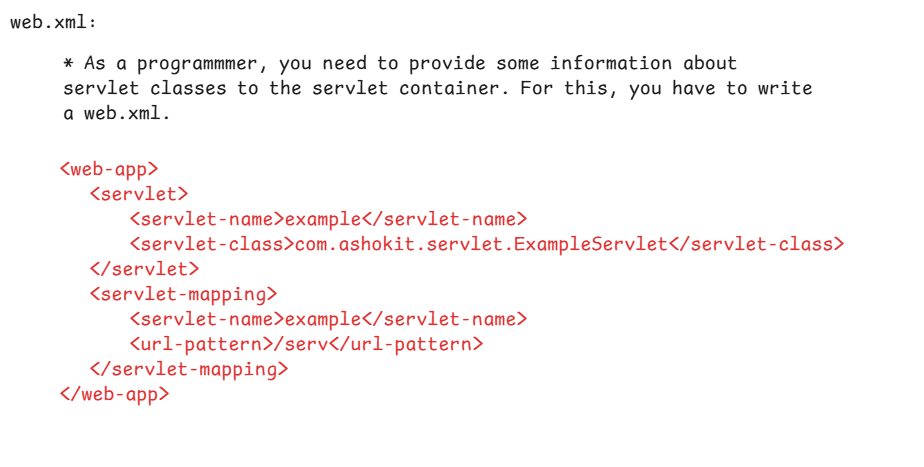
username: admin

password: admin

click on next buttons, on the last screen, uncheck the boxes, then click on finish.

Note: Now you machine/system has Tomcat server, at

C:\Program Files\Apache Software Foundation\Tomcat 10.1



First web application steps in Eclipse:

1. Launch eclipse
2. File 🡪 New 🡪 Dynamic web project 🡪

project: FirstWebApp

dynamic web module version: 2.5

1. follow next buttons 🡪 finish
2. expand the project folder(FirstWebApp) 🡪 expand src folder 🡪 expand main folder 🡪 right click on webapp folder 🡪 new 🡪 HTML file 🡪 enter filename: index.html 🡪next 🡪 finish
3. add the anchor tag like below.

<body>

<a href = *"./serv"*> click me </a>

</body>

1. Right click on the project folder 🡪 build path 🡪 configure build path 🡪 Add external jars 🡪 navigate to Tomcat\lib folder 🡪 choose servlet-api.jar 🡪 open
2. right click on java folder 🡪 new 🡪 other 🡪 class 🡪 enter

package: com.ashokit.example

name : ExampleServlet

super class: Click Browse and type generic, then select GenericServlet 🡪 finish.

**public** **class** ExampleServlet **extends** GenericServlet {

@Override

**public** **void** service(ServletRequest request, ServletResponse response) **throws** ServletException, IOException {

//set the content type

response.setContentType("text/html");

//create PrintWriter object

//using this object, a servlet class can write the response content.

PrintWriter out = response.getWriter();

out.println("<html>");

out.println("<body>");

out.println("<h2> Welcome to Servlet Programming </h2>");

out.println("</body>");

out.println("</html>");

out.close();

}

}

1. open web.xml file, and write the below code.

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns=*"http://java.sun.com/xml/ns/javaee"* xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"* id=*"WebApp\_ID"* version=*"2.5"*>

<display-name>FirstWebApp</display-name>

<servlet>

<servlet-name>example</servlet-name>

<servlet-class>com.ashokit.example.ExampleServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>example</servlet-name>

<url-pattern>/serv</url-pattern>

</servlet-mapping>

</web-app>

1. click on window menu 🡪 show view 🡪 severs
2. click on new server 🡪 expand Apache folder🡪 select Tomcat v10.1 🡪 next 🡪 browse 🡪 select Tomcat 10.1 folder from Apache Software Foundation 🡪 finish.
3. Right click on the project folder 🡪 RunAs 🡪 Run on server 🡪 choose Tomcat v10.1 server 🡪 next 🡪 finish
4. open a browser like Google chrome, and enter the below request url.

<http://localhost:8080/FirstWebApp/index.html>

Life cycle methods of a Java servlet:

* Life cycle methods of a Java servlet refers to the methods that are executed during the servlet object creation phase, execution phase and the termination phase.
* The life cycle methods are used to initialize the resources, handle the requests and clean up the resource before the object is being destroyed.
* The life cycle methods are init, service and destroy.
* init method: When a servlet class object is created, this init method is called by the web container for only once.
* This init method is used to initalize the resources like opening a file, creating database connections, creating mail server connections, etc.

syntax:

public void init(ServletConfig config) throws ServletException {

//initialization logic

}

* service method: The service method is called by the container for each request from the clients, to handle the request and to generate dynamic response.

syntax:

public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException {

//request handling logic

}

* destroy method: This method is called by the web container for only once, when a servlet object is removed from the memory, or when a server is shutdown.
* This method is used to release the resources like closing the database connections, closing the files, closing the mail server connections etc

syntax:

public void destroy() {

//release the resources

}

early and lazy loading of a servlet:

------------------------------------

* For a servlet class, the object will be created by the web container/servlet container.
* The servlet container creates the object for a servlet class, when a first request is arrived.
* It means, by default a servlet is lazy loaded.
* Suppose, if a servlet object should be created before any request, then we can tell the container that early load the servlet.
* To tell the container, in the <servlet> tag of web.xml file, include <load-on-startup> tag with a value >=0.

for example,

<servlet>

<servlet-name>example</servlet-name>

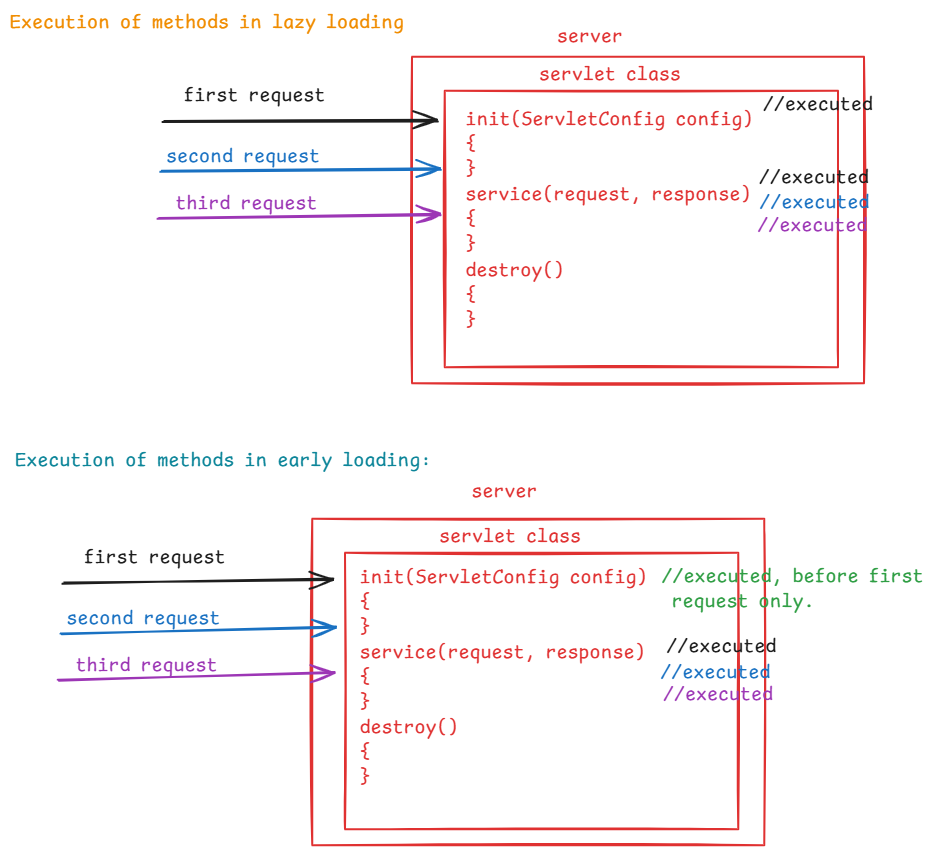
<servlet-class>

com.ashokit.example.ExampleServlet

</servlet-class>

<load-on-startup> 1 </load-on-startup>

</servlet>



HTTP methods:

* while clients are communicating with the servers, the clients like browsers will use some standard request types called HTTP methods.
* The HTTP methods, defines what action should be taken by the server, when a client makes a request.
* There are 7 HTTP methods.

GET

POST

PUT

DELETE

PATCH

HEAD

OPTIONS

* The client(browser) uses request type as GET, to retrieve the data or a file from the server, without making any changes to the server.
* The client uses request type as POST, to send the data to the server. Typically it is used to create a new resource in the server.
* While sending the sensitive data like emails, credit card numbers, passwords, OTP’s etc, to the server, the client uses the request type as POST.
* The client uses request type as PUT, to update or replace an existing data on the server.
* The client uses request type as DELETE, to delete a resource from the server.
* GET : Read
* POST : Create
* PUT : Update
* DELETE : Delete

HttpServlet class:

* HttpServlet is an abstract class, which extends GenericServlet class.
* GenericServlet class has an abstract method service(), which is a life cycle method and HttpServlet class has overridden this service() method.
* HttpServlet class does not contain abstract methods, but the class is abstract class.
* In Java, you can define a class as abstract class, without abstract methods.
* HttpServlet class has defined methods to handle the different HTTP request types, like

doGet(req,res): To handle HTTP GET request

doPost(req,res): To handle HTTP POST request

doPut(req,res): To handle HTTP PUT request

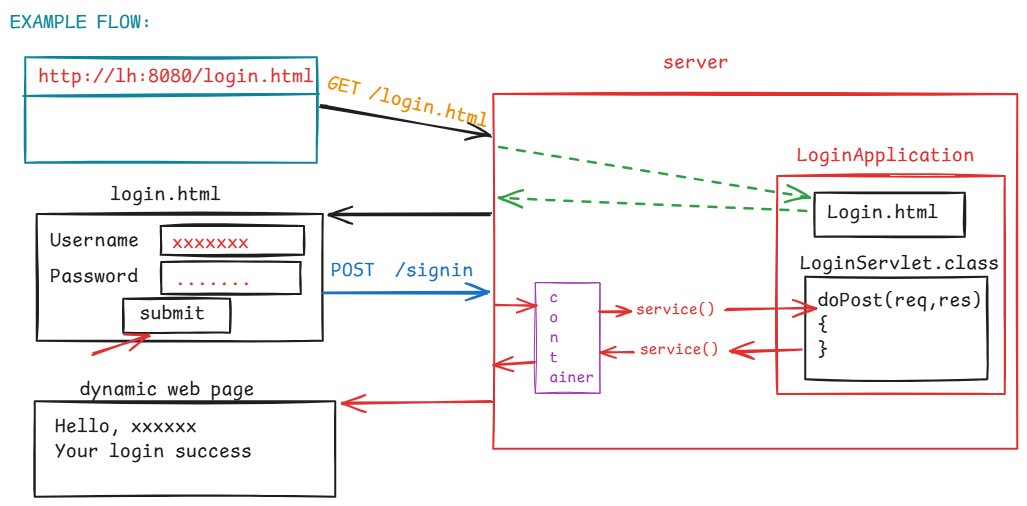
doDelete(req,res): To handle HTTP DELETE request

doPatch(req,res): To handle HTTP PATCH request

doHead(req,res): To handle HTTP HEAD request

doOptions(req,res):To handle HTTP OPTIONS request

* In short the above seven methods are called doXxx() methods.
* The functionality defined in the doXxx() methods, it not upto the mark. It means, this logic can’t handle the requests properly.
* So, to denote that HttpServlet class as incomplete class, it is given as abstract class.
* In our web application development, we create our servlet classes mostly by extending HttpServlet class. Because, we can define the logics in different methods like doGet()/doPost(), so the servlet becomes easily understandable to the other developers.
* For each request, the servlet container calls service() [life cycle method] and it will dispatch the request to doGet()/doPost(), based on request type.



login.html

<!DOCTYPE html>

<html>

<head>

<meta charset=*"UTF-8"*>

<title>Login Page</title>

</head>

<body>

<form action=*"./signin"* method=*"post"*>

<table>

<tr>

<td>Username </td>

<td> <input type=*"text"* name=*"username"*> </td>

</tr>

<tr>

<td>Password </td>

<td> <input type=*"password"* name=*"password"*> </td>

</tr>

</table>

<button type=*"submit"*>submit</button>

</form>

</body>

</html>

LoginServlet.java

**package** com.ashokit.example;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** jakarta.servlet.ServletException;

**import** jakarta.servlet.http.HttpServlet;

**import** jakarta.servlet.http.HttpServletRequest;

**import** jakarta.servlet.http.HttpServletResponse;

**public** **class** LoginServlet **extends** HttpServlet {

@Override

**protected** **void** doPost(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {

resp.setContentType("text/html");

PrintWriter out = resp.getWriter();

//read the user input

String user = req.getParameter("username");

String pwd = req.getParameter("password");

**if**( user!=**null** && user.trim().length() > 0 && user.equals("Allen"))

{

**if** (pwd != **null** && pwd.trim().length() >0 && pwd.equals("allen@123"))

{

out.println("<html> <body>");

out.println("<h2> Hello, " + user + "</h2> <br> ");

out.println("<font color='green'>Your Login success! </font>");

out.println("</body> </html>");

}

**else** {

out.println("<html> <body> ");

out.println("<h2> Hello, " + user + "</h2> <br> ");

out.println("<font color='red'> Your password is mismatched </font> <br>");

out.println("<a href='login.html'>Try again </a>");

out.println("</body> </html>");

}

}

**else** {

out.println("<html> <body> ");

out.println("<font color='red'> Username or Password is mismatched </font> <br>");

out.println("<a href='login.html'>Try again </a>");

out.println("</body> </html>");

}

out.close();

}

}

web.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns=*"http://java.sun.com/xml/ns/javaee"* xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"* id=*"WebApp\_ID"* version=*"2.5"*>

<display-name>LoginApp</display-name>

<servlet>

<servlet-name>Login</servlet-name>

<servlet-class>com.ashokit.example.LoginServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>Login</servlet-name>

<url-pattern>/signin</url-pattern>

</servlet-mapping>

</web-app>

Can I execute my application from outside of the Eclipse IDE?

Answer: Yes, You can.

How?

Answer: follow the below steps.

1. open a command prompt and place the cursor under the project folder.

ex:

D:\Servlet-5PM\LoginApp>

1. create a war(web archive) file, with the below command.

D:\Servlet-5PM\LoginApp> jar -cvf LoginApp.war .

1. copy the LoginApp.war file to C:\Program Files\Apache Software Foundation\Tomcat 10.1\webapps folder.
2. Goto C:\Program Files\Apache Software Foundation\Tomcat 10.1\bin folder, double click on Tomcat10 to start the server.
3. open the browser, and enter the below url

http://localhost:8080/LoginApp/src/main/webapp/login.html

LoginApp with Database:

1. create a table in MySQL like below:

CREATE TABLE USER\_LOGIN (

USERNAME VARCHAR(20),

PASSWORD VARCHAR(20),

STATUS VARCHAR(20),

PRIMARY KEY(USERNAME)

);

1. insert some sample records into the table and commit.
2. copy mysql-connector-j-8.3.0.jar into WEB-INF\lib folder.
3. copy login.html file from the previous example
4. add servlet-api.jar to the build path
5. create LoginServlet.java file like below.

**package** com.ashokit.demo;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.PreparedStatement;

**import** java.sql.ResultSet;

**import** jakarta.servlet.ServletException;

**import** jakarta.servlet.annotation.WebServlet;

**import** jakarta.servlet.http.HttpServlet;

**import** jakarta.servlet.http.HttpServletRequest;

**import** jakarta.servlet.http.HttpServletResponse;

@WebServlet("/signin")

**public** **class** LoginServlet **extends** HttpServlet {

@Override

**protected** **void** doPost(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {

//set the content type

resp.setContentType("text/html");

//create PrintWriter object

PrintWriter out = resp.getWriter();

//read the request parameters

String username = req.getParameter("username");

String password = req.getParameter("password");

Connection conn = **null**;

PreparedStatement pstmt = **null**;

**try** {

Class.*forName*("com.mysql.cj.jdbc.Driver");

conn = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/test", "root", "root");

pstmt = conn.prepareStatement("SELECT \* FROM USER\_LOGIN WHERE USERNAME = ?");

pstmt.setString(1, username);

ResultSet rs = pstmt.executeQuery();

**if**(rs.next()) {

String pwd = rs.getString("PASSWORD");

String status = rs.getString("STATUS");

**if**(pwd.equals(password) && status.equalsIgnoreCase("active"))

{

out.println("<html> <body>");

out.println("<h2> Hello, " + username + "</h2> <br> ");

out.println("<font color='green'>Your Login success! </font>");

out.println("</body> </html>");

}

**else** {

out.println("<html> <body> ");

out.println("<h2> Hello, " + username + "</h2> <br> ");

out.println("<font color='red'> Your password is mismatched or user is inactive</font> <br>");

out.println("<a href='login.html'>Try again </a>");

out.println("</body> </html>");

}

}

**else** {

out.println("<html> <body> ");

out.println("<font color='red'> Username is mismatched </font> <br>");

out.println("<a href='login.html'>Try again </a>");

out.println("</body> </html>");

}

rs.close();

}

**catch**(Exception ex) {

ex.printStackTrace();

}

**finally** {

**try** {

**if**(pstmt != **null**)

pstmt.close();

**if**(conn!=**null**)

conn.close();

}

**catch**(Exception ex) {

ex.printStackTrace();

}

}

out.close();

}

}

1. In web.xml file, add the below tag.

<welcome-file-list>

<welcome-file>login.html</welcome-file>

</welcome-file-list>

1. execute the project.

Request dispatching in servlets:

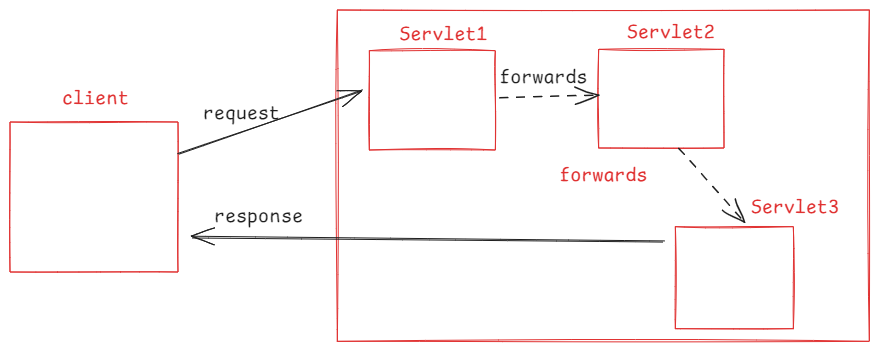
* Request dispatching is a process of forwarding a request from one servlet to another resource like another servlet or HTML or JSP, or including the content of another resource to the response of the current servlet.
* If you have created different servlet classes to handle different tasks, instead of writing the complete logic in a single servlet class, then the servlet classes should collaborate to fulfill the user request.
* Request dispatching is a process, which provides servlets collaboration.
* for example, when a user is a buying a product in e-commerce application, by clicking on proceed to checkout button.

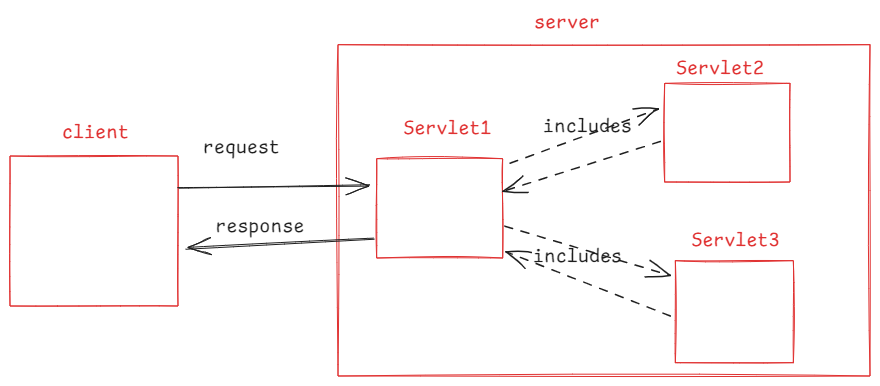
1. a CartServlet retrieves all the items from the cart and then forwards the request to CheckoutServlet.
2. a CheckoutServlet collects user details like shipping address and then forwards the request to PaymentServlet.
3. a PaymentServlet collects payment details, processes the payment then forwards the rquest to ConfirmationServlet.
4. a ConfirmationServlet displays the confirmation details like order number, shipping details, expected delivery date etc.. to the user.

* RequestDispatcher interface is provided with two methods, for forwarding a request and for including the content.

1. forward(req, resp)
2. include(req, resp)

* when a request is forwarding to another servlet or when the content is including from another servlet, the information is not provided to the client.





* Some key points are,

. one servlet can forward a request to atmost one servlet at a time.

. one servlet can include the content from multiple servlets also at a time.

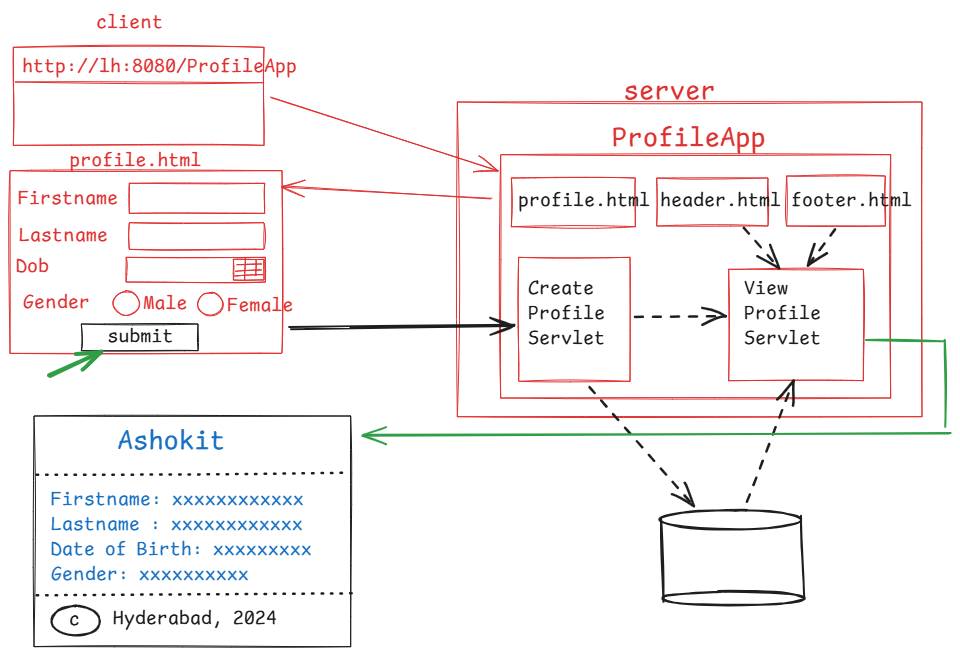
. In forwarding, the control is passed to the next resource, and it is not returned back.

. In including, the control is passed to the next resource, and it is returned back.

. In forwarding, the last servlet in the chain generates the dynamic web page.

. In including, both the original and included servlets contribute to the response.

. Forwarding is a one-to-one communication between servlets and including is a one-to-many communication between servlets.



CREATE TABLE USER\_PROFILES (

FIRSTNAME VARCHAR(20),

LASTNAME VARCHAR(20),

DATEOFBIRTH DATE,

GENDER VARCHAR(20)

);

profile.html

<!DOCTYPE html>

<html>

<head>

<meta charset=*"UTF-8"*>

<title>Profile Page</title>

</head>

<body>

<form action=*"./create"* method=*"post"*>

<table>

<tr>

<td>First Name </td> <td> <input type=*"text"* name=*"fname"*> </td>

</tr>

<tr>

<td>Last Name </td> <td> <input type=*"text"* name=*"lname"*> </td>

</tr>

<tr>

<td>Dob </td> <td> <input type=*"date"* name=*"dob"*> </td>

</tr>

<tr>

<td>Gender </td>

<td>

<input type=*"radio"* name=*"gender"* value=*"Male"*>Male

<input type=*"radio"* name=*"gender"* value=*"Female"*>Female

</td>

</tr>

</table>

<button type=*"submit"*>submit</button>

</form>

</body>

</html>

header.html

<!DOCTYPE html>

<html>

<head>

<meta charset=*"UTF-8"*>

<title>Insert title here</title>

</head>

<body>

<center>

<font color=*'green'* size=*"10"*>Ashokit</font>

</center>

</body>

</html>

footer.html

<!DOCTYPE html>

<html>

<head>

<meta charset=*"UTF-8"*>

<title>Insert title here</title>

</head>

<body>

<center>

<h2> &copy; Hyderabad, 2024 </h2>

</center>

</body>

</html>

CreateProfileServlet.java

package pack1;

import java.io.IOException;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.time.LocalDate;

import jakarta.servlet.RequestDispatcher;

import jakarta.servlet.ServletException;

import jakarta.servlet.annotation.WebServlet;

import jakarta.servlet.http.HttpServlet;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.servlet.http.HttpServletResponse;

@WebServlet("/create")

public class CreateProfileServlet extends HttpServlet {

@Override

protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

//read request parameters

String firstName = req.getParameter("fname");

String lastName = req.getParameter("lname");

String strDate = req.getParameter("dob");

String gender = req.getParameter("gender");

//converting a string to LocalDate

LocalDate dob = LocalDate.parse(strDate);

Connection conn = null;

PreparedStatement pstmt = null;

try {

Class.forName("com.mysql.cj.jdbc.Driver");

conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/test", "root", "root");

pstmt = conn.prepareStatement("INSERT INTO USER\_PROFILES VALUES(?, ?, ?, ?)");

//setting the values

pstmt.setString(1, firstName);

pstmt.setString(2, lastName);

pstmt.setObject(3, dob);

pstmt.setString(4, gender);

pstmt.executeUpdate();

pstmt.close();

}

catch(Exception ex) {

ex.printStackTrace();

}

finally {

try {

if(pstmt != null)

pstmt.close();

if(conn!=null)

conn.close();

}

catch(Exception ex) {

ex.printStackTrace();

}

}

//forward the request to next servlet

RequestDispatcher dispatcher = req.getRequestDispatcher("./view");

dispatcher.forward(req, resp);

}

}

ViewProfileServlet.java

package pack1;

import java.io.IOException;

import java.io.PrintWriter;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import jakarta.servlet.RequestDispatcher;

import jakarta.servlet.ServletException;

import jakarta.servlet.annotation.WebServlet;

import jakarta.servlet.http.HttpServlet;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.servlet.http.HttpServletResponse;

@WebServlet("/view")

public class ViewProfileServlet extends HttpServlet {

@Override

protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

//set the content type

resp.setContentType("text/html");

//create PrintWriter object

PrintWriter out = resp.getWriter();

//read the request parameter firstname

String firstName = req.getParameter("fname");

Connection conn = null;

PreparedStatement pstmt = null;

try {

Class.forName("com.mysql.cj.jdbc.Driver");

conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/test", "root", "root");

pstmt = conn.prepareStatement("SELECT \* FROM USER\_PROFILES WHERE FIRSTNAME = ?");

pstmt.setString(1, firstName);

RequestDispatcher dispatcher1 = req.getRequestDispatcher("header.html");

dispatcher1.include(req, resp);

out.println("<hr>");

ResultSet rs = pstmt.executeQuery();

if(rs.next()) {

out.println("<h3>");

out.println("Firstname : " + rs.getString(1));

out.println("<br>");

out.println("Lastname : " + rs.getString(2));

out.println("<br>");

out.println("DateofBirth : " + rs.getString(3));

out.println("<br>");

out.println("gender : " + rs.getString(4));

out.println("<br>");

out.println("</h3>");

}

out.println("<hr>");

RequestDispatcher dispatcher2 = req.getRequestDispatcher("footer.html");

dispatcher2.include(req, resp);

rs.close();

}

catch(Exception ex) {

ex.printStackTrace();

}

finally {

try {

if(pstmt != null)

pstmt.close();

if(conn!=null)

conn.close();

}

catch(Exception ex) {

ex.printStackTrace();

}

}

out.close();

}

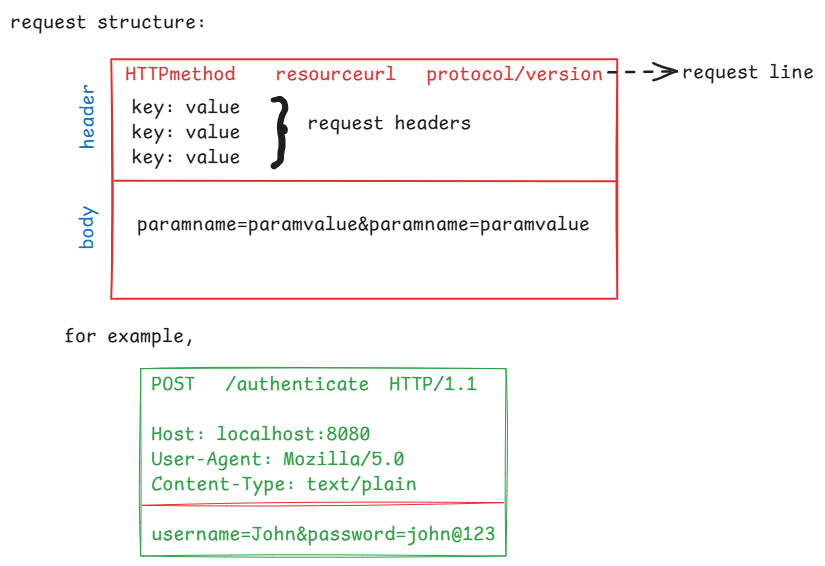
}

forwarding vs redirecting:

* Both of these are for servlet to servlet communication. but they have some differences.
* 1. forwarding is done without any information to the client. But redirection provides information to the client.
* 2. In forwarding, the same original request is forwarded from servlet1 to servlet2. In Redirection, the client/browser will make a new request to servlet2.
* 3. In forwarding, the same original request is forwarded to servlet2. So, the parameters are also forwarded. But in redirection, new request is created. So, the parameters in the original request are not redirected.
* 4. For request forwarding, both the servlet classes must be on same server. For redirection, both the servlet classes may on the same server or on different servers.
* 5. For forwarding, we call dispatcher.forward(req,resp). For redirection, we call resp.sendRedirect(“url”).

request and response structure:

* a request is a message, created by the client like browser/application and then it will be sent to the server.
* a response is a message, created by the server and sent back to the client.





HTTP status codes:

* In response, the server will send response status code to the client.
* This code indicates that whether request was successful, or there was any error, of if any additional actions are required.
* These status codes are 5 categories.

1xx : informational

2xx : success

3xx : redirection

4xx : client errors

5xx : server errors

100 Continue

. when a client is going to send a large file/image in the request body, first the client will send a request and tells the server that I am going to send a large data/file in the request.

. The server responds with status code 100 continue, then the client will send the large file/data in the request body.

200 Ok

.The server sends status code 200 Ok, if the request was processed successfully and if the server has returned the response data.

201 Created

.The server responds with 201 Created, if the request was successful and if a new resource is created at the server, because of this request.

204 No Content

.The server responds with 204 No Content, if the request was successful, but the server is not returning any content.

301 Moved Permanently

.The server responds with 301 Moved Permanently, if the request resource has been permanently moved to a new URL.

302 Found

.The server responds with 302 Found, if the requested resource is temporarily available at a different URL.

400 Bad Request

.The server responds with 400 Bad Request, if the server is unable to understand the request due to invalid syntax.

401 Unauthorized

.The server responds with 401 Unauthorized, if the client is sending the request to a protected resource, without authorization.

404 Not Found

.The server responds with 404 Not Found, if the server can not find the request resource.

405 Method NotAllowed

.The server responds with 405 Method Not Allowed, if the request method is not supported by the server.

500 Internal Server Error

.The server responds with 500 Internal Server Error, if the server has encountered a generic error.

503 Service Unavailable

.The server responds with 503 Service Unavailable, due to overload of requests.

Session Management

* HTTP is a stateless protocol.
* stateless means, every request from a client to a server is treated as an independent request. The server does not rememeber previous interactions with the client.
* Suppose, if a user is sending multiple requests, the server treats like each request is coming from a new user.
* In most web applications, keeping track of information about the user across the mulitple requests is required.
* So, session management techniques are used.
* Session management is a process, used to maintain stateful interactions between the client and server in web applications, despite of the stateless nature of HTTP.
* Suppose, in E-commerce applications, if session management technique is not used, then when a user is moving from one page another page, the server treats that user as a new user and the cart will be reset with every request. So, a user can not purchase multiple items at a time.
* If session management technique is used, the server can track and manage the user and the items added to the cart when a user is moving between pages. Which means, the server remembers the user’s data across mulitple requests.
* The servlets technology has provided HTTP Cookies and HttpSession api to manage the sessions.

HTTP Cookies:

-------------

* Cookie class is provided in Servlet API, to create cookies.
* A cookie is an object, which can store a key-value pair, created at server and the server sends to the client, where client will store and sends back to the server with the subsequent requests.
* The server can create multiple cookies to keep track of user information across multiple requests.
* If there are more cookies, then sending from server to client and then client to server will increase network traffic and the cookies may lost due to network errors.
* So, HTTP cookies are good for managing small piece of information across multiple requests and it is not a good technique to manage large amount of information.
* A cookie can be created in a servlet class like below.

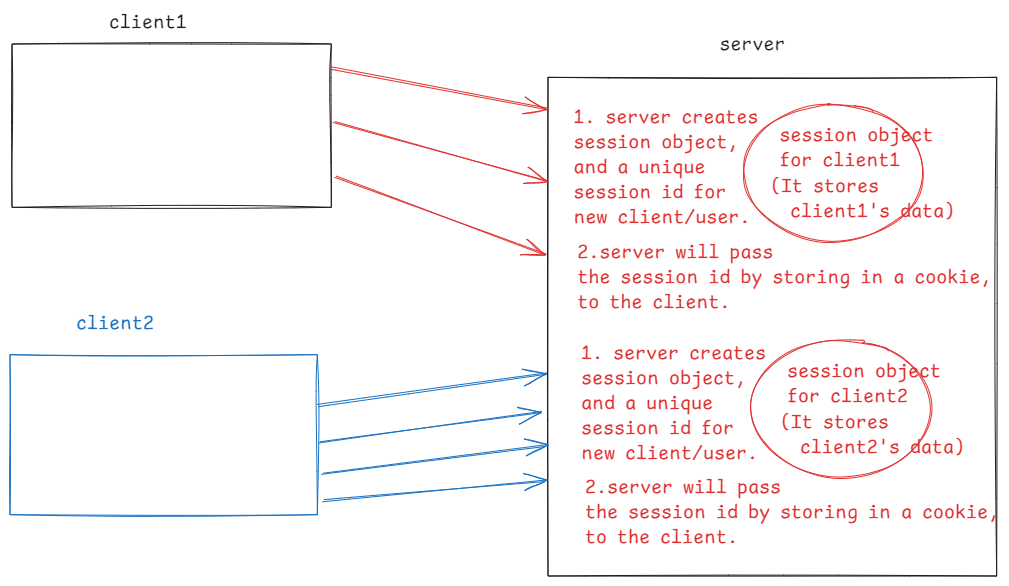
Cookie loginCookie = new Cookie(“username”, username);

loginCookie.setMaxAge(30 \* 60); //expires in 30 minutes

resp.addCookie(loginCookie); //add cookie to the response

HttpSession API

* HttpSession API is the most common way to handle session management in servlets.
* A HttpSession object is created at server, to store the user data across multiple requests, for each user/client.
* The server also creates a unique session id for each user/client and this id will be passed in a cookie to the client.
* When the client is sending subsequent requests, the client will also send the cookie with session id to the server. With this session id, the server will recognize the client as an existing user.
* The session object is created at server, for each client, but not for each request.
* Suppose, there are 2 clients, where client1 is sending 3 requests to server and client2 is sending 5 requests to server. So, the number of session objects created by the server is 2.



creating a session object:

. a session object can be created by calling getSession() method.

. when a user has sent first request, it means, the user is a new user then this getSession() method creates a new session object.

. when a user has sent next request, it means, the user is an existing user then this getSession() method retrieves the session object associated with that user.

HttpSession session = request.getSession();

. we can also pass boolean parameter to the getSession() method

. getSession(true) works like getSession() only.

. getSession(false) returns the session object of a user. If not exist then returns null.

storing or retrieving the data:

. session object stores data as key-value pairs.

. A user data can be stored in a session object, by calling

setAttribute(key, value) method.

. The data can be retrieved from a session object by calling

getAttribute(key) method.

. The data can be removed from a session object by calling

removeAttribute(key) method.

expiring a session object:

. a session object should be expired from a server, when a user

logout of the application or when a user is not coming back to

the server for some period.

. To remove the session object, when a user logout then call

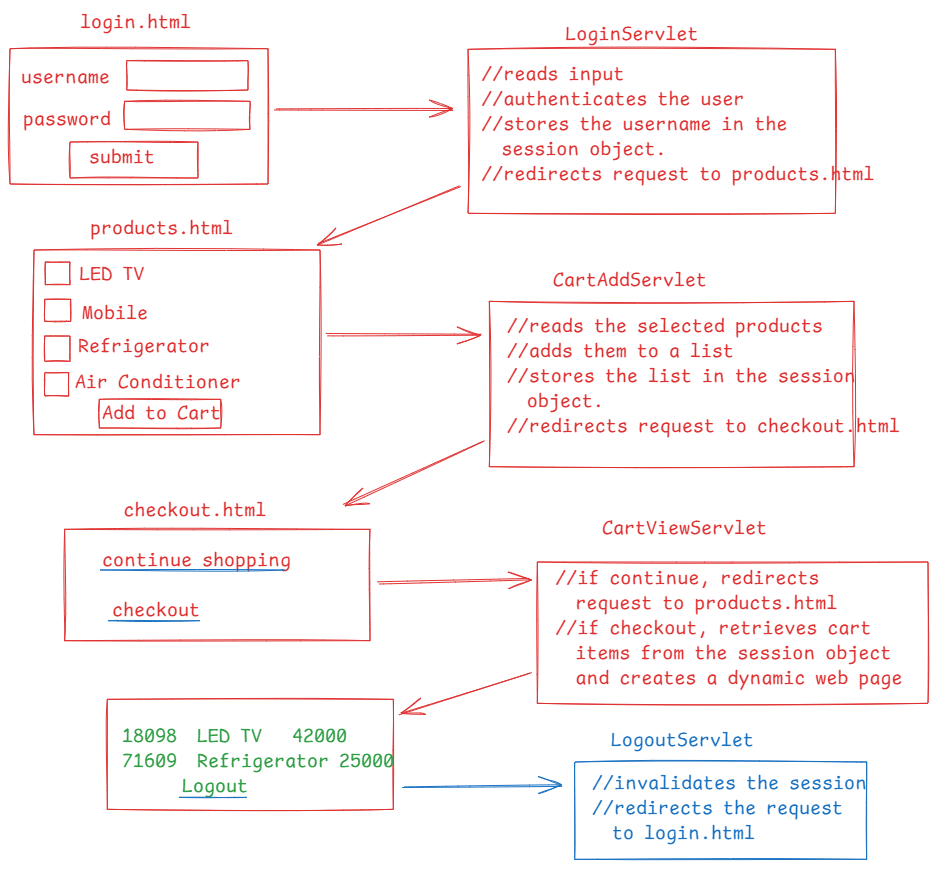
invalidate() method.

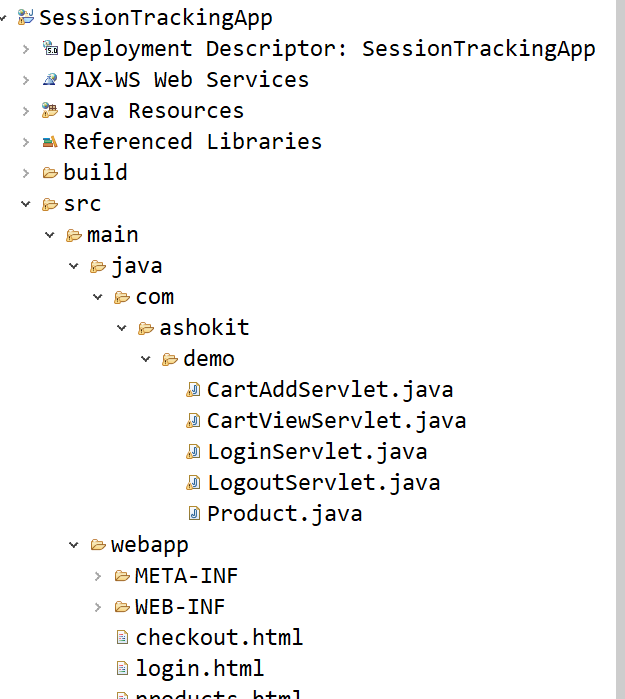
. The inactivity period can be set for a session, by calling

setMaxInactiveInterval() method.

for example,

session.setMaxInactiveInterval(120); // 2 minutes





login.html

-------------

<!DOCTYPE html>

<html>

<head>

<meta charset=*"UTF-8"*>

<title>Login Page</title>

</head>

<body>

<form action=*"./signin"* method=*"post"*>

<table>

<tr>

<td>Username </td>

<td> <input type=*"text"* name=*"username"*> </td>

</tr>

<tr>

<td>Password </td>

<td> <input type=*"password"* name=*"password"*> </td>

</tr>

</table>

<button type=*"submit"*>submit</button>

</form>

</body>

</html>

products.html

-------------

<!DOCTYPE html>

<html>

<head>

<meta charset=*"UTF-8"*>

<title>Products List</title>

</head>

<body>

<form action=*"./add-to-cart"*>

<input type=*"checkbox"* name=*"product"* value=*"p-101"*>LED TV <br>

<input type=*"checkbox"* name=*"product"* value=*"p-102"*>Mobile <br>

<input type=*"checkbox"* name=*"product"* value=*"p-103"*>Refrigerator <br>

<input type=*"checkbox"* name=*"product"* value=*"p-104"*>Air Conditioner <br>

<button type=*"submit"*>Add to Cart</button>

</form>

</body>

</html>

checkout.html

-------------

<!DOCTYPE html>

<html>

<head>

<meta charset=*"UTF-8"*>

<title>checkout page</title>

</head>

<body>

<a href = *"./view?option=continue"*>continue shopping</a> <br>

<a href = *"./view?option=checkout"*>checkout</a>

</body>

</html>

LoginServlet.java

-----------------

**package** com.ashokit.demo;

**import** java.io.IOException;

**import** jakarta.servlet.ServletException;

**import** jakarta.servlet.annotation.WebServlet;

**import** jakarta.servlet.http.HttpServlet;

**import** jakarta.servlet.http.HttpServletRequest;

**import** jakarta.servlet.http.HttpServletResponse;

**import** jakarta.servlet.http.HttpSession;

@WebServlet("/signin")

**public** **class** LoginServlet **extends** HttpServlet {

@Override

**protected** **void** doPost(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {

//read input from request

String username = req.getParameter("username");

String password = req.getParameter("password");

**if**(authenticate(username, password)) {

//create session

HttpSession session = req.getSession();

//add username to the session

session.setAttribute("username", username);

resp.sendRedirect("products.html");

}

**else** {

resp.sendRedirect("login.html");

}

}

**private** **boolean** authenticate(String username, String password) {

**return** username != **null** && !username.isEmpty() && password != **null** && !password.isEmpty();

}

}

Product.java

------------

**package** com.ashokit.demo;

**public** **class** Product {

**private** String id;

**private** String name;

**private** **double** price;

**public** Product(String id, String name, **double** price) {

**super**();

**this**.id = id;

**this**.name = name;

**this**.price = price;

}

**public** String getId() {

**return** id;

}

**public** **void** setId(String id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **double** getPrice() {

**return** price;

}

**public** **void** setPrice(**double** price) {

**this**.price = price;

}

}

CartAddServlet.java

-------------------

**package** com.ashokit.demo;

**import** java.io.IOException;

**import** java.util.ArrayList;

**import** java.util.List;

**import** jakarta.servlet.ServletException;

**import** jakarta.servlet.annotation.WebServlet;

**import** jakarta.servlet.http.HttpServlet;

**import** jakarta.servlet.http.HttpServletRequest;

**import** jakarta.servlet.http.HttpServletResponse;

**import** jakarta.servlet.http.HttpSession;

@WebServlet("/add-to-cart")

**public** **class** CartAddServlet **extends** HttpServlet {

**private** **static** List<Product> *productsList* = **null**;

**static** {

*productsList* = **new** ArrayList<>();

*productsList*.add(**new** Product("p-101", "LED TV", 21999.0));

*productsList*.add(**new** Product("p-102", "Mobile", 14999.0));

*productsList*.add(**new** Product("p-103", "Refrigerator", 26999.0));

*productsList*.add(**new** Product("p-104", "Air Conditioner", 41999.0));

}

@Override

**protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {

String[] ids = req.getParameterValues("product");

//retrieve the session

HttpSession session = req.getSession();

List<Product> cart = (List<Product>) session.getAttribute("cart");

**if**(cart == **null**) {

cart = **new** ArrayList<Product>();

}

**for** ( String id : ids ) {

Product selectedProduct = *productsList*.stream()

.filter(p -> p.getId().equals(id))

.findFirst()

.orElse(**null**);

**if**(selectedProduct != **null** ) {

cart.add(selectedProduct);

}

}

session.setAttribute("cart", cart);

resp.sendRedirect("checkout.html");

}

}

CartViewServlet.java

----------------------

**package** com.ashokit.demo;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** java.util.List;

**import** jakarta.servlet.ServletException;

**import** jakarta.servlet.annotation.WebServlet;

**import** jakarta.servlet.http.HttpServlet;

**import** jakarta.servlet.http.HttpServletRequest;

**import** jakarta.servlet.http.HttpServletResponse;

**import** jakarta.servlet.http.HttpSession;

@WebServlet("/view")

**public** **class** CartViewServlet **extends** HttpServlet {

@Override

**protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {

String str = req.getParameter("option");

**if** ( str.equals("continue") )

resp.sendRedirect("products.html");

**else** {

HttpSession session = req.getSession();

resp.setContentType("text/html");

PrintWriter out = resp.getWriter();

List<Product> lst = (List<Product>) session.getAttribute("cart");

String username = (String) session.getAttribute("username");

out.println("<html> <body>");

out.println("<h3> Hello " + username + "</h3>");

out.println("<br>");

out.println("Your cart items");

out.println("<hr>");

out.println("<ol>");

**for** ( Product p : lst ) {

out.println("<li>");

out.println(p.getId() + " " + p.getName() + " " + p.getPrice() );

out.println("</li>");

}

out.println("</ol>");

out.println("<br>");

out.println("<a href = './logout'> Logout </a>");

out.println("</body></html>");

out.close();

}

}

}

LogoutServlet.java

------------------

**package** com.ashokit.demo;

**import** java.io.IOException;

**import** jakarta.servlet.ServletException;

**import** jakarta.servlet.annotation.WebServlet;

**import** jakarta.servlet.http.HttpServlet;

**import** jakarta.servlet.http.HttpServletRequest;

**import** jakarta.servlet.http.HttpServletResponse;

**import** jakarta.servlet.http.HttpSession;

@WebServlet("/logout")

**public** **class** LogoutServlet **extends** HttpServlet {

@Override

**protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {

HttpSession session = req.getSession();

session.invalidate();

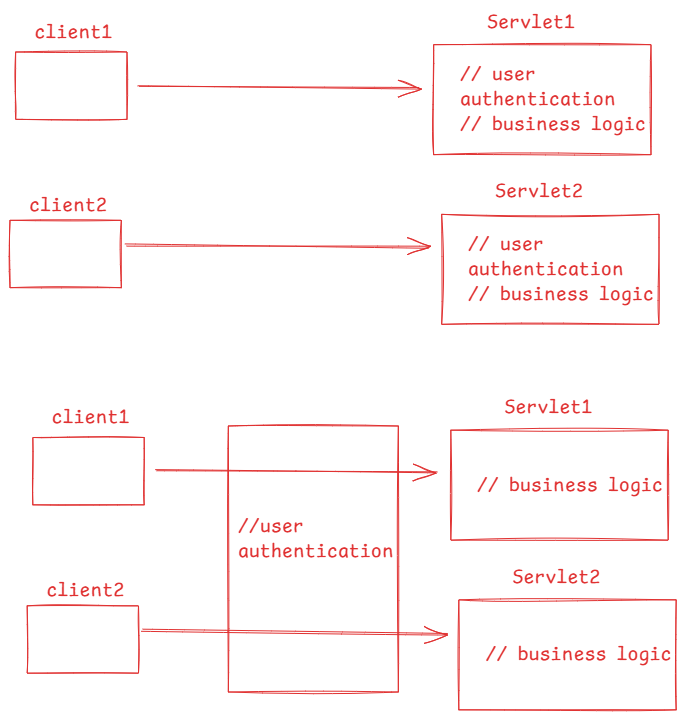
resp.sendRedirect("login.html");

}

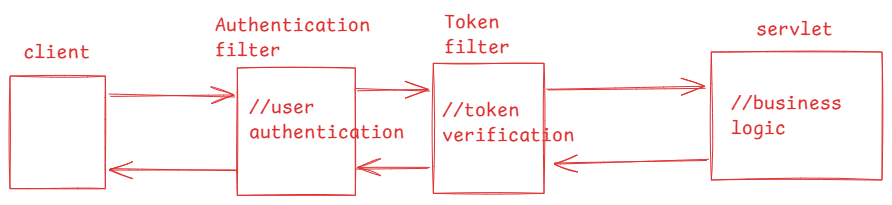
}

Filters in web application:

* Filters in a web application are used to handle the tasks that need to be executed before or after the core processing logic by a servlet.
* Filters provide a way to keep the secondary tasks like authentication, authorization, validation etc.. from the main business logic.
* Filters allow the execution of code both before a request reaches to the servlet or after a servlet is executed and before the response reaches to the client.
* Filters allow to define preprocessing and postprocessing tasks.
* Filters will provide resusability, where a filter can be across multiple servlets, without duplicating the code.



* create a separate filter to handle one secondary task. Don’t create a filter with mulitple secondary tasks.



creating a filter:

* A filter can be created in a web application by implementing Filter interface.
* Filter interface provides 3 abstract methods, which are called filter life cycle methods.

1. init
2. doFilter
3. destroy

* Like a servlet, a filter life cycle will also be managed by the web container only.
* The web container executes init and destroy methods for once. But executes doFilter method for every request.
* You have to define the pre/post processing code of a filter in the doFilter method.
* Like a servlet, a filter also need to be configured in the web.xml file, using <filter> and <filter-mapping> tags.
* We can also use annotation @WebFilter to avoid the configuration in web.xml file.

for example:

@WebFilter(“/srv1”)

public class MyFilter implements Filter {

@Override

public void init(FilterConfig config) throws ServletException {

}

@Override

public void doFilter(ServleRequest req, ServletResponse resp, FilterChain chain) throws ServletException, IOException {

//preprocessing code

chain.doFilter(req,resp);

//postprocessing code

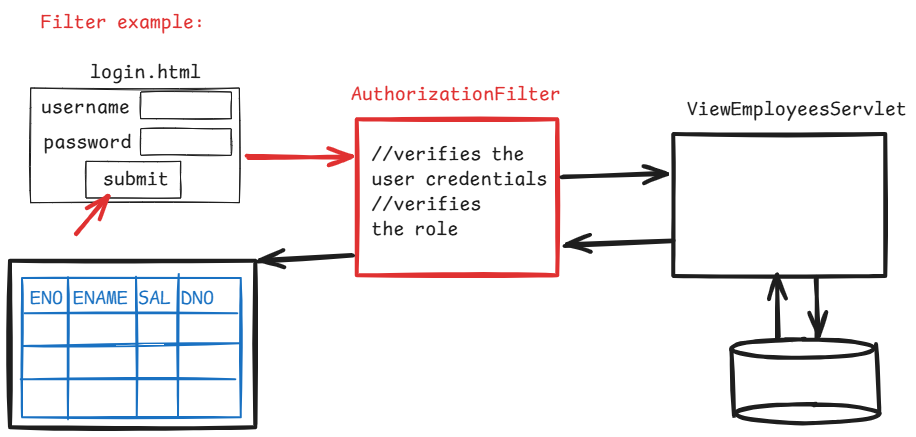
}

@Override

public void destroy() {

}

}



login.html

<!DOCTYPE html>

<html>

<head>

<meta charset=*"UTF-8"*>

<title>Login Page</title>

</head>

<body>

<form action=*"./view-servlet"* method=*"post"*>

<table>

<tr>

<td>Username </td>

<td> <input type=*"text"* name=*"username"*> </td>

</tr>

<tr>

<td>Password </td>

<td> <input type=*"password"* name=*"password"*> </td>

</tr>

</table>

<button type=*"submit"*>submit</button>

</form>

</body>

</html>

User.java

**package** com.example.model;

**public** **class** User {

**private** String username;

**private** String password;

**private** String role;

**public** User(String username, String password, String role) {

**super**();

**this**.username = username;

**this**.password = password;

**this**.role = role;

}

**public** String getUsername() {

**return** username;

}

**public** **void** setUsername(String username) {

**this**.username = username;

}

**public** String getPassword() {

**return** password;

}

**public** **void** setPassword(String password) {

**this**.password = password;

}

**public** String getRole() {

**return** role;

}

**public** **void** setRole(String role) {

**this**.role = role;

}

}

AuthorizationFilter.java

**package** com.example.filter;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** java.util.ArrayList;

**import** java.util.List;

**import** com.example.model.User;

**import** jakarta.servlet.Filter;

**import** jakarta.servlet.FilterChain;

**import** jakarta.servlet.FilterConfig;

**import** jakarta.servlet.ServletException;

**import** jakarta.servlet.ServletRequest;

**import** jakarta.servlet.ServletResponse;

**import** jakarta.servlet.annotation.WebFilter;

@WebFilter( value = "/view-servlet")

**public** **class** AuthorizationFilter **implements** Filter {

**private** List<User> lstUsers;

@Override

**public** **void** init(FilterConfig filterConfig) **throws** ServletException {

// **TODO** Auto-generated method stub

lstUsers = **new** ArrayList<>();

lstUsers.add(**new** User("John", "John@123", "CLERK"));

lstUsers.add(**new** User("Allen", "Allen@123", "ADMIN"));

lstUsers.add(**new** User("Miller", "Miller@123", "HELPER"));

lstUsers.add(**new** User("Mary", "Mary@123", "ADMIN"));

}

@Override

**public** **void** doFilter(ServletRequest req, ServletResponse resp, FilterChain chain)

**throws** IOException, ServletException {

// **TODO** Auto-generated method stub

//read the input

String username = req.getParameter("username");

String password = req.getParameter("password");

//using stream api(Java8)

User user = lstUsers.stream()

.filter(u -> u.getUsername().equals(username) &&

u.getPassword().equals(password) &&

u.getRole().equals("ADMIN"))

.findFirst()

.orElse(**null**);

**if**(user != **null**) {

chain.doFilter(req, resp);

}

**else** {

PrintWriter out = resp.getWriter();

resp.setContentType("text/html");

out.println("<html>");

out.println("<body>");

out.println("<h3>Username/password is mismatched or You are not an ADMIN </h3>");

out.println("<br>");

out.println("<a href = 'login.html'> click here </a>");

out.println("</body> </html>");

out.close();

}

}

}

ViewEmployeesServlet.java

package com.example.servlet;

import java.io.IOException;

import java.io.PrintWriter;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;

import jakarta.servlet.ServletException;

import jakarta.servlet.annotation.WebServlet;

import jakarta.servlet.http.HttpServlet;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.servlet.http.HttpServletResponse;

@WebServlet( value = "/view-servlet")

public class ViewEmployeesServlet extends HttpServlet {

@Override

protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

// TODO Auto-generated method stub

resp.setContentType("text/html");

PrintWriter out = resp.getWriter();

try {

Class.forName("com.mysql.cj.jdbc.Driver");

Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/test", "root", "root");

Statement stmt = conn.createStatement();

String query = "SELECT \* FROM EMP";

ResultSet rs = stmt.executeQuery(query);

out.println("<html> <body>");

out.println("<table border=1>");

out.println("<tr>");

out.println("<th>DEPTNO</th>");

out.println("<th>EMPNO</th>");

out.println("<th>SAL</th>");

out.println("<th>ENAME</th>");

out.println("</tr>");

while( rs.next() ) {

out.println("<tr>");

out.println("<td>" + rs.getInt(1) + "</td>");

out.println("<td>" + rs.getInt(2) + "</td>");

out.println("<td>" + rs.getDouble(3) + "</td>");

out.println("<td>" + rs.getString(4) + "</td>");

out.println("</tr>");

}

out.println("</table>");

out.println("</body> </html>");

rs.close();

stmt.close();

conn.close();

}

catch(Exception ex) {

ex.printStackTrace();

}

out.close();

}

}

ServletConfig object:

* ServletConfig object is created by the servlet container for each servlet during the initialization process.
* ServletConfig object is used to pass intialization parameters to a servlet.
* Lets say you have a servlet, which connects to a database.
* Your servlet might need database connection details like driver class, url, username and password. So, these details can be passed as init parameters.
* We can configure the init parameters to a servlet in the web.xml file with <init-param> tag or we can use initParams attribute in @WebServlet annotation.

web.xml

-------

<serlvet>

<servlet-name>DatabaseServlet</servlet-name>

<servlet-class>com.ashokit.DatabaseServlet</servlet-class>

<init-param>

<param-name>driver</param-name>

<param-value>com.mysql.cj.jdbc.Driver</param-value>

</init-param>

<init-param>

<param-name>url</param-name>

<param-value>jdbc:mysql://localhost:3306/test</param-

value>

</init-param>

. . . . . . .

. . . . . . .

</servlet>

Accessing ServletConfig in the servlet:

public class DatabaseServlet extends HttpServlet {

@Override

public void init(ServletConfig config) throws ServletException

{

String str1 = config.getInitParameter(“driver”);

String str2 = config.getInitParameter(“url”);

. . . . .

. . . . .

}

@Override

public void doGet(req,resp) throws ServletExcepiton,IOException {

//logic

}

}

ServletContext object:

---------------------

* ServletContext object is a global object, created by the servlet container, when a web application is deployed on a server.
* This ServletContext object is used to pass a global information to all the servlets in a web application.
* The ServletContext object is shared across all servlets in a web application.
* A servlet can access the ServletContext object using getServletContext() method.
* In web.xml file, <context-param> is used to configure the global parameters. It should be used under <web-app> tag.

<web-app>

<context-param>

<param-name>imageLocation</param-name>

<param-value>D:/Images/Local/dail.gif</param-value>

</context-param>

. . . . . .

. . . . . .

</web-app>

Accessing ServletContext in a servlet:

public class DashboardServlet extends HttpServlet {

@Override

public void doGet(req, resp) throws SE, IOE {

ServletContext context = getServletContext();

String path = context.getInitParameter(“imageLocation”);

. . . . .

}

}

