Tutorial 01

1. A Java program that increases a server's capabilities is called a servlet. Specifically created to manage requests and deliver answers in a server-side setting are servlets. They are frequently used in the development of dynamic web applications and are a component of the Java Platform, Enterprise Edition (Java EE).
2. There are two distinct methods for producing and distributing web content: static and dynamic websites. The way the material is created and presented to users is where the main differences reside.

Websites that are static:

1) Content Generation: Fixed, static content is the main feature of static websites. All users see the same material, which is created and saved as HTML files, regardless of how they interact with the website.

2) Page Loading: On a static website, every page is pre-made and saved as a static file. The server merely forwards the pre-existing HTML file to the user's browser upon receiving a page request.

3) Interactivity: The amount of interactivity on static webpages is constrained. While they can be used to convey information, they lack the responsiveness and dynamic nature of dynamic websites.

4)Creation and Provisioning:

Because static websites don't require databases or server-side scripting, they are frequently easier to develop. Static sites are also easy to host and can be hosted on a variety of platforms, such as content delivery networks (CDNs) or basic web servers.

5)As an illustration:

Small business websites, personal blogs, and brochure websites are frequently stagnant. HTML files must be manually edited for updates, as the content is rarely changed.

Websites that are dynamic:

1) material Generation: In reaction to user requests, dynamic websites create material instantly. User interactions, user preferences, or information obtained from a database may all influence the content.

2) Page Loading: Dynamic websites employ server-side technologies (such PHP, Python, or Java) to produce HTML content dynamically based on user inputs or other criteria, as opposed to serving pre-built HTML files.

3) Interactivity: More personalization and interactivity are available on dynamic websites. They are able to manage user inputs (such forms) and modify information according to user behavior, all while offering a more personalized and interesting user experience.

4) Development and Hosting: Creating dynamic websites requires database integration, server-side programming, and frequently more intricate coding. Servers that can run server-side scripts and handle databases host dynamic websites.

5) Examples of dynamic websites include social networking platforms, e-commerce sites, content management systems (CMS), and online applications. They are dependent on real-time updates, database interactions, and user input.

Variations:

1) Content Flexibility: While dynamic websites can create content dynamically based on user interactions and other factors, static websites have content that is fixed.

2) Interactivity: Compared to static websites, dynamic websites provide more user engagement and interactivity.

3) Complexity of Development: The use of databases, server-side scripting, and other dynamic components makes developing dynamic websites frequently more complex.

4) Hosting Requirements: Dynamic websites need servers that can run server-side scripts and manage databases, whereas static websites can be hosted on more basic servers or content delivery networks (CDNs).

5) Update Process: While dynamic websites can update material dynamically using server-side software, static websites must manually alter their HTML files.

1. 1. Goal:

GET: Used to get information from the server.

The query string contains parameters that are part of the URL.

Ideal for requests that are idempotent—that is, when they have no negative effects—and non-sensitive data.

POST: A protocol used to send data to a designated resource for processing.

The request body contains the parameters.

Ideal for requests with side effects (not idempotent), sensitive data, and big data collections.

2. Data Handling:

GET: A query string containing data is attached to the URL.

Data is seen in the address bar of the browser and is restricted by the length of the URL.

The request can be stored in bookmarks and may be cached.

POST: The request body contains data.

capable of handling bigger data volumes.

Security is improved since data is hidden from view in the URL.

Not bookmarkable or cached since it might contain sensitive information.

3. Safety:

GET: Sensitive information is less safe since the URL exposes data.

appropriate for read-only functions.

POST: Sensitive information can be transmitted more securely because no data is revealed in the URL.

Appropriate for write operations, which have the potential to alter server data.

04)

package servlets;

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

@WebServlet(name = "HelloServlet", urlPatterns = {"/hello"})

public class HelloServlet extends HttpServlet {

@Override

protected void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

response.setContentType("text/html;charset=UTF-8");

try (PrintWriter out = response.getWriter()) {

out.println("<!DOCTYPE html>");

out.println("<html>");

out.println("<head>");

out.println("<title>HelloServlet</title>");

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

out.println("<body>");

out.println("<h1>Hello, Servlet!</h1>");

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

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

}

}

}

05)

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

<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_4\_0.xsd"

version="4.0">

<!-- Servlet Configuration -->

<servlet>

<servlet-name>HelloServlet</servlet-name>

<servlet-class>servlets.HelloServlet</servlet-class>

</servlet>

<!-- Servlet Mapping -->

<servlet-mapping>

<servlet-name>HelloServlet</servlet-name>

<url-pattern>/hello</url-pattern>

</servlet-mapping>

</web-app>

06)

package servlets;

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

@WebServlet(name = "HelloServlet", urlPatterns = {"/hello"})

public class HelloServlet extends HttpServlet {

@Override

protected void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

// Set the content type of the response

response.setContentType("text/html;charset=UTF-8");

// Get the PrintWriter object to write the response

try (PrintWriter out = response.getWriter()) {

// Write the HTML response

out.println("<!DOCTYPE html>");

out.println("<html>");

out.println("<head>");

out.println("<title>HelloServlet</title>");

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

out.println("<body>");

out.println("<h1>Hello, World!</h1>");

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

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

}

}

}

07)

package servlets;

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

@WebServlet(name = "HelloServlet", urlPatterns = {"/hello"})

public class HelloServlet extends HttpServlet {

@Override

protected void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

// Set the content type of the response

response.setContentType("text/html;charset=UTF-8");

// Get the PrintWriter object to write the response

try (PrintWriter out = response.getWriter()) {

// Write the HTML response

out.println("<!DOCTYPE html>");

out.println("<html>");

out.println("<head>");

out.println("<title>HelloServlet</title>");

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

out.println("<body>");

out.println("<h1>Hello, World!</h1>");

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

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

}

}

@Override

protected void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

// Set the content type of the response

response.setContentType("text/html;charset=UTF-8");

// Get the PrintWriter object to write the response

try (PrintWriter out = response.getWriter()) {

// Write the HTML response for POST request

out.println("<!DOCTYPE html>");

out.println("<html>");

out.println("<head>");

out.println("<title>HelloServlet</title>");

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

out.println("<body>");

out.println("<h1>Hello, POST!</h1>");

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

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

}

}

}

08)

<%@ page contentType="text/html;charset=UTF-8" language="java" %>

<html>

<head>

<title>Index Page</title>

</head>

<body>

<h1>Welcome to the Index Page</h1>

<!-- Add a button that redirects to the HelloServlet -->

<form action="hello" method="post">

<input type="submit" value="Click me to go to HelloServlet">

</form>

</body>

</html>

09)

To deploy to GlassFish Server, follow these steps: 1) Create the Project: In NetBeans, right-click on your project (such as "HelloWeb").

To gather and package your project, select "Clean and Build".

2) To deploy your project to GlassFish, right-click on it.

To deploy your project to the GlassFish Server, select "Deploy".

3) Launch a Web Browser: Launch the web browser of your choice.

4) Go to the URL for the Servlet:

Enter the URL where your servlet is mapped in the address bar of your browser. The URL would be http://localhost:8080/HelloWeb/hello, for instance, if your servlet is mapped to /hello (change "HelloWeb" with the name of your project).

Check the Servlet Message:

Depending on whether you did a GET or POST request, accessing the URL should display the message "Hello, World!" or "Hello, POST!" if you followed the directions in the earlier responses.

Accessing the root URL (e.g., http://localhost:8080/HelloWeb/) should display the index page if you produced an index.jsp page with a button to redirect to the servlet. Clicking the button should then redirect to your servlet, showing the relevant message.

Troubleshooting:

Look for error messages in the GlassFish Server logs in NetBeans if you run into any problems.

Verify that GlassFish Server is launched and set up correctly in NetBeans.