

SERVLET

1. Explain the lifecycle of a servlet.
2. Describe the classes and interfaces of javax.servlet package.
3. Describe the classes and interfaces of javax.servlet.http package.
4. How do you use Tomcat for Servlet development?

programs

5. With an example program Explain the mechanism of SessionTracking in Servlets?
6. Write a JAVA Servlet Program to implement a dynamic HTML using Servlet (user name and Password should be accepted using HTML and displayed using a Servlet).
7. Write a JAVA Servlet Program to Auto Web Page Refresh (Consider a webpage which is displaying Date and time).
8. Write a Servlet program to read data from a HTML form (gender data from radio buttons and colors data from check boxes) and display.
9. Write a JAVA Servlet Program using cookies to remember user preferences.
10. Explain how do you handle Http Requests and response with an example program?

Part-2

11. Differentiate between Get and Post methods.
12. What is Cookie? Explain creation of Cookie and retrieving information from cookie using code snippets.
13. Differentiate between JSP and Servlets.
14. What is JSP? What are different JSP tags demonstrate with an example.

STRINGS

1. Explain the character extraction methods.
2. Explain the following: equals() and equalsIgnoreCase(), regionMatches(), startsWith() and endsWith(), equals vs ==
3. Write a program to check whether a string is palindrome or not.
4. Write a java program to check whether 2 strings are anagram or not.
5. What is String? Explain all String Constructors available with code snippets.
6. Explain how do you perform searching Strings along with an example program.

1. Explain the lifecycle of a servlet.

1) What is servlet.

1. Java Servlets are programs that run on a Web or Application server and act as a middle layer between a requests coming from a Web browser or other HTTP client and databases or applications on the HTTP server.
2. Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.
3. Java Servlets often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But Servlets offer several advantages in comparison with the CGI.

Servlets offer several advantages

- Performance is significantly better.
- Servlets execute within the address space of a Web server. It is not necessary to create a separate process to handle each client request.
- Servlets are platform-independent because they are written in Java.
- Java security manager on the server enforces a set of restrictions to protect the resources on a server machine. So servlets are trusted.
- The full functionality of the Java class libraries is available to a servlet. It can communicate with applets, databases, or other software via the sockets and RMI mechanisms that you have seen already.

2) servlet life cycle

A servlet life cycle can be defined as the entire process from its creation till the destruction. The following are the paths followed by a servlet.

1. The servlet is initialized by calling the **init()** method.
2. The servlet calls **service()** method to process a client's request.
3. The servlet is terminated by calling the **destroy()** method.
4. Finally, servlet is garbage collected by the garbage collector of the JVM.

I. The init() Method

- The init method is called only once. It is called only when the servlet is created, and not called for any user requests afterwards. So, it is used for one-time initializations, just as with the init method of applets.
- The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started.
- When a user invokes a servlet, a single instance of each servlet gets created, with each user request resulting in a new thread that is handed off to doGet or doPost as appropriate. The init() method simply creates or loads some data that will be used throughout the life of the servlet.
- The init method definition looks like this –

```
public void init() throws ServletException {  
    // Initialization code...  
}
```

II. The service() Method

- The service() method is the main method to perform the actual task. The servlet container (i.e. web server) calls the service() method to handle requests coming from the client(browsers) and to write the formatted response back to the client.
- Each time the server receives a request for a servlet, the server spawns a new thread and calls service. The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.
- Here is the signature of this method –

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

The doGet() and doPost() are most frequently used methods with in each service request. Here is the signature of these two methods.

I. The doGet() Method

- A GET request results from a normal request for a URL or from an HTML form that has no METHOD specified and it should be handled by doGet() method.

```
public void doGet(HttpServletRequest request, HttpServletResponse response)  
throws ServletException, IOException {  
    // Servlet code  
}
```

II. The doPost() Method

- A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.

```
public void doPost(HttpServletRequest request, HttpServletResponse response)  
throws ServletException, IOException {  
    // Servlet code  
}
```

III. The destroy() Method

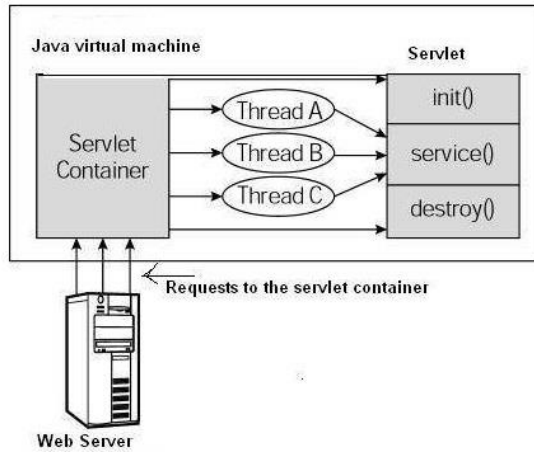
- The destroy() method is called only once at the end of the life cycle of a servlet. This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.
- After the destroy() method is called, the servlet object is marked for garbage collection. The destroy method definition looks like this –

```
public void destroy() {  
    // Finalization code...  
}
```

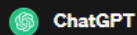
3) Architecture Diagram

The following figure depicts a typical servlet life-cycle scenario.

- First the HTTP requests coming to the server are delegated to the servlet container.
- The servlet container loads the servlet before invoking the service() method.
- Then the servlet container handles multiple requests by spawning multiple threads, each thread executing the service() method of a single instance of the servlet.



4. How do you use Tomcat for Servlet development?



ChatGPT

To create a Servlet application using Apache Tomcat server, follow these steps:

1. Create Directory Structure:

- Sun Microsystems defines a unique directory structure. Create the necessary directories for your application.

2. Creating a Servlet:

- There are three ways to create a servlet:
 - By extending `HttpServlet` class.
 - By extending `GenericServlet` class.
 - By implementing the `Servlet` interface.

Example of extending `HttpServlet` class:

```
java
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class MyServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response) {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<h1>Hello Readers</h1>");
    }
}
```

3. Compiling a Servlet:

- Download `javax.servlet-api.jar` file.
- Paste the `javax.servlet-api.jar` file inside `Java\jdk\jre\lib\ext` directory.
- Compile the Servlet class and paste the class file into `WEB-INF/classes/` directory.

4. Create Deployment Descriptor:

- Deployment Descriptor (DD) is an XML document used by the Web Container to run Servlets and JSP pages.
- Create a `web.xml` file:

```
xml
<web-app>
  <servlet>
    <servlet-name>MyServlet</servlet-name>
    <servlet-class>MyServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>MyServlet</servlet-name>
    <url-pattern>/hello</url-pattern>
  </servlet-mapping>
</web-app>
```

5. Starting Tomcat Server:

- Set `JAVA_HOME` or `JRE_HOME` in environment variables.
- Go to My Computer properties -> Click on advanced tab -> environment variables -> Click on the new tab of user variable -> Write `JAVA_HOME` in variable name and paste the path of JDK folder in variable value -> ok.

6. Run Servlet Application:

- Open a browser and type `http://localhost:8080/First/hello`.

Make sure to replace `First` with your application context path, and `hello` with the appropriate URL pattern mapped in your `web.xml` file.



5. With an example program Explain the mechanism of SessionTracking in Servlets?

Session tracking in servlets refers to the mechanism of maintaining state or data associated with a particular user across multiple HTTP requests. Since HTTP itself is a stateless protocol, session tracking is necessary for web applications to remember user-specific information or interactions throughout a browsing session.

There are several ways to implement session tracking in servlets, including:

1. **Cookies:** A cookie is a small piece of data sent from a website and stored on the user's computer by the user's web browser while the user is browsing. Cookies can be used to store session identifiers or other user-specific information.
2. **URL Rewriting:** Session IDs can be appended to URLs as query parameters, allowing the server to identify and associate requests with a specific session.
3. **HTTP Session:** The `HttpSession` object provided by the servlet container allows developers to store session-specific data that is accessible across multiple requests from the same client.

An example program in Java servlets that demonstrates the mechanism of session tracking. In this example, we'll create a simple servlet that uses session tracking to count the number of times a user has visited a webpage.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class VisitCounterServlet extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        // Get the current session or create a new one if it doesn't exist
        HttpSession session = request.getSession(true);

        // Get the visit count from the session, or initialize it to 0 if it doesn't
        Integer visitCount = (Integer) session.getAttribute("visitCount");
        if (visitCount == null) {
            visitCount = 0;
        }

        // Increment the visit count
        visitCount++;

        // Store the updated visit count in the session
        session.setAttribute("visitCount", visitCount);

        // Set response content type
        response.setContentType("text/html");

        // Create PrintWriter object to write HTML response
        PrintWriter out = response.getWriter();

        // Construct the HTML response
        out.println("<html><head><title>Visit Counter Servlet</title></head><body>");
        out.println("<h2>Visit Counter</h2>");
        out.println("<p>You have visited this page " + visitCount + " times.</p>");
        out.println("</body></html>");
    }
}
```

Explanation of the mechanism:

1. **Session Creation:** When a user accesses the servlet for the first time, `request.getSession(true)` is called. This method either returns the current `HttpSession` object associated with the request or creates a new one if it doesn't exist.
2. **Session Data Retrieval:** We retrieve the `visitCount` attribute from the session using `session.getAttribute("visitCount")`. This attribute stores the number of times the user has visited the webpage.
3. **Increment Visit Count:** We increment the `visitCount` by 1.
4. **Store Updated Data:** We store the updated `visitCount` back into the session using `session.setAttribute("visitCount", visitCount)`.
5. **HTML Response Generation:** We generate an HTML response using a `PrintWriter` object, displaying the current visit count to the user.

Each time the user accesses the servlet, the visit count is incremented and stored in the session, allowing the servlet to keep track of the user's visits across multiple requests. This is the essence of session tracking in servlets.

6. Write a JAVA Servlet Program to implement a dynamic HTML using Servlet

(user name and Password should be accepted using HTML and displayed using a Servlet).

Below is a basic Java servlet program to accept a username and password using HTML form and display them using a servlet:

```
java Copy code

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class LoginServlet extends HttpServlet {
    public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        String username = request.getParameter("username");
        String password = request.getParameter("password");

        out.println("<html>");
        out.println("<head>");
        out.println("<title>Login Details</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h2>Entered Login Details</h2>");
        out.println("<p>Username: " + username + "</p>");
        out.println("<p>Password: " + password + "</p>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

Make sure you have servlet-api.jar in your classpath. You can compile the servlet using a Java compiler.

To create a simple HTML form for entering the username and password, you can create a file named login.html:



To create a simple HTML form for entering the username and password, you can create a file named login.html:

```
html Copy code

<!DOCTYPE html>
<html>
<head>
    <title>Login</title>
</head>
<body>
    <form action="LoginServlet" method="post">
        <label for="username">Username:</label>
        <input type="text" id="username" name="username"><br><br>
        <label for="password">Password:</label>
        <input type="password" id="password" name="password"><br><br>
        <input type="submit" value="Submit">
    </form>
</body>
</html>
```

7. Write a JAVA Servlet Program to Auto Web Page Refresh(Consider a webpage which is displaying Date and time).

Same as 5th answer program

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class VisitCounterServlet extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        // Get the current session or create a new one if it doesn't exist
        HttpSession session = request.getSession(true);

        // Get the visit count from the session, or initialize it to 0 if it doesn't
        Integer visitCount = (Integer) session.getAttribute("visitCount");
        if (visitCount == null) {
            visitCount = 0;
        }

        // Increment the visit count
        visitCount++;

        // Store the updated visit count in the session
        session.setAttribute("visitCount", visitCount);

        // Set response content type
        response.setContentType("text/html");

        // Create PrintWriter object to write HTML response
        PrintWriter out = response.getWriter();

        // Construct the HTML response
        out.println("<html><head><title>Visit Counter Servlet</title></head><body>");
        out.println("<h2>Visit Counter</h2>");
        out.println("<p>You have visited this page " + visitCount + " times.</p>");
        out.println("</body></html>");

    }
}
```

8. Write a Servlet program to read data from a HTML form

(gender data from radio buttons and colors data from check boxes) and display.

Here's a simple example of a Servlet program that reads data from an HTML form containing radio buttons for gender selection and checkboxes for selecting colors. We'll also provide the 'index.html' code for the form.

index.html:

```
html Copy code

<!DOCTYPE html>
<html>
<head>
  <title>Form</title>
</head>
<body>
  <h2>Enter your details</h2>
  <form action="FormDataServlet" method="post">
    Gender:<br>
    <input type="radio" name="gender" value="male">Male<br>
    <input type="radio" name="gender" value="female">Female<br><br>

    Favorite Colors:<br>
    <input type="checkbox" name="color" value="red">Red<br>
    <input type="checkbox" name="color" value="blue">Blue<br>
    <input type="checkbox" name="color" value="green">Green<br><br>

    <input type="submit" value="Submit">
  </form>
</body>
</html>
```

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class FormDataServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        // Set response content type
        response.setContentType("text/html");

        // Get parameters from the form
        String gender = request.getParameter("gender");
        String[] colors = request.getParameterValues("color");

        // Create PrintWriter object to write HTML response
        PrintWriter out = response.getWriter();

        // Construct the HTML response
        out.println("<html><head><title>Form Data</title></head><body>");
        out.println("<h2>Form Data</h2>");
        out.println("<p>Gender: " + gender + "</p>");
        out.println("<p>Favorite Colors:</p>");
        out.println("<ul>");
        if (colors != null) {
            for (String color : colors) {
                out.println("<li>" + color + "</li>");
            }
        } else {
            out.println("<li>No colors selected</li>");
        }
        out.println("</ul>");
        out.println("</body></html>");
    }
}
```


9. Write a JAVA Servlet Program using cookies to remember user preferences.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class CookieServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
        // Create a cookie
        Cookie cookie = new Cookie("username", "John Doe");
        // Set the cookie's maximum age to 1 hour
        cookie.setMaxAge(60 * 60);
        // Add the cookie to the response
        response.addCookie(cookie);
        // Get the cookies from the request
        Cookie[] cookies = request.getCookies();
        // Print the cookies to the console
        for (Cookie c : cookies) {
            System.out.println(c.getName() + ": " + c.getValue());
        }
    }
}
```

10. Explain how do you handle Http Requests and response with an example program?

Handling HTTP Requests and Responses

When we talk about handling HTTP requests and responses in Java Servlets, we're essentially talking about how a Java application running on a web server interacts with the web browser (or any other client) that's sending requests to it.

1. HTTP Requests:

- When a user interacts with a web application (like filling out a form, clicking a button, etc.), their web browser sends an HTTP request to the server hosting the application.
- Servlets handle these requests using methods like **doGet()**, **doPost()**, etc., which correspond to different types of HTTP requests like GET, POST, etc.
- For example, when you submit a form on a web page, the browser sends a POST request to the server, and the servlet's **doPost()** method is called to handle that request.

2. HTTP Responses:

- Once the server receives an HTTP request, it generates an appropriate response.
- This response typically includes HTML content, which is then rendered by the web browser.
- Servlets generate this response dynamically based on the request parameters, user input, etc.

4) Example:

Let's consider a simple scenario where we have a web page with a form allowing users to select a color. When the user submits the form, the selected color is displayed back to them.

1. Handling HTTP GET Requests:

- In our example, we have a servlet (**ColorGetServlet**) that handles HTTP GET requests.
- When the user submits the form, their browser sends a GET request to the servlet.
- The servlet's **doGet()** method retrieves the selected color from the request parameters and generates an HTML response displaying that color.
- This response is then sent back to the user's browser, which renders it.

2. Handling HTTP POST Requests:

- Similarly, we have another servlet (**ColorPostServlet**) that handles HTTP POST requests.
- When the user submits the form with the POST method explicitly specified, their browser sends a POST request to the servlet.
- The servlet's **doPost()** method retrieves the selected color from the request parameters and generates an HTML response displaying that color.
- Again, this response is sent back to the user's browser for rendering.

In both cases, the servlets dynamically generate responses based on the user input and send them back to the browser, which then displays the results to the user.

```
@WebServlet("/SimpleServlet")
public class SimpleServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // Set response content type
        response.setContentType("text/html");

        // Actual logic to respond to the request
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Simple Servlet Example</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Hello, GET request!</h1>");
        out.println("</body>");
        out.println("</html>");
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // Set response content type
        response.setContentType("text/html");

        // Actual logic to respond to the request
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Simple Servlet Example</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Hello, POST request!</h1>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

```
html
Copy code

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Simple Servlet Example</title>
</head>
<body>
  <h1>Simple Servlet Example</h1>
  <h2>GET Request:</h2>
  <form action="SimpleServlet" method="GET">
    <input type="submit" value="Send GET Request">
  </form>

  <h2>POST Request:</h2>
  <form action="SimpleServlet" method="POST">
    <input type="submit" value="Send POST Request">
  </form>
</body>
</html>
```