Handling_Get_and_Post_Request

June 18, 2024

1 Explain handling in HTTP get and post request in servlet with example.

In a Servlet, HTTP GET and POST requests are handled by the `doGet()` and `doPost()` methods respectively. These methods are called by the `service()` method of the `HttpServlet` class.

Here's a simple example of a Servlet that handles both GET and POST requests:

[ ]:
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class MyServlet extends HttpServlet {

    // Method to handle GET method request.
    public void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

      PrintWriter out = response.getWriter();
      String title = "Using GET Method to Read Form Data";
      String name = request.getParameter("name");

      out.println("<html>\n" +
                  "<head><title>" + title + "</title></head>\n" +
                  "<body>\n" +
                  "<h1 align=\"center\">" + title + "</h1>\n" +
                  "<ul>\n" +
                  "  <li><b>Name</b>: " + name + "\n" +
                  "</ul>\n" +
                  "</body></html>");
    }

    // Method to handle POST method request.
    public void doPost(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

      doGet(request, response);
    }
```

```
}
```

In this example, the `doGet()` method is used to read form data and produce an HTML page that displays the submitted data. The `doPost()` method simply calls `doGet()`, so this servlet will handle POST requests in the same way as GET requests.

The `HttpServletRequest` object `request` is used to get the data from the user, and the `HttpServletResponse` object `response` is used to create the response.

The `getParameter()` method of `HttpServletRequest` is used to read the form data. In this case, it's reading a form field named "name".

The `PrintWriter` object `out` is used to send the output back to the client. It's creating an HTML page that displays the submitted name.

To use this servlet, you would need to configure it in your `web.xml` file and create an HTML form that submits data to the URL mapped to this servlet. The form could use either the GET or POST method.

2 Explain handling in HTTP get request in servlet with example.

In a Servlet, HTTP GET requests are handled by the `doGet()` method. This method is called by the `service()` method of the `HttpServlet` class when a GET request is received.

Here's a simple example of a Servlet that handles GET requests:

[ ]:
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class MyServlet extends HttpServlet {

    // Method to handle GET method request.
    public void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

      // Set response content type and other response header fields first
      PrintWriter out = response.getWriter();
      // Then write the data of the response
      String title = "Using GET Method to Read Form Data";
      // Get the name passed in the request
      String name = request.getParameter("name");

      out.println("<html>\n" +
                  "<head><title>" + title + "</title></head>\n" +
                  "<body>\n" +
                  "<h1 align=\"center\">" + title + "</h1>\n" +
                  "<ul>\n" +
                  "  <li><b>Name</b>: " + name + "\n" +
```

```
                "</ul>\n" +
                "</body></html>");
    }
}
```

In this example, the `doGet()` method is used to read form data and produce an HTML page that displays the submitted data.

The `HttpServletRequest` object `request` is used to get the data from the user, and the `HttpServletResponse` object `response` is used to create the response.

The `getParameter()` method of `HttpServletRequest` is used to read the form data. In this case, it's reading a form field named "name".

The `PrintWriter` object `out` is used to send the output back to the client. It's creating an HTML page that displays the submitted name.

To use this servlet, you would need to configure it in your `web.xml` file and create an HTML form that submits data to the URL mapped to this servlet. The form should use the GET method.

3 Explain handling in HTTP post request in servlet with example.

HTTP POST requests in a servlet are handled by the `doPost()` method. This method is called by the `service()` method of the `HttpServlet` class when a POST request is received.

Here's how you can modify your existing servlet to handle POST requests:

[ ]:
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class MyServlet extends HttpServlet {

    // Method to handle GET method request.
    public void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

      processRequest(request, response);
    }

    // Method to handle POST method request.
    public void doPost(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

      processRequest(request, response);
    }

    private void processRequest(HttpServletRequest request, HttpServletResponse␣
 ↪response)
```

```
        throws ServletException, IOException {

        // Set response content type and other response header fields first
        PrintWriter out = response.getWriter();
        // Then write the data of the response
        String title = "Using GET or POST Method to Read Form Data";
        // Get the name passed in the request
        String name = request.getParameter("name");

        out.println("<html>\n" +
                    "<head><title>" + title + "</title></head>\n" +
                    "<body>\n" +
                    "<h1 align=\"center\">" + title + "</h1>\n" +
                    "<ul>\n" +
                    "  <li><b>Name</b>: " + name + "\n" +
                    "</ul>\n" +
                    "</body></html>");
    }
}
```

In this example, both `doGet()` and `doPost()` methods call a `processRequest()` method, which handles the request and generates the response. This way, the servlet can handle both GET and POST requests in the same way.

The `HttpServletRequest` object `request` is used to get the data from the user, and the `HttpServletResponse` object `response` is used to create the response.

The `getParameter()` method of `HttpServletRequest` is used to read the form data. In this case, it's reading a form field named "name".

The `PrintWriter` object `out` is used to send the output back to the client. It's creating an HTML page that displays the submitted name.

To use this servlet, you would need to configure it in your `web.xml` file and create an HTML form that submits data to the URL mapped to this servlet. The form could use either the GET or POST method.

4 Explain RequestDispatcher. Also describe different ways to get the object of RequestDispatcher.

The `RequestDispatcher` interface in Java Servlet provides the facility of dispatching the request to another resource it may be html, servlet or jsp. This interface can also be used to include the content of another resource, which can be useful in servlet collaboration.

There are two methods defined in the `RequestDispatcher` interface:

1. `void forward(ServletRequest request, ServletResponse response)`: Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.

2. `void include(ServletRequest request, ServletResponse response)`: Includes the content of a resource (servlet, JSP page, HTML file) in the response.

There are two ways to get the object of `RequestDispatcher`:

1. **Using `request.getRequestDispatcher(String path)` method:** This method returns the object of `RequestDispatcher`. The `path` specified is relative to the current request, and can be relative to the current context root or to the current servlet.

```
[ ]: RequestDispatcher rd = request.getRequestDispatcher("servlet2");
```

2. **Using `ServletContext.getRequestDispatcher(String path)` method:** This method returns the object of `RequestDispatcher`. The `path` specified must be relative to the current context root.

```
[ ]: ServletContext context = getServletContext();
RequestDispatcher rd = context.getRequestDispatcher("/servlet2");
```

In both cases, the `RequestDispatcher` object can then be used to forward the request to another resource, like so:

```
[ ]: rd.forward(request, response);
```

Or to include the content of another resource in the response:

```
[ ]: rd.include(request, response);
```

5 Write a servlet program for servlet login and logout using cookies.

Here's a simple example of a Servlet program that handles login and logout using cookies. This example uses two servlets: `LoginServlet` and `LogoutServlet`.

```
[ ]: // LoginServlet.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class LoginServlet extends HttpServlet {

    protected void doPost(HttpServletRequest request, HttpServletResponse␣
 ↪response)
            throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        String name = request.getParameter("name");
        String password = request.getParameter("password");
```

```
        if (password.equals("admin123")) {
            out.print("You are successfully logged in!");
            out.print("<br>Welcome, " + name);

            // Creating cookies and adding them to the response object
            Cookie ck = new Cookie("name", name);
            response.addCookie(ck);
        } else {
            out.print("Sorry, username or password error!");
            request.getRequestDispatcher("login.html").include(request,␣
 ↪response);
        }

        out.close();
    }
}
```

[ ]:
```
// LogoutServlet.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class LogoutServlet extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse␣
 ↪response)
            throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        request.getRequestDispatcher("link.html").include(request, response);

        Cookie ck = new Cookie("name", "");
        ck.setMaxAge(0);
        response.addCookie(ck);

        out.print("You are successfully logged out!");
    }
}
```

In this example, `LoginServlet` checks if the password is "admin123". If it is, it creates a cookie with the username and sends it to the client. If the password is incorrect, it includes the login page in the response.

`LogoutServlet` invalidates the cookie by setting its maximum age to 0, effectively logging the user out.

You would also need to create the `login.html` and `link.html` files, and configure the servlets

in your `web.xml` file. The `login.html` file should contain a form that POSTs the username and password to `LoginServlet`, and the `link.html` file should contain a link to `LogoutServlet`.

6 Thank You!