VISVESVARAYA TECHNOLOGICAL UNIVERSITY
BELAGAVI-590018

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Activity Report on

“Reading Servlet Parameter”

Advanced Java (BCS613D)

Submitted by

Ms. Archana Nadakattin **2KA22CS007**

Under the Guidance of

Mrs. Rajeshwari S.G

SMT.KAMALA AND SHRI VENKAPPA M AGADI
COLLEGE OF ENGINEERING AND TECHNOLOGY
LAXMESHWAR-582116

AY 2024-2025

Reading Servlet Parameter

Introduction

In Java web development, servlets play a crucial role in handling client requests and generating dynamic responses. A fundamental aspect of servlet programming is the ability to read parameters sent by clients, typically through HTTP requests. These parameters can be passed via query strings in URLs, form data in POST requests, or even as part of the URL path. Understanding how to efficiently retrieve and process these parameters is essential for building robust web applications.

1. Retrieving Single Parameters

The most straightforward method to retrieve a single parameter is by using the getParameter(String name) method provided by the HttpServletRequest interface. This method returns the value of the specified parameter as a `String`. If the parameter does not exist, it returns null.

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

2. Retrieving Multiple Values for a Parameter

When a parameter can have multiple values (e.g., checkboxes in a form), the getParameterValues(String name) method is appropriate. It returns an array of String objects containing all the values for the specified parameter.

```
String[] selectedItems = request.getParameterValues("items");
```

Note: If the parameter does not exist, this method returns null. It's important to handle this case to avoid NullPointerException.

3. Enumerating All Parameters

To retrieve all parameters sent with the request, the getParameterNames() method can be used. This method returns an Enumeration of all parameter names. You can then iterate over this enumeration to access each parameter's value.

```
Enumeration<String> paramNames = request.getParameterNames();
while (paramNames.hasMoreElements()) {
    String paramName = paramNames.nextElement();
    String paramValue = request.getParameter(paramName);
    // Process each parameter
}
```

Alternatively, the getParameterMap() method returns a Map<String, String[]> containing all parameter names and their corresponding values.

```
Map<String, String[]> paramMap = request.getParameterMap();
for (Map.Entry<String, String[]> entry : paramMap.entrySet()) {
    String paramName = entry.getKey();
    String[] paramValues = entry.getValue();
    // Process each parameter
}
```

4. Handling Form Data

When dealing with form submissions, parameters are typically sent as part of the request body. It's important to note that once the request body is read using methods like getReader() or getInputStream(), it cannot be read again. Therefore, it's advisable to access form parameters before reading the request body.

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

5. Handling URL Path Parameters

In some cases, parameters may be embedded within the URL path, especially in RESTful web services. These parameters can be accessed using methods like getPathInfo() or getServletPath().

```
String pathInfo = request.getPathInfo();
// Process path parameters
```

An Example Code On Reading Servlet Parameter:

HTML Form (index.html)

This form collects a user's name and age and sends the data to the servlet using the POST method.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>User Information Form</title>
</head>
<body>
    <h2>User Information</h2>
    <form action="UserInfoServlet" method="POST">
        <label for="name">Name:</label>
        <input type="text" id="name" name="name" required><br><br>
        <label for="age">Age:</label>
        <input type="number" id="age" name="age" required><br><br>
        <input type="submit" value="Submit">
    </form>
</body>
</html>
```

Servlet Code (UserInfoServlet.java)

This servlet processes the form data and displays the user's name and age.

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

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

        // Get the PrintWriter object to send the response
        PrintWriter out = response.getWriter();

        // Retrieve the parameters from the request
        String name = request.getParameter("name");
        String age = request.getParameter("age");

        // Generate the HTML response
        out.println("<html><body>");
        out.println("<h2>User Information</h2>");
        out.println("<p>Name: " + name + "</p>");
        out.println("<p>Age: " + age + "</p>");
        out.println("</body></html>");
    }
}
```

Web Deployment Descriptor (web.xml)

This XML configuration maps the servlet to a URL pattern.

Xml

```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
             http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
         version="2.4">
    <servlet>
        <servlet-name>UserInfoServlet</servlet-name>
        <servlet-class>UserInfoServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>UserInfoServlet</servlet-name>
        <url-pattern>/UserInfoServlet</url-pattern>
    </servlet-mapping>
</web-app>
```

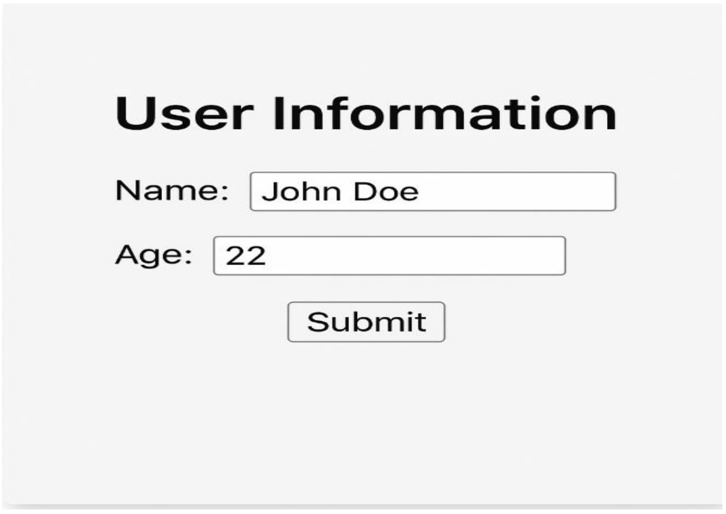
How to Run the Application

1. **Compile the Servlet:**
    - Ensure you have the servlet API in your class path.
    - Compile UserInfoServlet.java.
2. **Deploy the Servlet:**
    - Place the compiled .class file in the appropriate directory of your servlet container (e.g., webapps/YourApp/WEB-INF/classes for Tomcat).
3. **Start the Servlet Container:**
    - Start your servlet container (e.g., Apache Tomcat).
4. **Access the Form:**
    - Open a web browser and navigate to http://localhost:8080/YourApp/index.html.
5. **Submit the Form:**
    - Enter your name and age, then submit the form.
    - The servlet will process the data and display your information.

Explanation

- **HTML Form:** Collects user input and sends it to the servlet via the POST method.
- **Servlet:** Retrieves the parameters using request.getParameter("parameterName") and generates an HTML response displaying the user's information.
- **web.xml:** Configures the servlet and maps it to a URL pattern.

Output:


User Information

Name: John Doe

Age: 22

Demonstration:

Fig.1.3: Glimps of Demonstration.

Conclusion

Efficiently reading and processing servlet parameters is fundamental to building dynamic and secure web applications in Java. By understanding and utilizing the various methods provided by the HttpServletRequest interface, developers can handle client input effectively and create responsive user experiences.