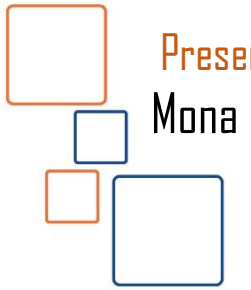
Building Dynamic Web Applications Using Servlets



Presented By
Mona Mahrous,MSc.

Invest In Yourself ,
Develop Your Career

Ch4 : ServletResponse Interface

- The ServletResponse interface defines an object that carries the server's response to the user (i.e. back to the client's browser).

- Commonly used methods:
  - `PrintWriter getWriter()`
  - `setContentType(String mimeType)`
  - `setContentLength(int len)`

Ch4 : ServletResponse Interface

- Sending response in Excel Sheet Format:

```
public void service(ServletRequest request,
                                ServletResponse response)
                                  throws ServletException, IOException
   {
       response.setContentType("application/vnd.ms-excel");
       PrintWriter out = response.getWriter();
       out.println("\t jan \t feb \t march \t total");
       out.println("salary \t100 \t200 \t300 \t=sum(B2:D2)");
  }
```

Ch4 : ServletRequest Interface

- The **`ServletRequest`** defines an object that is used to hold information coming from the user's request, including parameter name-value pairs, attributes, and an input stream.

- Commonly used methods:
  - **`Enumeration getParameterNames()`**
  - **`String getParameter(String paramName)`**
  - **`String[] getParameterValues(String paramName)`**

Ch4 : Sending Request Parameters to Servlet

- In order for the servlet to get the parameters from the request, the following code is written in the **`service(..)`** method:

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

    PrintWriter out = response.getWriter();
    String str = request.getParameter("MyName");
    out.println("The user input is: " + str);
  }
```

Ch4 : GenericServlet Wrapper Class

- The **`GenericServlet`** class is a convenient class that implements the **`Servlet`** interface.

- It overrides the required 5 methods, and hence makes your code simpler (i.e. you extend the class and only override the necessary methods).

Java™ Education and Technology Services

Chapter 5

Getting Familiar with HTTP Package

Ch5 : javax.servlet.http.*

- HTTPServlet ← GenericServlet
- HTTPServletRequest ← ServletRequest
- HTTPServletResponse ← ServletResponse

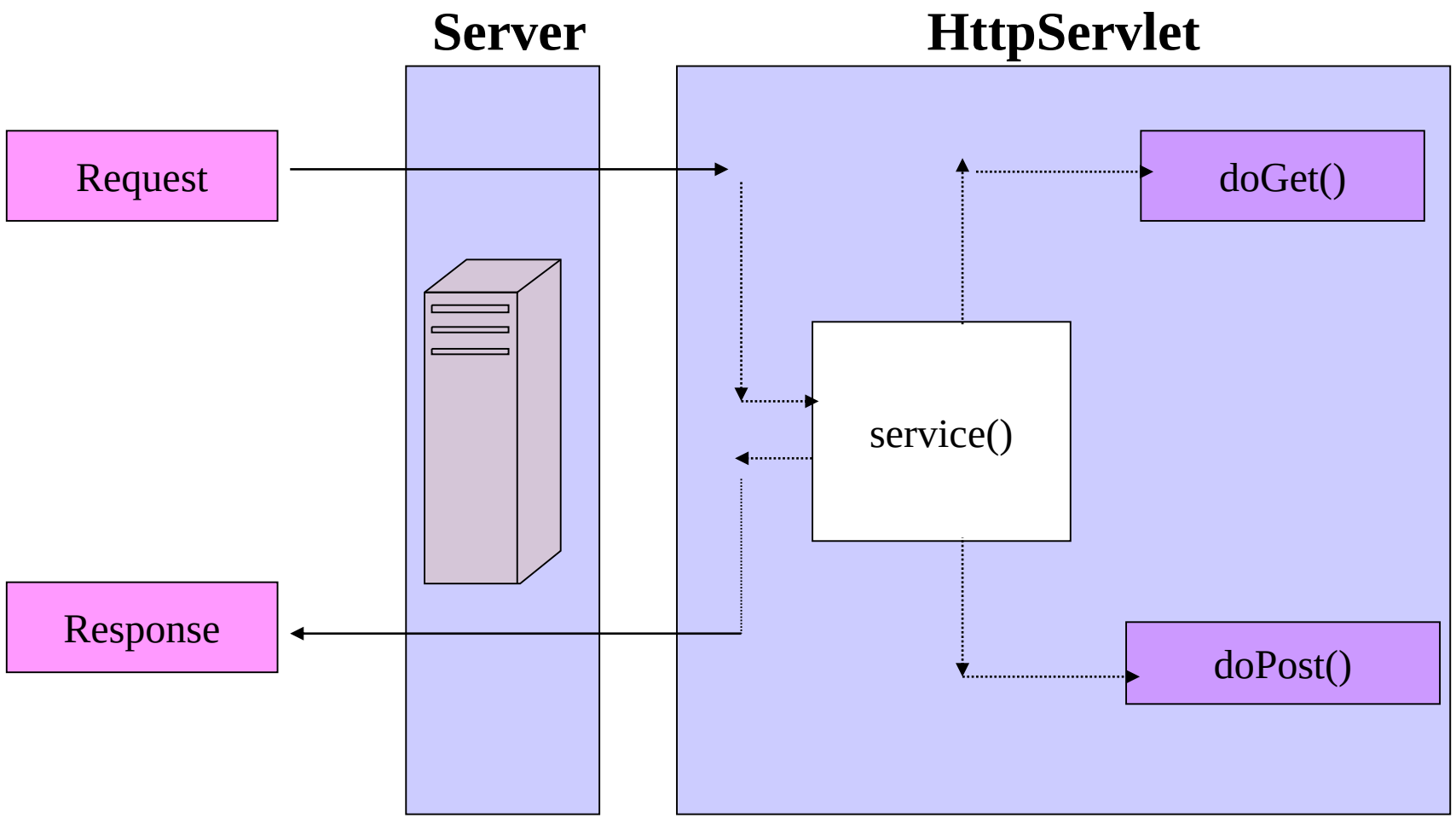
Ch5 : javax.servlet.http.*

- Some basic interfaces
  - HttpServletRequest
  - HttpServletResponse
  - HttpSession
- Classes
  - Cookie
  - HttpServlet

Ch5 : HttpServlet Class

- It extends from javax.servlet.GenericServlet
- It contains extra methods like :
  - doPost() doPut()
  - doGet () doDelete()
  - doOptions() doTrace()

Ch5 : HttpServlet Class



Ch5 : HttpServlet Example

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
                                throws ServletException, IOException
  {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      out.println("<HTML>");     out.println("<BODY>");
      out.println("The servlet has received a GET request." +
      "Now, click on the button below.");
      out.println("<BR>");
      out.println("<FORM METHOD=POST>");
      out.println("<INPUT TYPE=SUBMIT VALUE=Submit>");
      out.println("</FORM>");
      out.println("</BODY>");    out.println("</HTML>");
  }
```

Ch5 : HttpServlet Example

```
public void doPost(HttpServletRequest request,
                           HttpServletResponse response)
                                throws ServletException, IOException
   {
       response.setContentType("text/html");
       PrintWriter out = response.getWriter();
       out.println("<HTML>");     out.println("<HEAD>");
       out.println("<TITLE>The POST method</TITLE>");
       out.println("</HEAD>");
       out.println("<BODY>");
       out.println("The servlet has received a POST request.
                                          Thank you.");
       out.println("</BODY>");
       out.println("</HTML>");
   }
```

Ch5 : HttpServletRequest Interface

- Common methods:
  - `Enumeration getHeaderNames()`
  - `String getHeader(String headerName)`
  - `Enumeration getHeaders(String headerName)`
  - `String getQueryString()`
  - `HttpSession getSession(boolean create)`
  - `Cookie[] getCookies()`

- Remember, methods inherited from **`ServletRequest`**:
  - `Enumeration getParameterNames()`
  - `String getParameter(String paramName)`
  - `String[] getParameterValues(String paramName)`

Ch5 : HttpServletResponse Interface

- Common methods:
  - `setStatus(int statusCode)`
  - `sendRedirect(String newLocation)`
  - `sendError(int statusCode, String msg)`
  - `addCookie(Cookie cookie)`
  - `setHeader(String headerName, String value)`
  - `setIntHeader(String headerName, int value)`
  - `setDateHeader(String headerName, long milliseconds)`

- Remember, methods inherited from **`ServletResponse`**:
  - `PrintWriter getWriter()`
  - `setContentType(String mimeType)`
  - `setContentLength(int len)`

Ch5 : The sendRedirect Method

- The **`sendRedirect(..)`** method is used to redirect the browser's client to go to another page other than the one originally requested.

- This is useful in several situations, for example:
  - If the requested page has been migrated to another server or url.
  - Deciding which page the user will be transferred to after successful login.
  - Checking if the browser enables the use of cookies.

- The location parameter can either be the relative URL of a page or its fully qualified URL.

Ch5 : The sendRedirect Method

- The **`sendRedirect(..)`** method makes a round trip to the client and then back to the server.
- A brand new request object is created, and the previous one is lost.
- Any data written to the response before calling it will be lost.
- If the calling servlet wishes to maintain some information from the old request and propagate it to the new request, then you can pass the information as a query string appended to the destination URL:

```
response.sendRedirect("MySecondServlet?userName="
                      + userName + "&password=" + password);
```

Ch5 : The sendRedirect Method

- **Login Page: login.html**

```
<FORM method= POST action="logServlet">
    <BR> Username: <INPUT TYPE=TEXT NAME=userName>
    <BR> Password: <INPUT TYPE=PASSWORD
      NAME=password>
    <BR> <INPUT TYPE=SUBMIT VALUE=Submit>
 </FORM>
```

Ch5 : The sendRedirect Method

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
                                    throws ServletException, IOException
{
    .
    String userName = request.getParameter("userName");
    String password = request.getParameter("password");
    If // check against Data Base
    {
      response.sendRedirect("WelcomeServletURL");
    }
    else {
    out.println("Login Failed, please try again.");
    out.println("<a href='login.html'> Login Page</a>");
    }
}
```

Java™ Education and Technology Services

Lab Exercise

Lab Exercise

- **Login :**
  - Write a Login html page that has two text input fields for the user's name and mail, along with a submit button.
  - Upon pressing the submit button, the page should send the data to a servlet (doPost(..)).
  - Upon submitting the login form, in case of successful login the user should be redirected to a welcome servlet which prints a welcome message with the user name.
  - If the login data is incorrect, redirect the user to the login page.

Ch5 : The RequestDispatcher Interface

- The **`RequestDispatcher`** interface has 2 methods only:
  - `include(ServletRequest request, ServletResponse response)`
  - `forward(ServletRequest request, ServletResponse response)`

Ch5 : The RequestDispatcher Interface - include

- The `include(..)` method is used to include the output of another servlet or the content of another page (html, JSP, …etc) to the current servlet.
- This technique is famously used in websites to include a certain header or footer to all the pages of the website.
- It may also be used to include some common JavaScript functions to many web pages.
- The original servlet can write to the response before and after the included servlet or page writes to the response.

Ch5 : The RequestDispatcher Interface - forward

- The `forward(..)` method passes the processing of an HTTP request from the current servlet to another servlet or JSP (i.e. gives up the control for another servlet or jsp).
- It is up to the new servlet or JSP to write to the response object and then commit it to the client.
- Any output written to the response object by the original servlet will be lost (i.e. don't write to the response before forwarding to another servlet).
- The original servlet can perform some initial tasks on the request object before forwarding it (e.g. put additional attributes on it, which will be used by the new servlet or jsp).

Ch5 : The RequestDispatcher Interface

- There are two ways to get a dispatcher for the desired destination servlet or page. Either from the request, or from the context.

- From the **ServletRequest**:
  - **getRequestDispatcher(String path)**

```
//relative path or fully qualified path
```

- From the **ServletContext**:
  - **getRequestDispatcher(String fullPath)**

```
// fully qualified path within the context, beginning with "/"
```

  - **getNamedDispatcher(String servletName)**

```
// the servlet name in the web.xml file
```

Ch5 : The RequestDispatcher Interface

- The following code demonstrates the use of the **RequestDispather**:
  - ```
    RequestDispatcher rd =
                request.getRequestDispatcher("SecondServletURL");
    ```
  - ```
    rd.include(request, response);
    ```

  OR:
  - ```
    rd.forward(request, response);
    ```

- Note: If you use **include** or **forward** from a **doPost** method, then the **doPost** method of the second servlet will be invoked.
- If you use them from a **doGet** method, then the **doGet** method of the second servlet will be invoked.
- Therefore, it is advisable to implement both **doGet** and **doPost** in the destination servlet (write the code in one of the two methods, and call it from the other method).
    ```

Ch5 : The RequestDispatcher Interface - Example

- **Include headers and footers to your page**

```
PrintWriter out = response.getWriter();
// add Header
out.println(“This the Header");
// real processing
request.setAttribute("username",“ITI");
RequestDispatcher rd = request.getRequestDispatcher("DispatcherServlet2");
rd.include(request, response);
// back to add footer
out.println(“This the Footer");
```

Ch5 : SendRedirect VS Forward

- **`sendRedirect(..)`**:
  - goes through a round trip to the client side then back to the server.
  - The client's original request will be lost.
  - To pass information, append a query string to the new location's URL.

- **`forward(..)`**:
  - Redirects the request internally inside the server.
  - Both the request and the response are passed to the new resource.

Java™ Education and Technology Services

Lab Exercise

Lab Exercise

- **Login :**
  - Write a Login html page that has two text input fields for the user's name and mail, along with a submit button.
  - Upon pressing the submit button, the page should send the data to a servlet (doPost(..)).
  - Upon submitting the login form, the user should be forwarded to a welcome page(in case of successful login.. Try to send user name to it).
  - If the login data is incorrect, display a colored error message before included the login page.

Ch5 : @WebServlet annoation

- **Starting from JEE 6**
- **Package javax.servlet.annotation**

```
@WebServlet(
  attribute1=value1,
  attribute2=value2,
  ...
)
public class TheServlet extends
javax.servlet.http.HttpServlet {
  // servlet code...
}
```

Ch5 : @WebServlet annoation

```
@WebServlet("/processForm")
public class TheServlet extends
  javax.servlet.http.HttpServlet {
  // servlet code...
}
```

```
@WebServlet(urlPatterns = {"/sendFile", "/uploadFile"})
public class TheServlet extends
  javax.servlet.http.HttpServlet {
  // servlet code...
}
```