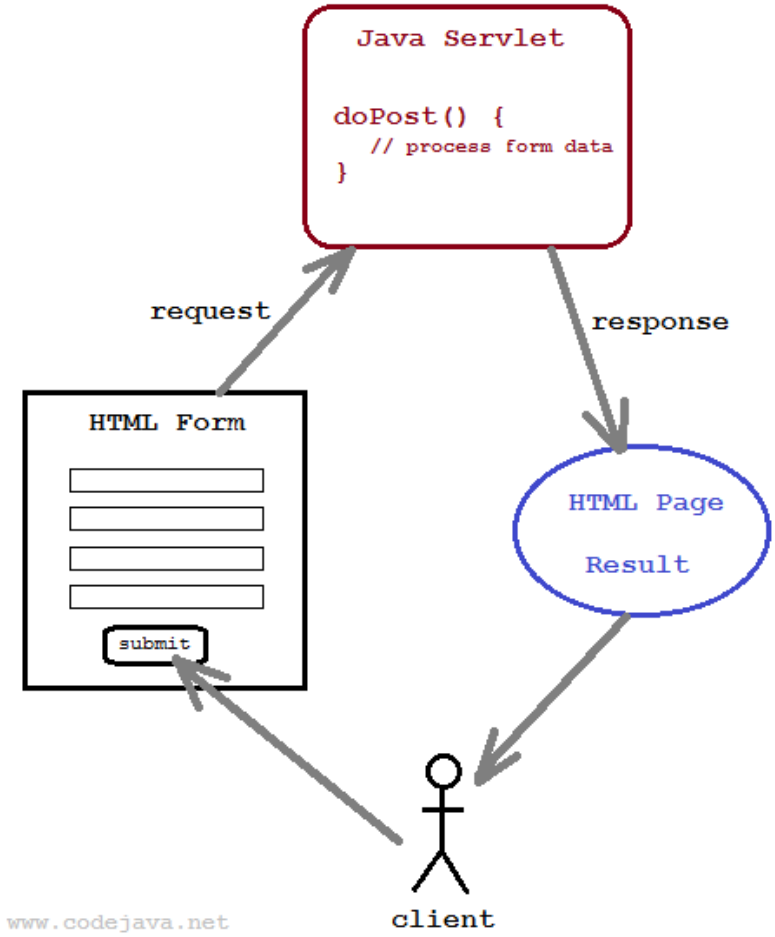
In this Java servlet tutorial, I will guide you how to read values of common input fields from HTML form on the server side with Java Servlet.
You know, handling form data represented in HTML page is a very common task in web development. A typical scenario is the user fills in fields of a form and submits it. The server will process the request based on the submitted data, and send response back to the client. The following picture depicts that workflow with Java servlet on the server side:



To create a form in HTML we need to use the following tags:

-
  - `<form>`: to create a form to add fields in its body.
  - `<input>`, `<select>`, `<textarea>`…: to create form fields like text boxes, dropdown list, text area, check boxes, radio buttons,… and submit button.

To make the form works with Java servlet, we need to specify the following attributes for the `<form>` tag:

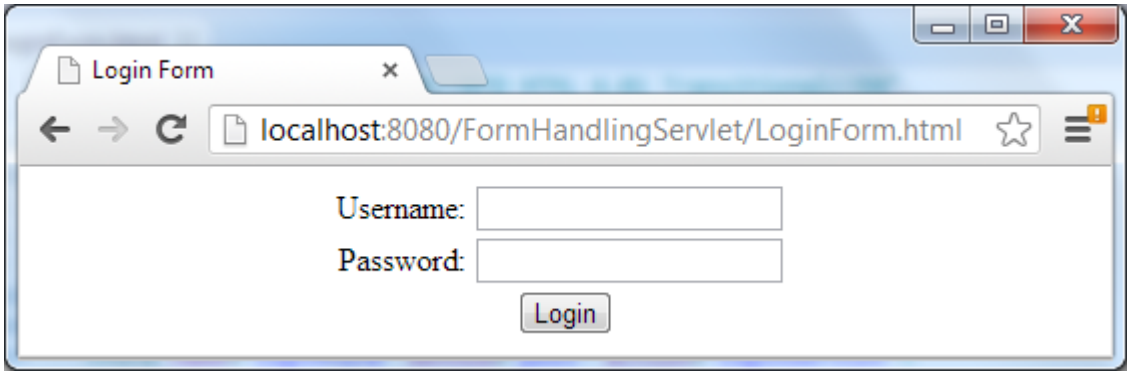
-
  - **method**=”post”: to send the form data as an HTTP POST request to the server. Generally, form submission should be done in HTTP POST method.
  - **action**=”*URL of the servlet*”: specifies relative URL of the servlet which is responsible for handling data posted from this form.

For example, following is HTML code of a login form:

```
1<form name="loginForm" method="post" action="loginServlet">
```

```
    Username: <input type="text" name="username"/> <br/>
    Password: <input type="password" name="password"/> <br/>
    <input type="submit" value="Login" />
</form>

```

This form would look like this in browser:



On the server side, we need to create a Java servlet which is mapped to the URL: *loginServlet*, as specified in the form's action attribute. Following is code of the servlet:

```
@WebServlet("/loginServlet")
public class LoginServlet extends HttpServlet {

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

        // code to process the form...

    }

}
```

Notice that the servlet's URL is specified by the `@WebServlet` annotation before the servlet class. When the user submits the login form above, the servlet's `doPost()` method will be invoked by the servlet container. Typically we will do the following tasks inside `doPost()` method:

-
  - Read values of the fields posted from the form via the `request` object (implementation of `javax.servlet.http.HttpServletRequest` interface).
  - Do some processing, e.g. connecting to database to validate the username and password.
  - Return response back to the user via the `respone` object (implementation of `javax.servlet.http.HttpServletResponse` interface).

To read values of form's fields, the `HttpServletRequest` interface provides the following methods:

-

- **`String getParameter(String name)`**: gets value of a field which is specified by the given name, as a String. The method returns null if there is no form field exists with the given name.
- **`String[] getParameterValues(String name)`**: gets values of a group of fields which have same name, in an array of String objects. The method returns null if there is no field exists with the given name.

Note that the above methods can also deal with parameters in URL's query string, hence the name getParameter.

For example, we can write the following code in the `doPost()` method to read values of form's fields:

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

To send response back to the client, we need to obtain a writer from the response object by calling the method `getWriter()` of the `HttpServletResponse` interface:

```
PrintWriter writer = response.getWriter();
```

Then use the `print()` or `println()` method to deliver the response (in form of HTML code). For example:

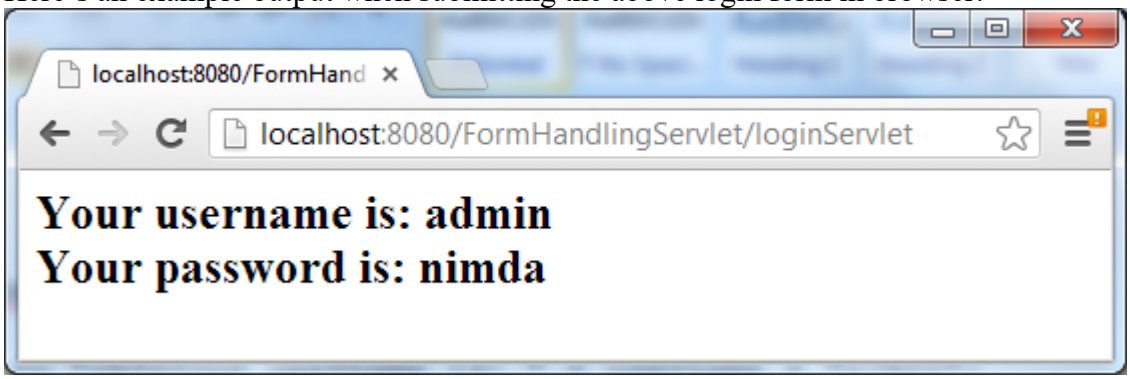
```
String htmlRespone = "<html>";
htmlRespone += "<h2>Your username is: " + username + "</h2>";
htmlRespone += "</html>";

writer.println(htmlRespone);
```

Here's complete code of the servlet class to process the login form:

```
package net.codejava.servlet;

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("/loginServlet")
public class LoginServlet extends HttpServlet {

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

        // read form fields
        String username = request.getParameter("username");
        String password = request.getParameter("password");

        System.out.println("username: " + username);
        System.out.println("password: " + password);

        // do some processing here...

        // get response writer
        PrintWriter writer = response.getWriter();

```

```
        // build HTML code
        String htmlRespone = "<html>";
        htmlRespone += "<h2>Your username is: " + username +
"<br/>";
        htmlRespone += "Your password is: " + password + "</h2>";
        htmlRespone += "</html>";

        // return response
        writer.println(htmlRespone);

    }
}
```

Here's an example output when submitting the above login form in browser:



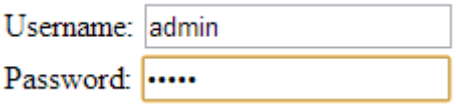
So far you have got the ins and outs when handling HTML form data with Java servlet. For your reference, we provide a list of examples for handling common HTML form fields as below. Note that we use the `System.out.println()` statement in servlet to demo the output.

1. Read values of text field and password field

- HTML code:

```
Username: <input type="text" name="username"/>
Password: <input type="password" name="password"/>
```

- Field image:



- Java code in servlet:

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

System.out.println("username is: " + username);
System.out.println("password is: " + password);
```

- Output:

```
username is: admin
password is: nimda
```

2. Read value of checkbox field

- HTML code:

```
Speaking language:
<input type="checkbox" name="language" value="english" />English
<input type="checkbox" name="language" value="french" />French
```

Speaking language: ☑ English ☑ French

- Field image:
- Java code in servlet:

```
String languages[] = request.getParameterValues("language");

if (languages != null) {
    System.out.println("Languages are: ");
    for (String lang : languages) {
        System.out.println("\t" + lang);
    }
}
```

- Output:

```
Languages are:
    english
    french
```

3. Read value of radio button field

- HTML code:

```
Gender:
<input type="radio" name="gender" value="male" />Male
<input type="radio" name="gender" value="female" />Female
```

Gender: ◉ Male ○ Female

- Field image:
- Java code in servlet:

```
String gender = request.getParameter("gender");
System.out.println("Gender is: " + gender);
```

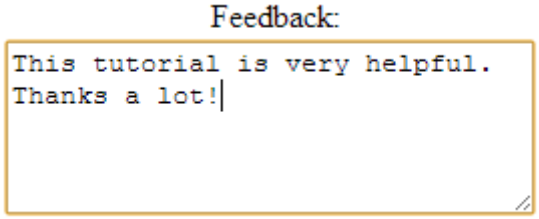
- Output:

```
Gender is: male
```

4. Read value of text area field

- HTML code:

```
Feedback:<br/>
<textarea rows="5" cols="30" name="feedback"></textarea>
```



- Field image:
- Java code in servlet:

```
String feedback = request.getParameter("feedback");
System.out.println("Feed back is: " + feedback);
```

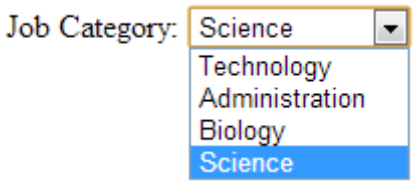
- Output:

```
Feed back is: This tutorial is very helpful. Thanks a lot!
```

5. Read value of dropdown list (combobox) field

- HTML code:

```

Job Category:
<select name="jobCat">
    <option value="tech">Technology</option>
    <option value="admin">Administration</option>
    <option value="biology">Biology</option>
    <option value="science">Science</option>
</select>

```



- Field image:
- Java code in servlet:

```
String jobCategory = request.getParameter("jobCat");
System.out.println("Job category is: " + jobCategory);
```

- Output:

```
Job category is: science
```

RequestDispatcher in Servlet

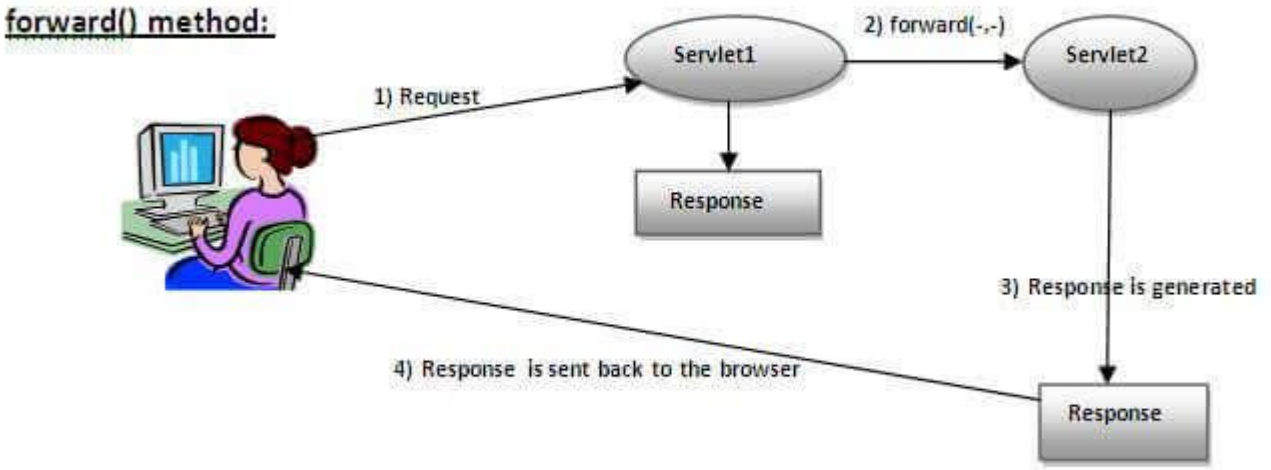
The RequestDispatcher interface provides the facility of dispatching the request to another resource it may be html, servlet or jsp. This interface can also be used to include the content of another resource also. It is one of the way of servlet collaboration.

There are two methods defined in the RequestDispatcher interface.

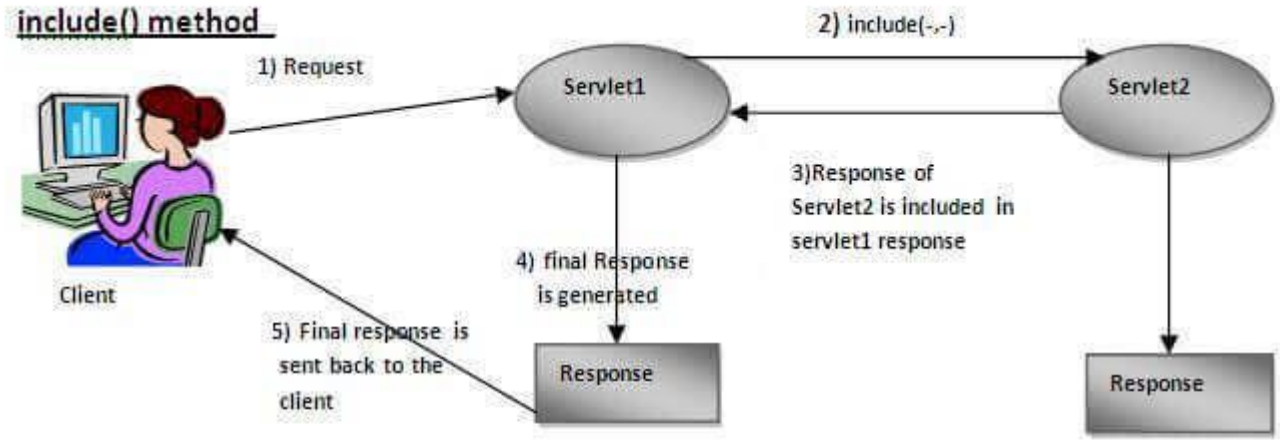
Methods of RequestDispatcher interface

The RequestDispatcher interface provides two methods. They are:

1. **public void forward(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException:**Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.
2. **public void include(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException:**Includes the content of a resource (servlet, JSP page, or HTML file) in the response.



As you see in the above figure, response of second servlet is sent to the client. Response of the first servlet is not displayed to the user.



As you can see in the above figure, response of second servlet is included in the response of the first servlet that is being sent to the client.

How to get the object of RequestDispatcher

The getRequestDispatcher() method of ServletRequest interface returns the object of RequestDispatcher. Syntax:

Syntax of getRequestDispatcher method

```
1. public RequestDispatcher getRequestDispatcher(String resource);
```

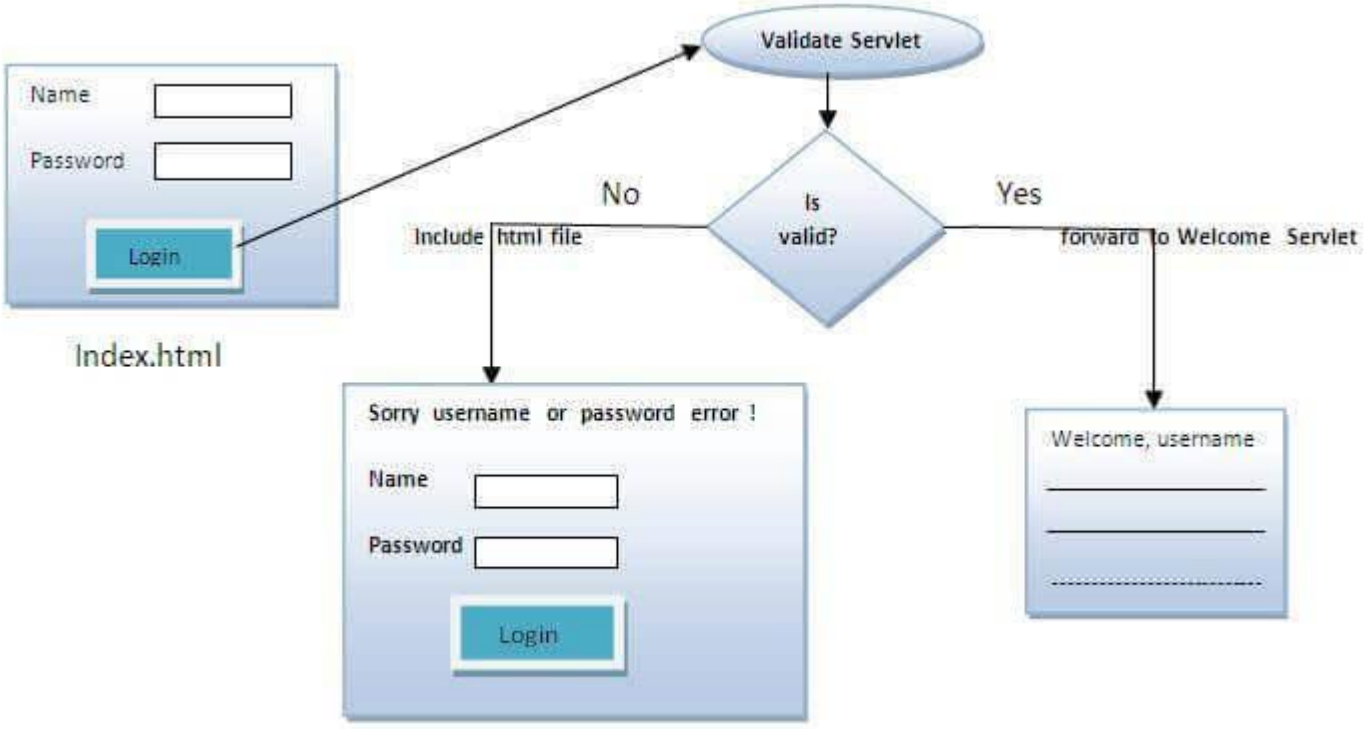
Example of using getRequestDispatcher method

```
1. RequestDispatcher rd=request.getRequestDispatcher("servlet2");
2. //servlet2 is the url-pattern of the second servlet
3.
4. rd.forward(request, response);//method may be include or forward
```

Example of RequestDispatcher interface

In this example, we are validating the password entered by the user. If password is servlet, it will forward the request to the WelcomeServlet, otherwise will show an error message: sorry username or password error!. In this program, we are cheking for hardcoded information. But you can check it to the database also that we will see in the development chapter. In this example, we have created following files:

- **index.html file:** for getting input from the user.
- **Login.java file:** a servlet class for processing the response. If password is servet, it will forward the request to the welcome servlet.
- **WelcomeServlet.java file:** a servlet class for displaying the welcome message.
- **web.xml file:** a deployment descriptor file that contains the information about the servlet.



---

index.html

```
1. <form action="servlet1" method="post">
2. Name:<input type="text" name="userName"/><br/>
3. Password:<input type="password" name="userPass"/><br/>
4. <input type="submit" value="login"/>
5. </form>
```

Login.java

```
1. import java.io.*;
2. import javax.servlet.*;
3. import javax.servlet.http.*;
4.
5.
6. public class Login extends HttpServlet {
7.
8. public void doPost(HttpServletRequest request, HttpServletResponse response)
9.         throws ServletException, IOException {
10.
11.    response.setContentType("text/html");
12.    PrintWriter out = response.getWriter();
13.
14.    String n=request.getParameter("userName");
15.    String p=request.getParameter("userPass");
```

```
16.
17.    if(p.equals("servlet"){
18.       RequestDispatcher rd=request.getRequestDispatcher("servlet2");
19.       rd.forward(request, response);
20.    }
21.    else{
22.       out.print("Sorry UserName or Password Error!");
23.       RequestDispatcher rd=request.getRequestDispatcher("/index.html");
24.       rd.include(request, response);
25.
26.       }
27.    }
28.
29. }
```

WelcomeServlet.java

```
1.  import java.io.*;
2.  import javax.servlet.*;
3.  import javax.servlet.http.*;
4.
5.  public class WelcomeServlet extends HttpServlet {
6.
7.     public void doPost(HttpServletRequest request, HttpServletResponse response)
8.        throws ServletException, IOException {
9.
10.    response.setContentType("text/html");
11.    PrintWriter out = response.getWriter();
12.
13.    String n=request.getParameter("userName");
14.    out.print("Welcome "+n);
15.    }
16.
17. }
```

---

web.xml

```
1.  <web-app>
2.   <servlet>
3.     <servlet-name>Login</servlet-name>
4.     <servlet-class>Login</servlet-class>
5.    </servlet>
6.    <servlet>
7.     <servlet-name>WelcomeServlet</servlet-name>
8.     <servlet-class>WelcomeServlet</servlet-class>
9.    </servlet>
10.
11.
12.   <servlet-mapping>
```

```
    <servlet-name>Login</servlet-name>
    <url-pattern>/servlet1</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>WelcomeServlet</servlet-name>
    <url-pattern>/servlet2</url-pattern>
  </servlet-mapping>

  <welcome-file-list>
   <welcome-file>index.html</welcome-file>
  </welcome-file-list>
</web-app>
```