REFERENCES: <https://www.tutorialspoint.com/servlets/servlets-form-data.htm>

http://www.oracle.com/technetwork/java/servlet-142430.html

**HMTL SERVLET**

A servlet is an extension to a server that enhances the server's functionality. The most common use for a servlet is to extend a web server by providing dynamic web content. Web servers display documents written in HyperText Markup Language (HTML) and respond to user requests using the HyperText Transfer Protocol (HTTP). HTTP is the protocol for moving hypertext files across the internet. HTML documents contain text that has been marked up for interpretation by an HTML browser such as Netscape.

Servlets are easy to write. All you need is Tomcat, which is the combined Java Server Pages 1.1 and Servlets 2.2 reference implementation. You can download a free copy of [Tomcat](http://www.oracle.com/technetwork/java/index-jsp-138231.html).

**About the Example**

A browser accepts end user input through an HTML form. The simple form used in this lesson has one text input field for the end user to enter text and a Submit button. When the end user clicks the Submit button, the simple servlet is invoked to process the end user input.

In this example, the simple servlet returns an HTML page that displays the text entered by the end user.

**HTML Form**

The HTML form is embedded in this [HTML file](http://www.oracle.com/technetwork/java/simplehtml-137341.html). The diagram shows how the HTML page looks when it is opened in a browser.

The HTML file and form are similar to the simple application and applet examples in [Lesson 4](http://www.oracle.com/technetwork/java/front-139339.html) so you can compare the code and learn how servlets, applets, and applications handle end user inputs.

When the user clicks the Click Me button, the servlet gets the entered text, and returns an HTML page with the text.

The HTML page returned to the browser by the [ExampServlet.java](http://www.oracle.com/technetwork/java/servlet-142430.html) servlet is shown below. The servlet code to retrieve the user's input and generate the HTML page follows with a discussion.

|  |
| --- |
|  |

**Servlet Backend**

[ExampServlet.java](http://www.oracle.com/technetwork/java/servlet-142430.html) builds an HTML page to return to the end user. This means the servlet code does not use any Project Swing or Abstract Window Toolkit (AWT) components or have event handling code. For this simple servlet, you only need to import these packages:

* java.io for system input and output. The HttpServlet class uses the IOException class in this package to signal that an input or output exception of some kind has occurred.

* javax.servlet, which contains generic (protocol-independent) servlet classes. TheHttpServlet class uses the ServletException class in this package to indicate a servlet problem.

* javax.servlet.http, which contains HTTP servlet classes. The HttpServlet class is in this package.

|  |
| --- |
| import java.io.\*;  import javax.servlet.\*;  import javax.servlet.http.\*;  public class ExampServlet extends HttpServlet {  public void doPost(HttpServletRequest request,  HttpServletResponse response)  throws ServletException, IOException  {  response.setContentType("text/html");  PrintWriter out = response.getWriter();  out.println("<title>Example</title>" +  "<body bgcolor=FFFFFF>");  out.println("<h2>Button Clicked</h2>");  String DATA = request.getParameter("DATA");  if(DATA != null){  out.println(DATA);  } else {  out.println("No text entered.");  }  out.println("<P>Return to  <A HREF="../simpleHTML.html">Form</A>");  out.close();  }  } |

**Class and Method Declarations**

All servlet classes extend the HttpServlet abstract class. HttpServlet simplifies writing HTTP servlets by providing a framework for handling the HTTP protocol. Because HttpServlet isabstract, your servlet class must extend it and override at least one of its methods. An abstractclass is a class that contains unimplemented methods and cannot be instantiated itself.

|  |
| --- |
| public class ExampServlet extends HttpServlet {  public void doPost(HttpServletRequest request,  HttpServletResponse response)  throws ServletException, IOException  { |

The ExampServlet class is declared public so the web server that runs the servlet, which is not local to the servlet, can access it.

The ExampServlet class defines a doPost method with the same name, return type, and parameter list as the doPost method in the HttpServlet class. By doing this, the ExampServletclass overrides and implements the doPost method in the HttpServlet class.

The doPost method performs the HTTP POST operation, which is the type of operation specified in the HTML form used for this example. The other possibility is the HTTP GET operation, in which case you would implement the doGet method instead.

In short, POST requests are for sending any amount of data directly over the connection without changing the URL, and GET requests are for getting limited amounts of information appended to the URL. POST requests cannot be bookmarked or emailed and do not change the Uniform Resource Locators (URL) of the response. GET requests can be bookmarked and emailed and add information to the URL of the response.

The parameter list for the doPost method takes a request and a response object. The browser sends a request to the servlet and the servlet sends a response back to the browser.

The doPost method implementation accesses information in the request object to find out who made the request, what form the request data is in, and which HTTP headers were sent, and uses the response object to create an HTML page in response to the browser's request.

doPostIOExceptionServletExceptionHttpServlet

**Method Implementation**

The first part of the doPost method uses the response object to create an HTML page. It first sets the response content type to be text/html, then gets a PrintWriter object for formatted text output.

|  |
| --- |
| response.setContentType("text/html");  PrintWriter out = response.getWriter();  out.println("<title>Example</title>" +  "<body bgcolor=#FFFFFF>");  out.println("<h2>Button Clicked</h2>"); |

The next line uses the request object to get the data from the text field on the form and store it in the DATA variable. The getparameter method gets the named parameter, returns null if the parameter was not set, and an empty string if the parameter was sent without a value.

String DATA = request.getParameter("DATA");

The next part of the doPost method gets the data out of the DATA parameter and passes it to theresponse object to add to the HTML response page.

|  |
| --- |
| if(DATA != null){  out.println(DATA);  } else {  out.println("No text entered.");  } |

The last part of the doPost method creates a link to take the end user from the HTML response page back to the original form, and closes the response.

out.println("<P>Return to

<A HREF="../simpleHTML.html">Form</A>");

out.close();

}

Servlets - Form Data

## **GET Method**

The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the **?**(question mark) symbol as follows −

The GET method is the default method to pass information from browser to web server and it produces a long string that appears in your browser's Location:box. Never use the GET method if you have password or other sensitive information to pass to the server. The GET method has size limitation: only 1024 characters can be used in a request string.

This information is passed using QUERY\_STRING header and will be accessible through QUERY\_STRING environment variable and Servlet handles this type of requests using **doGet()** method.

## **POST Method**

A generally more reliable method of passing information to a backend program is the POST method. This packages the information in exactly the same way as GET method, but instead of sending it as a text string after a ? (question mark) in the URL it sends it as a separate message. This message comes to the backend program in the form of the standard input which you can parse and use for your processing. Servlet handles this type of requests using **doPost()** method.

## **Reading Form Data using Servlet**

Servlets handles form data parsing automatically using the following methods depending on the situation −

* **getParameter()** − You call request.getParameter() method to get the value of a form parameter.
* **getParameterValues()** − Call this method if the parameter appears more than once and returns multiple values, for example checkbox.
* **getParameterNames()** − Call this method if you want a complete list of all parameters in the current request.

## **GET Method Example using URL**

Here is a simple URL which will pass two values to HelloForm program using GET method.

**http://localhost:8080/HelloForm?first\_name = ZARA&last\_name = ALI**

Given below is the **HelloForm.java** servlet program to handle input given by web browser. We are going to use **getParameter()** method which makes it very easy to access passed information −

CODE:

// Import required java libraries

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

// Extend HttpServlet class

public class HelloForm extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

// Set response content type

response.setContentType("text/html");

PrintWriter out = response.getWriter();

String title = "Using GET Method to Read Form Data";

String docType =

"<!doctype html public \"-//w3c//dtd html 4.0 " + "transitional//en\">\n";

out.println(docType +

"<html>\n" +

"<head><title>" + title + "</title></head>\n" +

"<body bgcolor = \"#f0f0f0\">\n" +

"<h1 align = \"center\">" + title + "</h1>\n" +

"<ul>\n" +

" <li><b>First Name</b>: "

+ request.getParameter("first\_name") + "\n" +

" <li><b>Last Name</b>: "

+ request.getParameter("last\_name") + "\n" +

"</ul>\n" +

"</body>

</html>"

);

}

}

Assuming your environment is set up properly, compile HelloForm.java as follows −

$ javac HelloForm.java

If everything goes fine, above compilation would produce **HelloForm.class**file. Next you would have to copy this class file in <Tomcat-installationdirectory>/webapps/ROOT/WEB-INF/classes and create following entries in **web.xml** file located in <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/

<servlet>

<servlet-name>HelloForm</servlet-name>

<servlet-class>HelloForm</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>HelloForm</servlet-name>

<url-pattern>/HelloForm</url-pattern>

</servlet-mapping>

Now type *http://localhost:8080/HelloForm?first\_name=ZARA&last\_name=ALI*in your browser's Location:box and make sure you already started tomcat server, before firing above command in the browser. This would generate following result −

# Using GET Method to Read Form Data

* **First Name**: ZARA
* **Last Name**: ALI

## **GET Method Example Using Form**

Here is a simple example which passes two values using HTML FORM and submit button. We are going to use same Servlet HelloForm to handle this imput.

<html>

<body>

<form action = "HelloForm" method = "GET">

First Name: <input type = "text" name = "first\_name">

<br />

Last Name: <input type = "text" name = "last\_name" />

<input type = "submit" value = "Submit" />

</form>

</body>

</html>

Keep this HTML in a file Hello.htm and put it in <Tomcat-installationdirectory>/webapps/ROOT directory. When you would access *http://localhost:8080/Hello.htm*, here is the actual output of the above form.

Top of Form

First Name:  Last Name: 

Bottom of Form

Try to enter First Name and Last Name and then click submit button to see the result on your local machine where tomcat is running. Based on the input provided, it will generate similar result as mentioned in the above example.

## **POST Method Example Using Form**

Let us do little modification in the above servlet, so that it can handle GET as well as POST methods. Below is **HelloForm.java** servlet program to handle input given by web browser using GET or POST methods.

// Import required java libraries

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

// Extend HttpServlet class

public class HelloForm extends HttpServlet {

// Method to handle GET method request.

public void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

// Set response content type

response.setContentType("text/html");

PrintWriter out = response.getWriter();

String title = "Using GET Method to Read Form Data";

String docType =

"<!doctype html public \"-//w3c//dtd html 4.0 " +

"transitional//en\">\n";

out.println(docType +

"<html>\n" +

"<head><title>" + title + "</title></head>\n" +

"<body bgcolor = \"#f0f0f0\">\n" +

"<h1 align = \"center\">" + title + "</h1>\n" +

"<ul>\n" +

" <li><b>First Name</b>: "

+ request.getParameter("first\_name") + "\n" +

" <li><b>Last Name</b>: "

+ request.getParameter("last\_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);

}

}

Now compile and deploy the above Servlet and test it using Hello.htm with the POST method as follows −

<html>

<body>

<form action = "HelloForm" method = "POST">

First Name: <input type = "text" name = "first\_name">

<br />

Last Name: <input type = "text" name = "last\_name" />

<input type = "submit" value = "Submit" />

</form>

</body>

</html>

Here is the actual output of the above form, Try to enter First and Last Name and then click submit button to see the result on your local machine where tomcat is running.

Top of Form

First Name:  Last Name: 

Bottom of Form

Based on the input provided, it would generate similar result as mentioned in the above examples.

## **Passing Checkbox Data to Servlet Program**

Checkboxes are used when more than one option is required to be selected.

Here is example HTML code, CheckBox.htm, for a form with two checkboxes

<html>

<body>

<form action = "CheckBox" method = "POST" target = "\_blank">

<input type = "checkbox" name = "maths" checked = "checked" /> Maths

<input type = "checkbox" name = "physics" /> Physics

<input type = "checkbox" name = "chemistry" checked = "checked" />

Chemistry

<input type = "submit" value = "Select Subject" />

</form>

</body>

</html>

The result of this code is the following form

Top of Form

 Maths  Physics  Chemistry

Bottom of Form

Given below is the CheckBox.java servlet program to handle input given by web browser for checkbox button.

// Import required java libraries

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

// Extend HttpServlet class

public class CheckBox extends HttpServlet {

// Method to handle GET method request.

public void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

// Set response content type

response.setContentType("text/html");

PrintWriter out = response.getWriter();

String title = "Reading Checkbox Data";

String docType =

"<!doctype html public \"-//w3c//dtd html 4.0 " + "transitional//en\">\n";

out.println(docType +

"<html>\n" +

"<head><title>" + title + "</title></head>\n" +

"<body bgcolor = \"#f0f0f0\">\n" +

"<h1 align = \"center\">" + title + "</h1>\n" +

"<ul>\n" +

" <li><b>Maths Flag : </b>: "

+ request.getParameter("maths") + "\n" +

" <li><b>Physics Flag: </b>: "

+ request.getParameter("physics") + "\n" +

" <li><b>Chemistry Flag: </b>: "

+ request.getParameter("chemistry") + "\n" +

"</ul>\n" +

"</body>

</html>"

);

}

// Method to handle POST method request.

public void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

doGet(request, response);

}

}

For the above example, it would display following result −

# Reading Checkbox Data

* **Maths Flag :** : on
* **Physics Flag:** : null
* **Chemistry Flag:** : on

## **Reading All Form Parameters**

Following is the generic example which uses **getParameterNames()** method of HttpServletRequest to read all the available form parameters. This method returns an Enumeration that contains the parameter names in an unspecified order

Once we have an Enumeration, we can loop down the Enumeration in standard way by, using *hasMoreElements()* method to determine when to stop and using *nextElement()* method to get each parameter name.

// Import required java libraries

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

import java.util.\*;

// Extend HttpServlet class

public class ReadParams extends HttpServlet {

// Method to handle GET method request.

public void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

// Set response content type

response.setContentType("text/html");

PrintWriter out = response.getWriter();

String title = "Reading All Form Parameters";

String docType =

"<!doctype html public \"-//w3c//dtd html 4.0 " + "transitional//en\">\n";

out.println(docType +

"<html>\n" +

"<head><title>" + title + "</title></head>\n" +

"<body bgcolor = \"#f0f0f0\">\n" +

"<h1 align = \"center\">" + title + "</h1>\n" +

"<table width = \"100%\" border = \"1\" align = \"center\">\n" +

"<tr bgcolor = \"#949494\">\n" +

"<th>Param Name</th>

<th>Param Value(s)</th>\n"+

"</tr>\n"

);

Enumeration paramNames = request.getParameterNames();

while(paramNames.hasMoreElements()) {

String paramName = (String)paramNames.nextElement();

out.print("<tr><td>" + paramName + "</td>\n<td>");

String[] paramValues = request.getParameterValues(paramName);

// Read single valued data

if (paramValues.length == 1) {

String paramValue = paramValues[0];

if (paramValue.length() == 0)

out.println("<i>No Value</i>");

else

out.println(paramValue);

} else {

// Read multiple valued data

out.println("<ul>");

for(int i = 0; i < paramValues.length; i++) {

out.println("<li>" + paramValues[i]);

}

out.println("</ul>");

}

}

out.println("</tr>\n</table>\n</body></html>");

}

// Method to handle POST method request.

public void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

doGet(request, response);

}

}

Now, try the above servlet with the following form −

<html>

<body>

<form action = "ReadParams" method = "POST" target = "\_blank">

<input type = "checkbox" name = "maths" checked = "checked" /> Maths

<input type = "checkbox" name = "physics" /> Physics

<input type = "checkbox" name = "chemistry" checked = "checked" /> Chem

<input type = "submit" value = "Select Subject" />

</form>

</body>

</html>

Now calling servlet using the above form would generate the following result −

# Reading All Form Parameters

|  |  |
| --- | --- |
| **Param Name** | **Param Value(s)** |
| maths | on |
| chemistry | on |

You can try the above servlet to read any other form's data having other objects like text box, radio button or drop down box etc.