Handling client request form data with Servlets

HTML forms are commonly used for collecting data from users and transmitting them to web applications that process data.

HTML forms can contain a variety of user interface controls that are used for collecting input within a Web page. Each control has a name and a value. The control name is specified in HTML document, while value can be collected from user or can be a default value.

The form data are process by a web application (program) that is specified with the URL within the form. When a user submits the form, all names and values of the user controls are sent to the specified URL as a string. The created string can be sent by using two methods:

- **HTTP GET**. The method appends the form data to the end of the specified URL of a program that processes data. Form data are appended after a question mark (?).
- **HTTP POST**. The method sends the data after HTTP request headers and a blink line (without presenting a query string behind URL).

Web Forms Basics

HTML forms allow users to enter data to be processed by Web applications.

HTML forms allow you to create a set of data input elements associated with a particular URL. Each of these elements is typically given a name in the HTML source code, and each has a value based on the original HTML or user input. When the form is submitted, the names and values of all active elements are collected into a string with = between each name and value and with & between each name/value pair. This string is then transmitted to the URL designated by the FORM element. The string is either appended to the URL after a question mark or sent on a separate line after the HTTP request headers and a blank line, depending on whether GET or POST is used as the submission method.. The following sections cover the various user interface controls that can be used within forms

FORM element

This section covers the FORM element itself, used primarily to designate the URL and to choose the submission method. The FORM element envelops data input elements for collecting user input, and create the URL with collected data. HTML FORM element has the following syntax:

`<FORM ACTION="..." ...> ... </FORM>`

An example of a form that sends created URL with form input data to servlet **`GetPostFormsServlet`** is:

```
<form ACTION="GetPostFormsServlet" METHOD="POST">
  First name:
  <input TYPE="TEXT" NAME="firstName" VALUE="Paja"><br>
  Last name:
  <input TYPE="TEXT" NAME="lastName" VALUE="Patak"><p>
  <input TYPE="SUBMIT" VALUE="Submit with HTTP POST">
</form>
```

The most important attributes of the FORM element are:

- **`ACTION`** specifies address of a server side application that will process collected data within the form. If the server application is on different computer an absolute URL should be used (e.g. **`http://www.website.com/servlet/ServletName`**). If the server application is on the same computer, like in testing during development, relative addresses are commonly used. In addition, for data can be sent to a specific email address if **`ACTION`** is in the form **`mailto:name@website.com`**, which assumes using HTTP POST method for separating sent values from URL address.
- **`METHOD`** specifies how collected data will be sent to server side program. If HTTP GET method is used collected data are appended to URL after question mark, while with HTTP POST method collected data are sent separate from URL. **The advantages of HTTP GET method** because of including sent data in URL are:
  - Saving and bookmarking URL withdata collected from Web form, which is the main reason for using GET in search engines such as Google and Yahoo.
  - Data can be manually typed in URL, which is useful for testing servlets.

  The advantages of HTTP POST method because of separating sent data from URL are:
  - Transmission of large portions of data that are separated from URL, which is the only option for uploading large files.
  - Sending binary data that should not be encoded.
  - Preserving privacy of data since they are separated from URL that appears in address line of Web browser. This is especially important for handling passwords and data that need higher level of security.
- **`ENCTYPE`** specifies how data should be encoded before transmitting them. Default encoding in Web is UTF-8. Additional **`ENCTYPE`** is **`multipart/form-data`** that transmits each field as a separate part of MIME compatible document, assuming use of POST method.

Text controls

HTML supports three types of text-input controls: textfields, password fields, and text areas. Each control is identified with the name, while the value is extracted from the control field within a Web form.

`Textfield` is used for collecting a single line of text from a user. The control does not have ending tag. The syntax is:

```
<INPUT TYPE="TEXT" NAME="firstName" ...>
```

The most important attributes are:

- **`NAME`** specifies the name of a control, which will be used for accessing the value in server side processing of submitted data.
- **`VALUE`** specifies initial value of the field.
- **`SIZE`** specifies the width of the control based on the average control width. If the text is larger, it will scroll.
- **`MAXLENGTH`** specifies the maxim number of characters that will be accepted.

`Password` is used for collecting text from users, assuming that characters are not echoing. Instead of typed characters specific charter appears, usually asterisk character (*). It is useful for collecting passwords, credit card numbers or other data requiring higher level of security. The control does not have ending tag. The syntax is:

```
<INPUT TYPE="PASSWORD" NAME="creditCardNumber" ...>
```

It uses the same attributes as simple text field.

`Text Area` is used for collecting multiline text from users. The syntax is:

```
<TEXTAREA NAME="..." ROWS=xxx COLS=yyy>
...
</TEXTAREA>
```

The control does not have **VALUE** attribute, but the initial value is the text between starting and ending tags. In addition to the name of the control, required attributes are **ROWS** and **COLS** for specifying the number of visible rows and the visible width of the area. If the text is higher than it is specified, vertical scrollbar appears.

Push Buttons

Two main types of push buttons are used for submitting the content of Web form to server side program and for resetting the values in form controls to initial values. Push buttons adopt look and feel of other elements of client operating system.

`Submit Buttons` are used for submitting values from controls in Web forms to server side programs. The control does not have ending tag. The syntax is:

```
<INPUT TYPE="SUBMIT" VALUE="Confirm" ...>
```

The main attributes are **NAME** and **VALUE**. **NAME** is used for identifying a button in server side programs , which enable including several **SUBMIT** buttons for handling different types of actions in a single Web form. **VALUE** is used for presenting a text in the button label. For example, several submit buttons in a web form are presented in the following listing.

```
<INPUT TYPE="TEXT" NAME="Item" VALUE="256MB SIMM"><BR>
<INPUT TYPE="SUBMIT" NAME="Add" VALUE="Add Item to Cart">
<INPUT TYPE="SUBMIT" NAME="Delete" VALUE="Delete Item from Cart">
```

Servlet code for handling different submit buttons is then presented in the following listing:

```
if (request.getParameter("Add") != null) {
      doCartAdditionOperation(...);
} else if (request.getParameter("Delete") != null) {
      doCartDeletionOperation(...);
}
```

`Reset Buttons` are used for resetting values in all form fields to values specified in attribute **VALUE**. The control does not have ending tag. The syntax is:

```
<INPUT TYPE="RESET" VALUE="Clear form" ...>
```

The main attributes are NAME and VALUE.

Check boxes

Check boxes are controls used for selecting among set of predefined choices. Check boxes can be selected or deselected individually. Check box is control whose pair name/value is submitted only if the check box is checked when the form is submitted. The control does not have ending tag. The syntax is:

`<INPUT TYPE="CHECKBOX" NAME="Name" ...>`

The only mandatory attribute is **`NAME`**, while **`VALUE`** can be supplied for handling HTTP request on server in a pair with the value of attribute **`CHECKED`** (on/off). For example if a Web form contain a check box

```
<P>
<INPUT TYPE="CHECKBOX" NAME="noEmail" CHECKED>
Check here if you do <I>not</I> want to get our email newsletter
<P>
```

in the URL submitted to Web server will be transmitted a pair **`noEmail=on`**.

Radio buttons

Radio buttons are controls used for selecting among set of predefined choices. Radio buttons are grouped so that only one option in the group can be selected at a time. Group of radio buttons is indicated with the attribute **`NAME`** in all buttons belonging to the group. Only one button can be selected (pressed) at a time, and the **`VALUE`** of pressed button is transmitted when the form is submitted. The control does not have ending tag. The syntax is:

`<INPUT TYPE="RADIO" NAME="groupName" VALUE="buttonName" ...>`

For example, if we need to select the type of credit card HTML code is

```
<INPUT TYPE="RADIO" NAME="creditCard" VALUE="visa">Visa<br>
<INPUT TYPE="RADIO" NAME="creditCard" VALUE="mastercard">Master Card<br>
<INPUT TYPE="RADIO" NAME="creditCard" VALUE="amex">American Express
```

When submitting Web form selected option (**Visa**) will be added to URL string as a pair **`creditCard=visa`**.

Combo boxes

A **`SELECT`** element presents a set of options to the user. For combo boxes, implemented as drop-down menus, only a single entry can be selected and no visible size has been specified. The choices themselves are specified by **`OPTION`** entries embedded in the **`SELECT`** element. Example is:

```
<SELECT NAME="creditCard">
      <OPTION VALUE="visa">Visa
      <OPTION VALUE="mastercard">Master Card
      <OPTION VALUE="amex">American Express
</SELECT>
```

When submitting Web form selected option (**Master Card**) will be added to URL string as a pair **`creditCard=mastercard`**.

List box

The second option for using SELECT element is a list box, enabling multiple selection. If more than one entry is active when the form is submitted, then more than one value is sent, listed as separate entries (repeating **`NAME`**). Example is:

```
<SELECT NAME="language" MULTIPLE>
  <OPTION VALUE="c">C
  <OPTION VALUE="c++">C++
  <OPTION VALUE="java" SELECTED>Java
</SELECT>
```

When submitting Web form selected options (**C, Java**) will be added to URL string as two pairs with the same name **`language:`**

`language=c&language=java`

Reading Form Data with Servlets

The main issue in server side programming is how to parse strings with data received from Web forms. Servlets introduce several useful improvements:

- Servlets enable a common way for parsing these strings in both cases when they are received through HTTP GET and HTTP POST.
- There is no need to parse complicated strings containing & character for separating parameters. Parameters can be easily accessed by using servlet methods.
- All characters are sent unchanged nevertheless if they are special characters or if specific character encoding is used.
- There is no need for special code for handling cases when some parameters are omitted.

Servlets automatically handle parsing of request strings by calling:

- **Reading a single value:** **`request.getParameter`** method for getting values for each parameter. If the parameter does not exists the returned value is **`null`**.
- **Reading multiple values:** **`request.getParameterValues`** method if the parameter appears more than once. However, when designing Web forms it is advisable to ensure unique names for all input elements.
- **Looking up parameter names:** **`request.getParameterNames`** method for getting a complete list of all parameters in request, which is useful in the cases when servlet do not know exact names of parameters, or for debugging.
- **Reading raw form data and uploaded files:** **`getReader`** or **`getInputStream`** methods for handling raw request data, which is useful in the following situations: (1) when data comes from custom client and not from Web forms (e.g. data from Applets), and (2) when data are from uploaded files if Web form element is of FILE type (**`<INPUT TYPE="FILE"...>`**). For uploading files it is necessary to use third-part libraries, such as **Apache Commons FileUpload** (http://commons.apache.org/proper/commons-fileupload/).

Literature and Links

[1] **The Apache Tomcat**. http://tomcat.apache.org/

[2] Marty Hall and Larry Brown. *Core Servlets and JavaServer Pages, Free Online Version of Second Edition*. http://pdf.coreservlets.com/. Chapter 19: Creating and Processing HTML Forms.

[3] Marty Hall and Larry Brown. *Core Servlets and JavaServer Pages, Free Online Version of Second Edition*. http://pdf.coreservlets.com/. Chapter 4: Handling the Client Request: Form Data.

[4] https://tomcat.apache.org/tomcat-9.0-doc/servletapi/overview-summary.html

[5] https://tomcat.apache.org/tomcat-9.0-doc/servletapi/index.html.

[6] **Apache Commons FileUpload.** http://commons.apache.org/proper/commons-fileupload/.

[7] **W3Schools. HTML Forms**. https://www.w3schools.com/html/html_forms.asp.