4.1 The Role of Form Data

If you've ever used a search engine, visited an online bookstore, tracked stocks on the Web, or asked a Web-based site for quotes on plane tickets, you've probably seen funny-looking URLs like `http://host/path?user=Marty+Hall&origin=bwi&dest=sfo`. The part after the question mark (i.e., `user=Marty+Hall&origin=bwi&dest=sfo`) is known as *form data* (or *query data*) and is the most common way to get information from a Web page to a server-side program. Form data can be attached to the end of the URL after a question mark (as above) for `GET` requests; form data can also be sent to the server on a separate line for `POST` requests. If you're not familiar with HTML forms, Chapter 19 (Creating and Processing HTML Forms) gives details on how to build forms that collect and transmit data of this sort. However, here are the basics.

1. **Use the `FORM` element to create an HTML form.** Use the `ACTION` attribute to designate the address of the servlet or JSP page that will process the results; you can use an absolute or relative URL. For example:

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

   If `ACTION` is omitted, the data is submitted to the URL of the current page.

2. **Use input elements to collect user data.** Place the elements between the start and end tags of the `FORM` element and give each input element a `NAME`. Textfields are the most common input element; they are created with the following.

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

3. **Place a submit button near the bottom of the form.** For example:

   ```
   <INPUT TYPE="SUBMIT">
   ```

   When the button is pressed, the URL designated by the form's `ACTION` is invoked. With `GET` requests, a question mark and name/value pairs are attached to the end of the URL, where the names come from the `NAME` attributes in the HTML input elements and the values come from the end user. With `POST` requests, the same data is sent, but on a separate request line instead of attached to the URL.

Extracting the needed information from this form data is traditionally one of the most tedious parts of server-side programming.

First of all, before servlets you generally had to read the data one way for `GET` requests (in traditional CGI, this is usually through the `QUERY_STRING` environment variable) and a different way for `POST` requests (by reading the standard input in traditional CGI).

Second, you have to chop the pairs at the ampersands, then separate the parameter names (left of the equal signs) from the parameter values (right of the equal signs).

Third, you have to *URL-decode* the values: reverse the encoding that the browser uses on certain characters. Alphanumeric characters are sent unchanged by the browser, but spaces are converted to plus signs and other characters are converted to %*XX*, where *XX* is the ASCII (or ISO Latin-1) value of the character, in hex. For example, if someone enters a value of "`~hall, ~gates, and ~mcnealy`" into a textfield with the name `users` in an HTML form, the data is sent as "`users=%7Ehall%2C+%7Egates%2C+and+%7Emcnealy`", and the server-side program has to reconstitute the original string.

Finally, the fourth reason that it is tedious to parse form data with traditional server-side technologies is that values can be omitted (e.g., "`param1=val1& `**`param2=`**`&param3=val3`") or a parameter can appear more than once (e.g., "**`param1=val1`**`&param2=val2&`**`param1=val3`**"), so your parsing code needs special cases for these situations.

Fortunately, servlets help us with much of this tedious parsing. That's the topic of the next section.

[Team LiB] PREVIOUS NEXT