4.2 Reading Form Data from Servlets

One of the nice features of servlets is that all of this form parsing is handled automatically. You call `request.getParameter` to get the value of a form parameter. You can also call `request.getParameterValues` if the parameter appears more than once, or you can call `request.getParameterNames` if you want a complete list of all parameters in the current request. In the rare cases in which you need to read the raw request data and parse it yourself, call `getReader` or `getInputStream`.

Reading Single Values: getParameter

To read a request (form) parameter, you simply call the `getParameter` method of `HttpServletRequest`, supplying the case-sensitive parameter name as an argument. You supply the parameter name exactly as it appeared in the HTML source code, and you get the result exactly as the end user entered it; any necessary URL-decoding is done automatically. Unlike the case with many alternatives to servlet technology, you use `getParameter` exactly the same way when the data is sent by `GET` (i.e., from within the `doGet` method) as you do when it is sent by `POST` (i.e., from within `doPost`); the servlet knows which request method the client used and automatically uses the appropriate method to read the data. An empty `String` is returned if the parameter exists but has no value (i.e., the user left the corresponding textfield empty when submitting the form), and `null` is returned if there was no such parameter.

Parameter names are case sensitive so, for example, `request.getParameter("Param1")` and `request.getParameter("param1")` are *not* interchangeable.

Core Warning

The values supplied to `getParameter` and `getParameterValues` are case sensitive.

Reading Multiple Values: getParameterValues

If the same parameter name might appear in the form data more than once, you should call `getParameterValues` (which returns an array of strings) instead of `getParameter` (which returns a single string corresponding to the first occurrence of the parameter). The return value of `getParameterValues` is `null` for nonexistent parameter names and is a one-element array when the parameter has only a single value.

Now, if you are the author of the HTML form, it is usually best to ensure that each textfield, checkbox, or other user interface element has a unique name. That way, you can just stick with the simpler `getParameter` method and avoid `getParameterValues` altogether. However, you sometimes write servlets or JSP pages that handle other people's HTML forms, so you have to be able to deal with all possible cases. Besides, multiselectable list boxes (i.e., HTML `SELECT` elements with the `MULTIPLE` attribute set; see Chapter 19 for details) repeat the parameter name for each selected element in the list. So, you cannot always avoid multiple values.

Looking Up Parameter Names: getParameterNames and getParameterMap

Most servlets look for a specific set of parameter names; in most cases, if the servlet does not know the name of the parameter, it does not know what to do with it either. So, your primary

tool should be `getParameter`. However, it is sometimes useful to get a full list of parameter names. The primary utility of the full list is debugging, but you occasionally use the list for applications where the parameter names are very dynamic. For example, the names themselves might tell the system what to do with the parameters (e.g., `row-1-col-3-value`), the system might build a database update assuming that the parameter names are database column names, or the servlet might look for a few specific names and then pass the rest of the names to another application.

Use `getParameterNames` to get this list in the form of an `Enumeration`, each entry of which can be cast to a `String` and used in a `getParameter` or `getParameterValues` call. If there are no parameters in the current request, `getParameterNames` returns an empty `Enumeration` (not `null`). Note that `Enumeration` is an interface that merely guarantees that the actual class will have `hasMoreElements` and `nextElement` methods: there is no guarantee that any particular underlying data structure will be used. And, since some common data structures (hash tables, in particular) scramble the order of the elements, you should not count on `getParameterNames` returning the parameters in the order in which they appeared in the HTML form.

Core Warning

Don't count on `getParameterNames` returning the names in any particular order.

An alternative to `getParameterNames` is `getParameterMap`. This method returns a `Map`: the parameter names (strings) are the table keys and the parameter values (string arrays as returned by `getParameterNames`) are the table values.

Reading Raw Form Data and Parsing Uploaded Files: getReader or getInputStream

Rather than reading individual form parameters, you can access the query data directly by calling `getReader` or `getInputStream` on the `HttpServletRequest` and then using that stream to parse the raw input. Note, however, that if you read the data in this manner, it is not guaranteed to be available with `getParameter`.

Reading the raw data is a bad idea for regular parameters since the input is neither parsed (separated into entries specific to each parameter) nor URL-decoded (translated so that plus signs become spaces and %*XX* is replaced by the original ASCII or ISO Latin-1 character corresponding to the hex value *XX*). However, reading the raw input is of use in two situations.

The first case in which you might read and parse the data yourself is when the data comes from a custom client rather than by an HTML form. The most common custom client is an applet; applet-servlet communication of this nature is discussed in Volume 2 of this book.

The second situation in which you might read the data yourself is when the data is from an uploaded file. HTML supports a `FORM` element (`<INPUT TYPE="FILE"...>`) that lets the client upload a file to the server. Unfortunately, the servlet API defines no mechanism to read such files. So, you need a third-party library to do so. One of the most popular ones is from the Apache Jakarta Project. See http://jakarta.apache.org/commons/fileupload/ for details.

Reading Input in Multiple Character Sets: setCharacterEncoding

By default, `request.getParameter` interprets input using the server's current character set. To change this default, use the `setCharacterEncoding` method of `ServletRequest`. But, what if

input could be in more than one character set? In such a case, you cannot simply call `setCharacterEncoding` with a normal character set name. The reason for this restriction is that `setCharacterEncoding` must be called *before* you access any request parameters, and in many cases you use a request parameter (e.g., a checkbox) to determine the character set.

So, you are left with two choices: read the parameter in one character set and convert it to another, or use an autodetect feature provided with some character sets.

For the first option, you would read the parameter of interest, use `getBytes` to extract the raw bytes, then pass those bytes to the `String` constructor along with the name of the desired character set. Here is an example that converts a parameter to Japanese:

```
String firstNameWrongEncoding = request.getParameter("firstName");
String firstName =
  new String(firstNameWrongEncoding.getBytes(), "Shift_JIS");
```

For the second option, you would use a character set that supports detection and conversion from the default set. A full list of character sets supported in Java is available at http://java.sun.com/j2se/1.4.1/docs/guide/intl/encoding.doc.html. For example, to allow input in either English or Japanese, you might use the following.

```
request.setCharacterEncoding("JISAutoDetect");
String firstName = request.getParameter("firstName");
```

[Team LiB]

PREVIOUS NEXT