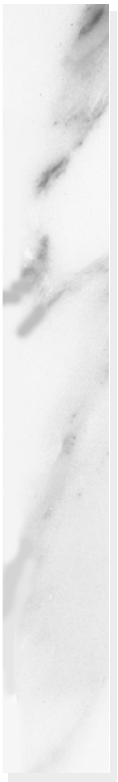
ACCESSING THE STANDARD CGI VARIABLES



Topics in This Chapter

- The idea of "CGI variables"
- The servlet equivalent of each standard CGI variable
- A servlet that shows the values of all CGI variables

Chapter 5

If you come to servlets with a background in traditional Common Gateway Interface (CGI) programming, you are probably used to the idea of "CGI variables." These are a somewhat eclectic collection of information about the current request. Some are based on the HTTP request line and headers (e.g., form data), others are derived from the socket itself (e.g., the name and IP address of the requesting host), and still others are taken from server installation parameters (e.g., the mapping of URLs to actual paths).

Although it probably makes more sense to think of different sources of data (request data, server information, etc.) as distinct, experienced CGI programmers may find it useful to see the servlet equivalent of each of the CGI variables. If you don't have a background in traditional CGI, first, count your blessings; servlets are easier to use, more flexible and more efficient than standard CGI. Second, just skim this chapter, noting the parts not directly related to the incoming HTTP request. In particular, observe that you can use `getServletContext().getRealPath` to map a URI (the part of the URL that comes after the host and port) to an actual path and that you can use `request.getRemoteHost` and `request.getRemoteAddress` to get the name and IP address of the client.

5.1 Servlet Equivalent of CGI Variables

For each standard CGI variable, this section summarizes its purpose and the means of accessing it from a servlet. As usual, once you are familiar with this information, you may want to use Appendix A (Servlet and JSP Quick Reference) as a reminder. Assume `request` is the `HttpServletRequest` supplied to the `doGet` and `doPost` methods.

AUTH_TYPE

If an `Authorization` header was supplied, this variable gives the scheme specified (`basic` or `digest`). Access it with `request.getAuthType()`.

CONTENT_LENGTH

For `POST` requests only, this variable stores the number of bytes of data sent, as given by the `Content-Length` request header. Technically, since the `CONTENT_LENGTH` CGI variable is a string, the servlet equivalent is `String.valueOf(request.getContentLength())` or `request.getHeader("Content-Length")`. You'll probably want to just call `request.getContentLength()`, which returns an `int`.

CONTENT_TYPE

`CONTENT_TYPE` designates the MIME type of attached data, if specified. See Table 7.1 in Section 7.2 (HTTP 1.1 Response Headers and Their Meaning) for the names and meanings of the common MIME types. Access `CONTENT_TYPE` with `request.getContentType()`.

DOCUMENT_ROOT

The `DOCUMENT_ROOT` variable specifies the real directory corresponding to the URL `http://host/`. Access it with `getServletContext().getRealPath("/")`. In older servlet specifications you accessed this variable with `request.getRealPath("/")`; the older access method is no longer supported. Also, you can use `getServletContext().getRealPath` to map an arbitrary URI (i.e., URL suffix that comes after the hostname and port) to an actual path on the local machine.

HTTP_XXX_YYY
Variables of the form `HTTP_HEADER_NAME` were how CGI programs obtained access to arbitrary HTTP request headers. The `Cookie` header became `HTTP_COOKIE`, `User-Agent` became `HTTP_USER_AGENT`, `Referer` became `HTTP_REFERER`, and so forth. Servlets should just use `request.getHeader` or one of the shortcut methods described in Chapter 4 (Handling the Client Request: HTTP Request Headers).

PATH_INFO
This variable supplies any path information attached to the URL after the address of the servlet but before the query data. For example, with `http://host/servlet/coreservlets.SomeServlet/foo/bar?baz=quux`, the path information is `/foo/bar`. Since servlets, unlike standard CGI programs, can talk directly to the server, they don't need to treat path information specially. Path information could be sent as part of the regular form data and then translated by `getServletContext().getRealPath`. Access the value of `PATH_INFO` by using `request.getPathInfo()`.

PATH_TRANSLATED
`PATH_TRANSLATED` gives the path information mapped to a real path on the server. Again, with servlets there is no need to have a special case for path information, since a servlet can call `getServletContext().getRealPath` to translate partial URLs into real paths. This translation is not possible with standard CGI because the CGI program runs entirely separately from the server. Access this variable by means of `request.getPathTranslated()`.

QUERY_STRING
For `GET` requests, this variable gives the attached data as a single string with values still URL-encoded. You rarely want the raw data in servlets; instead, use `request.getParameter` to access individual parameters, as described in Chapter 3 (Handling the Client Request: Form Data). However, if you do want the raw data, you can get it via `request.getQueryString()`.

REMOTE_ADDR
This variable designates the IP address of the client that made the request, as a `String` (e.g., `"198.137.241.30"`). Access it by calling `request.getRemoteAddr()`.

REMOTE_HOST

`REMOTE_HOST` indicates the fully qualified domain name (e.g., `whitehouse.gov`) of the client that made the request. The IP address is returned if the domain name cannot be determined. You can access this variable with `request.getRemoteHost()`.

REMOTE_USER

If an `Authorization` header was supplied and decoded by the server itself, the `REMOTE_USER` variable gives the user part, which is useful for session tracking in protected sites. Access it with `request.getRemoteUser()`. For decoding `Authorization` information directly in servlets, see Section 4.5 (Restricting Access to Web Pages).

REQUEST_METHOD

This variable stipulates the HTTP request type, which is usually `GET` or `POST` but is occasionally `HEAD`, `PUT`, `DELETE`, `OPTIONS`, or `TRACE`. Servlets rarely need to look up `REQUEST_METHOD` explicitly, since each of the request types is typically handled by a different servlet method (`doGet`, `doPost`, etc.). An exception is `HEAD`, which is handled automatically by the `service` method returning whatever headers and status codes the `doGet` method would use. Access this variable by means of `request.getMethod()`.

SCRIPT_NAME

This variable specifies the path to the servlet, relative to the server's root directory. It can be accessed through `request.getServletPath()`.

SERVER_NAME

`SERVER_NAME` gives the host name of the server machine. It can be accessed by means of `request.getServerName()`.

SERVER_PORT

This variable stores the port the server is listening on. Technically, the servlet equivalent is `String.valueOf(request.getServerPort())`, which returns a `String`. You'll usually just want `request.getServerPort()`, which returns an `int`.

SERVER_PROTOCOL
The `SERVER_PROTOCOL` variable indicates the protocol name and version used in the request line (e.g., `HTTP/1.0` or `HTTP/1.1`). Access it by calling `request.getProtocol()`.

SERVER_SOFTWARE
This variable gives identifying information about the Web server. Access it by means of `getServletContext().getServerInfo()`.

5.2 A Servlet That Shows the CGI Variables

Listing 5.1 presents a servlet that creates a table showing the values of all the CGI variables other than `HTTP_XXX_YYY`, which are just the HTTP request headers described in Chapter 4. Figure 5–1 shows the result for a typical request.

Listing 5.1 `ShowCGIVariables.java`

```
package coreservlets;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

/** Creates a table showing the current value of each
 *  of the standard CGI variables.
 */

public class ShowCGIVariables extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String[][] variables =
      { { "AUTH_TYPE", request.getAuthType() },
        { "CONTENT_LENGTH",
          String.valueOf(request.getContentLength()) },
        { "CONTENT_TYPE", request.getContentType() },
```

Listing 5.1 ShowCGIVariables.java (continued)

```
        { "DOCUMENT_ROOT",
          getServletContext().getRealPath("/") },
        { "PATH_INFO", request.getPathInfo() },
        { "PATH_TRANSLATED", request.getPathTranslated() },
        { "QUERY_STRING", request.getQueryString() },
        { "REMOTE_ADDR", request.getRemoteAddr() },
        { "REMOTE_HOST", request.getRemoteHost() },
        { "REMOTE_USER", request.getRemoteUser() },
        { "REQUEST_METHOD", request.getMethod() },
        { "SCRIPT_NAME", request.getServletPath() },
        { "SERVER_NAME", request.getServerName() },
        { "SERVER_PORT",
          String.valueOf(request.getServerPort()) },
        { "SERVER_PROTOCOL", request.getProtocol() },
        { "SERVER_SOFTWARE",
          getServletContext().getServerInfo() }
      };
    String title = "Servlet Example: Showing CGI Variables";
    out.println(ServletUtilities.headWithTitle(title) +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H1 ALIGN=CENTER>" + title + "</H1>\n" +
                "<TABLE BORDER=1 ALIGN=CENTER>\n" +
                "<TR BGCOLOR=\"#FFAD00\">\n" +
                "<TH>CGI Variable Name<TH>Value");
    for(int i=0; i<variables.length; i++) {
      String varName = variables[i][0];
      String varValue = variables[i][1];
      if (varValue == null)
        varValue = "<I>Not specified</I>";
      out.println("<TR><TD>" + varName + "<TD>" + varValue);
    }
    out.println("</TABLE></BODY></HTML>");
  }

  /** POST and GET requests handled identically. */

  public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
      throws ServletException, IOException {
    doGet(request, response);
  }
}
```

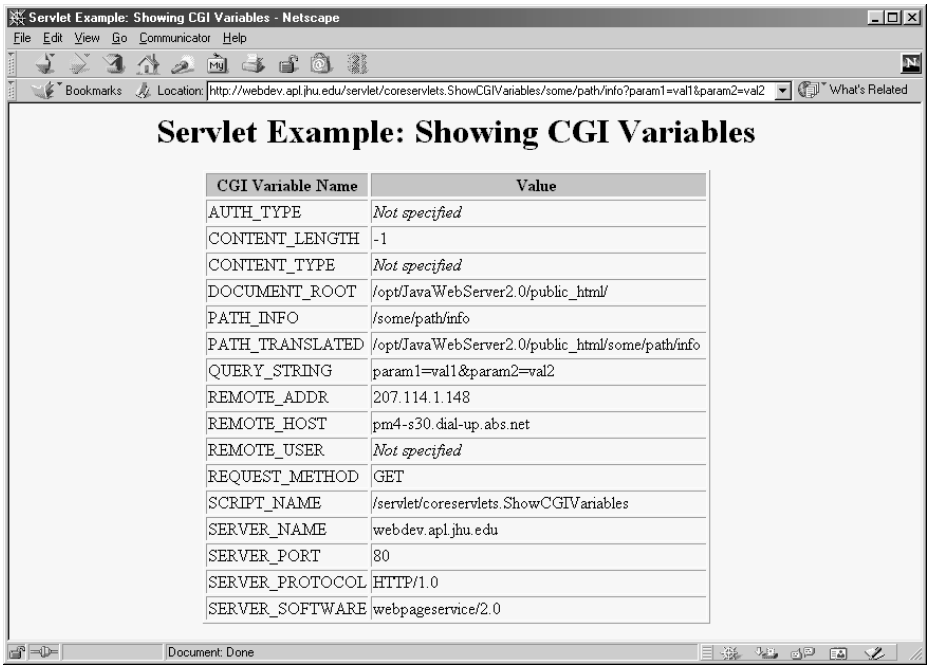


Figure 5–1 The standard CGI variables for a typical request.