GENERATING THE SERVER RESPONSE: HTTP STATUS CODES

Topics in This Chapter

- The purpose of HTTP status codes
- The way to specify status codes from servlets
- The meaning of each of the HTTP 1.1 status code values
- A servlet that uses status codes to redirect users to other sites and to report errors

Chapter 6

When a Web server responds to a request from a browser or other Web client, the response typically consists of a status line, some response headers, a blank line, and the document. Here is a minimal example:

```
HTTP/1.1 200 OK
Content-Type: text/plain

Hello World
```

The status line consists of the HTTP version (`HTTP/1.1` in the example above), a status code (an integer; 200 in the above example), and a very short message corresponding to the status code (`OK` in the example). In most cases, all of the headers are optional except for `Content-Type`, which specifies the MIME type of the document that follows. Although most responses contain a document, some don't. For example, responses to `HEAD` requests should never include a document, and there are a variety of status codes that essentially indicate failure and either don't include a document or include only a short error message document.

Servlets can perform a variety of important tasks by manipulating the status line and the response headers. For example, they can forward the user to other sites; indicate that the attached document is an image, Adobe Acrobat file, or HTML file; tell the user that a password is required to access the document; and so forth. This chapter discusses the various status codes and what

can be accomplished with them, and the following chapter discusses the response headers.

6.1 Specifying Status Codes

As just described, the HTTP response status line consists of an HTTP version, a status code, and an associated message. Since the message is directly associated with the status code and the HTTP version is determined by the server, all a servlet needs to do is to set the status code. The way to do this is by the `setStatus` method of `HttpServletResponse`. If your response includes a special status code *and* a document, be sure to call `setStatus` *before* actually returning any of the content via the `PrintWriter`. That's because an HTTP response consists of the status line, one or more headers, a blank line, and the actual document, *in that order*. The headers can appear in any order, and servlets buffer the headers and send them all at once, so it is legal to set the status code (part of the first line returned) even after setting headers. But servlets do not necessarily buffer the document itself, since users might want to see partial results for long pages. In version 2.1 of the servlet specification, the `PrintWriter` output is not buffered at all, so the first time you use the `PrintWriter`, it is too late to go back and set headers. In version 2.2, servlet engines are permitted to partially buffer the output, but the size of the buffer is left unspecified. You can use the `getBufferSize` method of `HttpServletResponse` to determine the size, or use `setBufferSize` to specify it. In version 2.2 with buffering enabled, you can set status codes until the buffer fills up and is actually sent to the client. If you aren't sure if the buffer has been sent, you can use the `isCommitted` method to check.

Core Approach

*Be sure to set status codes **before** sending any document content to the client.*

The `setStatus` method takes an `int` (the status code) as an argument, but instead of using explicit numbers, it is clearer and more reliable to use the constants defined in `HttpServletResponse`. The name of each constant is derived from the standard HTTP 1.1 message for each constant, all uppercase with a prefix of `SC` (for *Status Code*) and spaces changed to underscores. Thus, since the message for 404 is "Not Found," the equivalent constant in `HttpServletResponse` is `SC_NOT_FOUND`. In version 2.1 of the servlet specification, there are three exceptions. The constant for code 302 is derived from the HTTP 1.0 message (Moved Temporarily), not the HTTP 1.1 message (Found), and the constants for codes 307 (Temporary Redirect) and 416 (Requested Range Not Satisfiable) are missing altogether. Version 2.2 added the constant for 416, but the inconsistencies for 307 and 302 remain.

Although the general method of setting status codes is simply to call `response.setStatus(int)`, there are two common cases where a shortcut method in `HttpServletResponse` is provided. Just be aware that both of these methods throw `IOException`, whereas `setStatus` doesn't.

- **`public void sendError(int code, String message)`**
  The `sendError` method sends a status code (usually 404) along with a short message that is automatically formatted inside an HTML document and sent to the client.
- **`public void sendRedirect(String url)`**
  The `sendRedirect` method generates a 302 response along with a `Location` header giving the URL of the new document. With servlets version 2.1, this must be an absolute URL. In version 2.2, either an absolute or a relative URL is permitted and the system automatically translates relative URLs into absolute ones before putting them in the `Location` header.

Setting a status code does not necessarily mean that you don't need to return a document. For example, although most servers automatically generate a small "File Not Found" message for 404 responses, a servlet might want to customize this response. Remember that if you do send output, you have to call `setStatus` or `sendError` *first*.

6.2 HTTP 1.1 Status Codes and Their Purpose

The following sections describe each of the status codes available for use in servlets talking to HTTP 1.1 clients, along with the standard message associated with each code. A good understanding of these codes can dramatically increase the capabilities of your servlets, so you should at least skim the descriptions to see what options are at your disposal. You can come back to get details when you are ready to make use of some of the capabilities. Note that Appendix A (Servlet and JSP Quick Reference) presents a brief summary of these codes in tabular format.

The complete HTTP 1.1 specification is given in RFC 2616, which you can access on-line by going to `http://www.rfc-editor.org/` and following the links to the latest RFC archive sites. Codes that are new in HTTP 1.1 are noted, since many browsers support only HTTP 1.0. You should only send the new codes to clients that support HTTP 1.1, as verified by checking `request.getRequestProtocol`.

The rest of this section describes the specific status codes available in HTTP 1.1. These codes fall into five general categories:

- **100-199**
  Codes in the 100s are informational, indicating that the client should respond with some other action.
- **200-299**
  Values in the 200s signify that the request was successful.
- **300-399**
  Values in the 300s are used for files that have moved and usually include a `Location` header indicating the new address.
- **400-499**
  Values in the 400s indicate an error by the client.
- **500-599**
  Codes in the 500s signify an error by the server.

The constants in `HttpServletResponse` that represent the various codes are derived from the standard messages associated with the codes. In servlets, you usually refer to status codes only by means of these constants. For example, you would use `response.setStatus(response.SC_NO_CONTENT)` rather than `response.setStatus(204)`, since the latter is unclear to readers and is prone to typographical errors.

However, you should note that servers are allowed to vary the messages slightly, and clients pay attention only to the numeric value. So, for example, you might see a server return a status line of `HTTP/1.1 200 Document Follows` instead of `HTTP/1.1 200 OK`.

100 (Continue)

If the server receives an `Expect` request header with a value of `100-continue`, it means that the client is asking if it can send an attached document in a follow-up request. In such a case, the server should either respond with status 100 (`SC_CONTINUE`) to tell the client to go ahead or use 417 (`Expectation Failed`) to tell the browser it won't accept the document. This status code is new in HTTP 1.1.

101 (Switching Protocols)

A 101 (`SC_SWITCHING_PROTOCOLS`) status indicates that the server will comply with the `Upgrade` header and change to a different protocol. This status code is new in HTTP 1.1.

200 (OK)

A value of 200 (`SC_OK`) means that everything is fine. The document follows for `GET` and `POST` requests. This status is the default for servlets; if you don't use `setStatus`, you'll get 200.

201 (Created)

A status code of 201 (`SC_CREATED`) signifies that the server created a new document in response to the request; the `Location` header should give its URL.

202 (Accepted)

A value of 202 (`SC_ACCEPTED`) tells the client that the request is being acted upon, but processing is not yet complete.

203 (Non-Authoritative Information)

A 203 (`SC_NON_AUTHORITATIVE_INFORMATION`) status signifies that the document is being returned normally, but some of the response headers might be incorrect since a document copy is being used. This status code is new in HTTP 1.1.

204 (No Content)

A status code of 204 (`SC_NO_CONTENT`) stipulates that the browser should continue to display the previous document because no new document is available. This behavior is useful if the user periodically reloads a page by pressing the "Reload" button, and you can determine that the previous page is already up-to-date. For example, a servlet might do something like this:

```
int pageVersion =
  Integer.parseInt(request.getParameter("pageVersion"));
if (pageVersion >= currentVersion) {
  response.setStatus(response.SC_NO_CONTENT);
} else {
  // Create regular page
}
```

However, this approach does not work for pages that are automatically reloaded via the `Refresh` response header or the equivalent `<META HTTP-EQUIV="Refresh" ...>` HTML entry, since returning a 204 status code stops future reloading. JavaScript-based automatic reloading could still work in such a case, though. See the discussion of `Refresh` in Section 7.2 (HTTP 1.1 Response Headers and Their Meaning) for details.

205 (Reset Content)

A value of 205 (`SC_RESET_CONTENT`) means that there is no new document, but the browser should reset the document view. This status code is used to force browsers to clear form fields. It is new in HTTP 1.1.

206 (Partial Content)

A status code of 206 (`SC_PARTIAL_CONTENT`) is sent when the server fulfills a partial request that includes a `Range` header. This value is new in HTTP 1.1.

300 (Multiple Choices)

A value of 300 (`SC_MULTIPLE_CHOICES`) signifies that the requested document can be found several places, which will be listed in the returned document. If the server has a preferred choice, it should be listed in the `Location` response header.

301 (Moved Permanently)

The 301 (`SC_MOVED_PERMANENTLY`) status indicates that the requested document is elsewhere; the new URL for the document is given in the `Location` response header. Browsers should automatically follow the link to the new URL.

302 (Found)

This value is similar to 301, except that the URL given by the `Location` header should be interpreted as a temporary replacement, not a permanent one. Note: in HTTP 1.0, the message was `Moved Temporarily` instead of `Found`, and the constant in `HttpServletResponse` is `SC_MOVED_TEMPORARILY`, not the expected `SC_FOUND`.

Core Note

The constant representing 302 is `SC_MOVED_TEMPORARILY`*, not* `SC_FOUND`.

Status code 302 is very useful because browsers automatically follow the reference to the new URL given in the `Location` response header. It is so useful, in fact, that there is a special method for it, `sendRedirect`. Using `response.sendRedirect(url)` has a couple of advantages over using `response.setStatus(response.SC_MOVED_TEMPORARILY)` and `response.setHeader("Location", url)`. First, it is shorter and easier. Second, with `sendRedirect`, the servlet automatically builds a page containing the link to show to older browsers that don't automatically follow redirects. Finally, with version 2.2 of servlets (the version in J2EE), `sendRedirect` can handle relative URLs, automatically translating them into absolute ones. You must use an absolute URL in version 2.1, however.

If you redirect the user to another page within your own site, you should pass the URL through the `encodeURL` method of `HttpServletResponse`. Doing so is a simple precaution in case you ever use session tracking based on URL-rewriting. URL-rewriting is a way to track users who have cookies disabled while they are at your site. It is implemented by adding extra path information to the end of each URL, but the servlet session-tracking API takes care of the details automatically. Session

tracking is discussed in Chapter 9, and it is a good idea to use `encodeURL` routinely so that you can add session tracking at a later time with minimal changes to the code.

Core Approach

If you redirect users to a page within your site, plan ahead for session tracking by using
`response.sendRedirect(response.encodeURL(url))`,
rather than just
`response.sendRedirect(url)`.

This status code is sometimes used interchangeably with 301. For example, if you erroneously ask for `http://host/~user` (missing the trailing slash), some servers will reply with a 301 code while others will use 302.

Technically, browsers are only supposed to automatically follow the redirection if the original request was `GET`. For details, see the discussion of the 307 status code.

303 (See Other)

The 303 (`SC_SEE_OTHER`) status is similar to 301 and 302, except that if the original request was `POST`, the new document (given in the `Location` header) should be retrieved with `GET`. This code is new in HTTP 1.1.

304 (Not Modified)

When a client has a cached document, it can perform a conditional request by supplying an `If-Modified-Since` header to indicate that it only wants the document if it has been changed since the specified date. A value of 304 (`SC_NOT_MODIFIED`) means that the cached version is up-to-date and the client should use it. Otherwise, the server should return the requested document with the normal (200) status code. Servlets normally should not set this status code directly. Instead, they should implement the `getLastModified` method and let the default `service` method handle conditional requests based upon this modification date. An example of this approach is given in Section 2.8 (An Example Using Servlet Initialization and Page Modification Dates).

305 (Use Proxy)

A value of 305 (`SC_USE_PROXY`) signifies that the requested document should be retrieved via the proxy listed in the `Location` header. This status code is new in HTTP 1.1.

307 (Temporary Redirect)

The rules for how a browser should handle a 307 status are identical to those for 302. The 307 value was added to HTTP 1.1 since many browsers erroneously follow the redirection on a 302 response even if the original message is a `POST`. Browsers are supposed to follow the redirection of a `POST` request only when they receive a 303 response status. This new status is intended to be unambiguously clear: follow redirected `GET` *and* `POST` requests in the case of 303 responses; follow redirected `GET` but *not* `POST` requests in the case of 307 responses. Note: For some reason there is no constant in `HttpServletResponse` corresponding to this status code. This status code is new in HTTP 1.1.

Core Note

There is no `SC_TEMPORARY_REDIRECT` *constant in* `HttpServletResponse`*, so you have to use 307 explicitly.*

400 (Bad Request)

A 400 (`SC_BAD_REQUEST`) status indicates bad syntax in the client request.

401 (Unauthorized)

A value of 401 (`SC_UNAUTHORIZED`) signifies that the client tried to access a password-protected page without proper identifying information in the `Authorization` header. The response must include a `WWW-Authenticate` header. For an example, see Section 4.5, "Restricting Access to Web Pages."

403 (Forbidden)

A status code of 403 (`SC_FORBIDDEN`) means that the server refuses to supply the resource, regardless of authorization. This status is often the result of bad file or directory permissions on the server.

404 (Not Found)

The infamous 404 (`SC_NOT_FOUND`) status tells the client that no resource could be found at that address. This value is the standard "no such page" response. It is such a common and useful response that there is a special method for it in the `HttpServletResponse` class: `sendError("message")`. The advantage of `sendError` over `setStatus` is that, with `sendError`, the server automatically generates an error page showing the error message. Unfortunately, however, the default behavior of Internet Explorer 5 is to ignore the error page you send back and displays its own, even though doing so contradicts the HTTP specification. To turn off this setting, go to the Tools menu, select Internet Options, choose the Advanced tab, and make sure "Show friendly HTTP error messages" box is not checked. Unfortunately, however, few users are aware of this setting, so this "feature" prevents most users of Internet Explorer version 5 from seeing any informative messages you return. Other major browsers and version 4 of Internet Explorer properly display server-generated error pages. See Figures 6–3 and 6–4 for an example.

Core Warning

By default, Internet Explorer version 5 ignores server-generated error pages.

405 (Method Not Allowed)

A 405 (`SC_METHOD_NOT_ALLOWED`) value indicates that the request method (`GET`, `POST`, `HEAD`, `PUT`, `DELETE`, etc.) was not allowed for this particular resource. This status code is new in HTTP 1.1.

406 (Not Acceptable)

A value of 406 (`SC_NOT_ACCEPTABLE`) signifies that the requested resource has a MIME type incompatible with the types specified by the client in its `Accept` header. 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. The 406 value is new in HTTP 1.1.

407 (Proxy Authentication Required)
The 407 (`SC_PROXY_AUTHENTICATION_REQUIRED`) value is similar to 401, but it is used by proxy servers. It indicates that the client must authenticate itself with the proxy server. The proxy server returns a `Proxy-Authenticate` response header to the client, which results in the browser reconnecting with a `Proxy-Authorization` request header. This status code is new in HTTP 1.1.

408 (Request Timeout)
The 408 (`SC_REQUEST_TIMEOUT`) code means that the client took too long to finish sending the request. It is new in HTTP 1.1.

409 (Conflict)
Usually associated with `PUT` requests, the 409 (`SC_CONFLICT`) status is used for situations such as an attempt to upload an incorrect version of a file. This status code is new in HTTP 1.1.

410 (Gone)
A value of 410 (`SC_GONE`) tells the client that the requested document is gone and no forwarding address is known. Status 410 differs from 404 in that the document is known to be permanently gone, not just unavailable for unknown reasons, as with 404. This status code is new in HTTP 1.1.

411 (Length Required)
A status of 411 (`SC_LENGTH_REQUIRED`) signifies that the server cannot process the request (assumedly a `POST` request with an attached document) unless the client sends a `Content-Length` header indicating the amount of data being sent to the server. This value is new in HTTP 1.1.

412 (Precondition Failed)
The 412 (`SC_PRECONDITION_FAILED`) status indicates that some precondition specified in the request headers was false. It is new in HTTP 1.1.

413 (Request Entity Too Large)
A status code of 413 (`SC_REQUEST_ENTITY_TOO_LARGE`) tells the client that the requested document is bigger than the server wants to handle

now. If the server thinks it can handle it later, it should include a `Retry-After` response header. This value is new in HTTP 1.1.

414 (Request URI Too Long)

The 414 (`SC_REQUEST_URI_TOO_LONG`) status is used when the URI is too long. In this context, "URI" means the part of the URL that came after the host and port in the URL. For example, in `http://www.y2k-disaster.com:8080/we/look/silly/now/`, the URI is `/we/look/silly/now/`. This status code is new in HTTP 1.1.

415 (Unsupported Media Type)

A value of 415 (`SC_UNSUPPORTED_MEDIA_TYPE`) means that the request had an attached document of a type the server doesn't know how to handle. This status code is new in HTTP 1.1.

416 (Requested Range Not Satisfiable)

A status code of 416 signifies that the client included an unsatisfiable `Range` header in the request. This value is new in HTTP 1.1. Surprisingly, the constant that corresponds to this value was omitted from `HttpServletResponse` in version 2.1 of the servlet API.

Core Note

In version 2.1 of the servlet specification, there is no `SC_REQUESTED_RANGE_NOT_SATISFIABLE` *constant in* `HttpServletResponse`*, so you have to use 416 explicitly. The constant is available in version 2.2 and later.*

417 (Expectation Failed)

If the server receives an `Expect` request header with a value of `100-continue`, it means that the client is asking if it can send an attached document in a follow-up request. In such a case, the server should either respond with this status (417) to tell the browser it won't accept the document or use 100 (`SC_CONTINUE`) to tell the client to go ahead. This status code is new in HTTP 1.1.

500 (Internal Server Error)

500 (`SC_INTERNAL_SERVER_ERROR`) is the generic "server is confused" status code. It often results from CGI programs or (heaven forbid!) servlets that crash or return improperly formatted headers.

501 (Not Implemented)
The 501 (`SC_NOT_IMPLEMENTED`) status notifies the client that the server doesn't support the functionality to fulfill the request. It is used, for example, when the client issues a command like `PUT` that the server doesn't support.

502 (Bad Gateway)
A value of 502 (`SC_BAD_GATEWAY`) is used by servers that act as proxies or gateways; it indicates that the initial server got a bad response from the remote server.

503 (Service Unavailable)
A status code of 503 (`SC_SERVICE_UNAVAILABLE`) signifies that the server cannot respond because of maintenance or overloading. For example, a servlet might return this header if some thread or database connection pool is currently full. The server can supply a `Retry-After` header to tell the client when to try again.

504 (Gateway Timeout)
A value of 504 (`SC_GATEWAY_TIMEOUT`) is used by servers that act as proxies or gateways; it indicates that the initial server didn't get a timely response from the remote server. This status code is new in HTTP 1.1.

505 (HTTP Version Not Supported)
The 505 (`SC_HTTP_VERSION_NOT_SUPPORTED`) code means that the server doesn't support the version of HTTP named in the request line. This status code is new in HTTP 1.1.

6.3 A Front End to Various Search Engines

Listing 6.1 presents an example that makes use of the two most common status codes other than 200 (OK): 302 (Found) and 404 (Not Found). The 302 code is set by the shorthand `sendRedirect` method of `HttpServletResponse`, and 404 is specified by `sendError`.

In this application, an HTML form (see Figure 6–1 and the source code in Listing 6.3) first displays a page that lets the user choose a search string, the number of results to show per page, and the search engine to use. When the

form is submitted, the servlet extracts those three parameters, constructs a URL with the parameters embedded in a way appropriate to the search engine selected (see the `SearchSpec` class of Listing 6.2), and redirects the user to that URL (see Figure 6–2). If the user fails to choose a search engine or specify search terms, an error page informs the client of this fact (see Figures 6–3 and 6–4).

Listing 6.1 `SearchEngines.java`

```
package coreservlets;

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

/** Servlet that takes a search string, number of results per
 *  page, and a search engine name, sending the query to
 *  that search engine. Illustrates manipulating
 *  the response status line. It sends a 302 response
 *  (via sendRedirect) if it gets a known search engine,
 *  and sends a 404 response (via sendError) otherwise.
 */

public class SearchEngines extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    String searchString = request.getParameter("searchString");
    if ((searchString == null) ||
        (searchString.length() == 0)) {
      reportProblem(response, "Missing search string.");
      return;

    }
    // The URLEncoder changes spaces to "+" signs and other
    // non-alphanumeric characters to "%XY", where XY is the
    // hex value of the ASCII (or ISO Latin-1) character.
    // Browsers always URL-encode form values, so the
    // getParameter method decodes automatically. But since
    // we're just passing this on to another server, we need to
    // re-encode it.
    searchString = URLEncoder.encode(searchString);
    String numResults =
      request.getParameter("numResults");
```

Listing 6.1 `SearchEngines.java` (continued)

```
    if ((numResults == null) ||
        (numResults.equals("0")) ||
        (numResults.length() == 0)) {
      numResults = "10";
    }
    String searchEngine =
      request.getParameter("searchEngine");
    if (searchEngine == null) {
      reportProblem(response, "Missing search engine name.");
      return;
    }
    SearchSpec[] commonSpecs = SearchSpec.getCommonSpecs();
    for(int i=0; i<commonSpecs.length; i++) {
      SearchSpec searchSpec = commonSpecs[i];
      if (searchSpec.getName().equals(searchEngine)) {
        String url =
          searchSpec.makeURL(searchString, numResults);
        response.sendRedirect(url);
        return;
      }
    }
    reportProblem(response, "Unrecognized search engine.");
  }

  private void reportProblem(HttpServletResponse response,
                             String message)
      throws IOException {
    response.sendError(response.SC_NOT_FOUND,
                       "<H2>" + message + "</H2>");
  }

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

Listing 6.2 `SearchSpec.java`

```
package coreservlets;

/** Small class that encapsulates how to construct a
 *  search string for a particular search engine.
 */

class SearchSpec {
  private String name, baseURL, numResultsSuffix;

  private static SearchSpec[] commonSpecs =
    { new SearchSpec("google",
                     "http://www.google.com/search?q=",
                     "&num="),
      new SearchSpec("infoseek",
                     "http://infoseek.go.com/Titles?qt=",
                     "&nh="),
      new SearchSpec("lycos",
                     "http://lycospro.lycos.com/cgi-bin/" +
                        "pursuit?query=",
                     "&maxhits="),
      new SearchSpec("hotbot",
                     "http://www.hotbot.com/?MT=",
                     "&DC=")
    };

  public SearchSpec(String name,
                    String baseURL,
                    String numResultsSuffix) {
    this.name = name;
    this.baseURL = baseURL;
    this.numResultsSuffix = numResultsSuffix;
  }

  public String makeURL(String searchString,
                        String numResults) {
    return(baseURL + searchString +
           numResultsSuffix + numResults);
  }

  public String getName() {
    return(name);
  }

  public static SearchSpec[] getCommonSpecs() {
    return(commonSpecs);
  }
}
```

Figure 6–1 Front end to the `SearchEngines` servlet. See Listing 6.3 for the HTML source code.

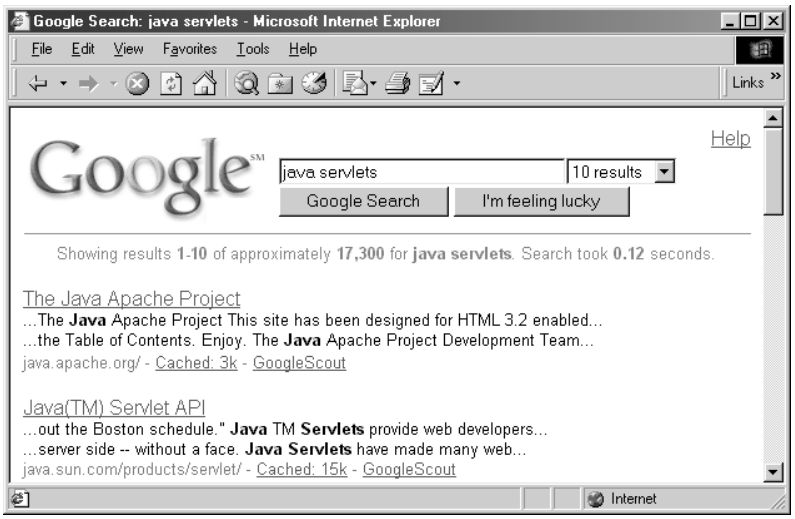

Figure 6–2 Result of the `SearchEngines` servlet when the form of Figure 6–1 is submitted.

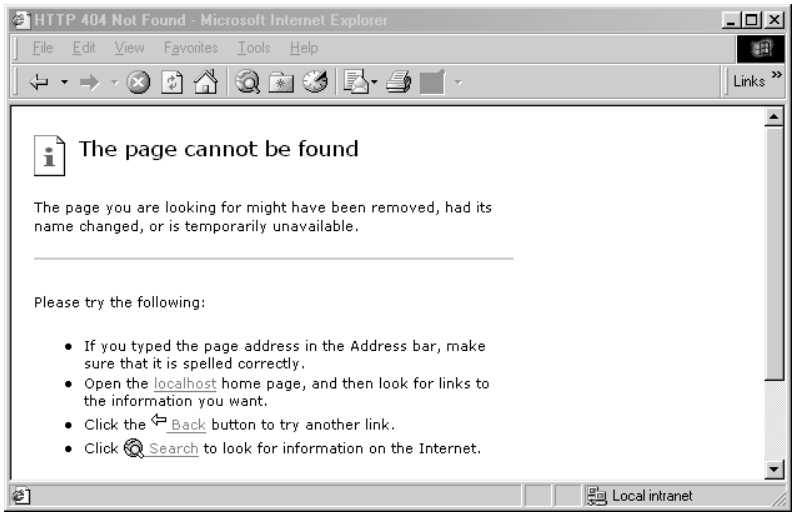

Figure 6–3 Result of `SearchEngines` servlet when no search string was specified. Internet Explorer 5 displays its own error page, even though the servlet generates one.

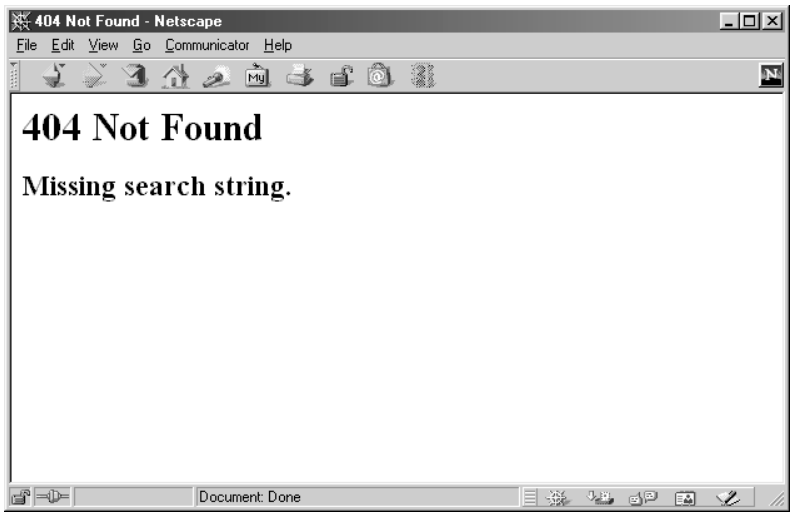

Figure 6–4 Result of `SearchEngines` servlet when no search string was specified. Netscape correctly displays the servlet-generated error page.

Listing 6.3 SearchEngines.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>Searching the Web</TITLE>
</HEAD>

<BODY BGCOLOR="#FDF5E6">
<H1 ALIGN="CENTER">Searching the Web</H1>

<FORM ACTION="/servlet/coreservlets.SearchEngines">
  <CENTER>
    Search String:
    <INPUT TYPE="TEXT" NAME="searchString"><BR>
    Results to Show Per Page:
    <INPUT TYPE="TEXT" NAME="numResults"
                       VALUE=10 SIZE=3><BR>
    <INPUT TYPE="RADIO" NAME="searchEngine"
                        VALUE="google">
    Google |
    <INPUT TYPE="RADIO" NAME="searchEngine"
                        VALUE="infoseek">
    Infoseek |
    <INPUT TYPE="RADIO" NAME="searchEngine"
                        VALUE="lycos">
    Lycos |
    <INPUT TYPE="RADIO" NAME="searchEngine"
                        VALUE="hotbot">
    HotBot
    <BR>
    <INPUT TYPE="SUBMIT" VALUE="Search">
  </CENTER>
</FORM>

</BODY>
</HTML>
```