5.3 Understanding HTTP 1.1 Request Headers

Access to the request headers permits servlets to perform a number of optimizations and to provide a number of features not otherwise possible. This section summarizes the headers most often used by servlets; for additional details on these and other headers, see the HTTP 1.1 specification, given in RFC 2616. The official RFCs are archived in a number of places; your best bet is to start at http://www.rfc-editor.org/ to get a current list of the archive sites. Note that HTTP 1.1 supports a superset of the headers permitted in HTTP 1.0.

Accept

This header specifies the MIME types that the browser or other clients can handle. A servlet that can return a resource in more than one format can examine the `Accept` header to decide which format to use. For example, images in PNG format have some compression advantages over those in GIF, but not all browsers support PNG. If you have images in both formats, your servlet can call `request.getHeader ("Accept")`, check for `image/png`, and if it finds a match, use `blah.png` filenames in all the `IMG` elements it generates. Otherwise, it would just use `blah.gif`.

See Table 7.1 in Section 7.2 (Understanding HTTP 1.1 Response Headers) for the names and meanings of the common MIME types.

Note that Internet Explorer 5 and 6 have a bug whereby the `Accept` header is sent improperly when you reload a page. It is sent properly in the original request, however.

Accept-Charset

This header indicates the character sets (e.g., ISO-8859-1) the browser can use.

Accept-Encoding

This header designates the types of encodings that the client knows how to handle. If the server receives this header, it is free to encode the page by using one of the formats specified (usually to reduce transmission time), sending the `Content-Encoding` response header to indicate that it has done so. This encoding type is completely distinct from the MIME type of the actual document (as specified in the `Content-Type` response header), since this encoding is reversed *before* the browser decides what to do with the content. On the other hand, using an encoding the browser doesn't understand results in incomprehensible pages. Consequently, it is critical that you explicitly check the `Accept-Encoding` header before using any type of content encoding. Values of `gzip` or `compress` are the two most common possibilities.

Compressing pages before returning them is a valuable service because the cost of decoding is likely to be small compared with the savings in transmission time. See Section 5.4 in which gzip compression is used to reduce download times by a factor of more than 10.

Accept-Language

This header specifies the client's preferred languages in case the servlet can produce results in more than one language. The value of the header should be one of the standard language codes such as `en`, `en-us`, `da`, etc. See RFC 1766 for details

(start at http://www.rfc-editor.org/ to get a current list of the RFC archive sites).

Authorization

This header is used by clients to identify themselves when accessing password-protected Web pages. For details, see the chapters on Web application security in Volume 2 of this book.

Connection

This header indicates whether the client can handle persistent HTTP connections. Persistent connections permit the client or other browser to retrieve multiple files (e.g., an HTML file and several associated images) with a single socket connection, thus saving the overhead of negotiating several independent connections. With an HTTP 1.1 request, persistent connections are the default, and the client must specify a value of `close` for this header to use old-style connections. In HTTP 1.0, a value of `Keep-Alive` means that persistent connections should be used.

Each HTTP request results in a new invocation of a servlet (i.e., a thread calling the servlet's `service` and `doXxx` methods), regardless of whether the request is a separate connection. That is, the server invokes the servlet only after the server has already read the HTTP request. This means that servlets need to cooperate with the server to handle persistent connections. Consequently, the servlet's job is just to make it *possible* for the server to use persistent connections; the servlet does so by setting the `Content-Length` response header. For details, see Chapter 7 (Generating the Server Response: HTTP Response Headers).

Content-Length

This header is applicable only to `POST` requests and gives the size of the `POST` data in bytes. Rather than calling `request.getIntHeader("Content-Length")`, you can simply use `request.getContentLength()`. However, since servlets take care of reading the form data for you (see Chapter 4), you rarely use this header explicitly.

Cookie

This header returns cookies to servers that previously sent them to the browser. Never read this header directly because doing so would require cumbersome low-level parsing; use `request.getCookies` instead. For details, see Chapter 8 (Handling Cookies). Technically, `Cookie` is not part of HTTP 1.1. It was originally a Netscape extension but is now widely supported, including in both Netscape and Internet Explorer.

Host

In HTTP 1.1, browsers and other clients are *required* to specify this header, which indicates the host and port as given in the original URL. Because of the widespread use of virtual hosting (one computer handling Web sites for multiple domain names), it is quite possible that the server could not otherwise determine this information. This header is not new in HTTP 1.1, but in HTTP 1.0 it was optional, not required.

If-Modified-Since

This header indicates that the client wants the page only if it has been changed after the specified date. The server sends a 304 (`Not Modified`) header if no newer result is available. This option is useful because it lets browsers cache documents

and reload them over the network only when they've changed. However, servlets don't need to deal directly with this header. Instead, they should just implement the `getLastModified` method to have the system handle modification dates automatically. For an example, see the lottery numbers servlet in Section 3.6 (The Servlet Life Cycle).

If-Unmodified-Since

This header is the reverse of `If-Modified-Since`; it specifies that the operation should succeed only if the document is older than the specified date. Typically, `If-Modified-Since` is used for `GET` requests ("give me the document only if it is newer than my cached version"), whereas `If-Unmodified-Since` is used for `PUT` requests ("update this document only if nobody else has changed it since I generated it"). This header is new in HTTP 1.1.

Referer

This header indicates the URL of the referring Web page. For example, if you are at Web page 1 and click on a link to Web page 2, the URL of Web page 1 is included in the `Referer` header when the browser requests Web page 2. Most major browsers set this header, so it is a useful way of tracking where requests come from. This capability is helpful for tracking advertisers who refer people to your site, for slightly changing content depending on the referring site, for identifying when users first enter your application, or simply for keeping track of where your traffic comes from. In the last case, most people rely on Web server log files, since the `Referer` is typically recorded there. Although the `Referer` header is useful, don't rely too heavily on it since it can easily be spoofed by a custom client. Also, note that, owing to a spelling mistake by one of the original HTTP authors, this header is `Referer`, not the expected `Referrer`.

Finally, note that some browsers (Opera), ad filters (Web Washer), and personal firewalls (Norton) screen out this header. Besides, even in normal situations, the header is only set when the user follows a link. So, be sure to follow the approach you should be using with all headers anyhow: check for `null` before using the header.

See Section 5.6 (Changing the Page According to How the User Got There) for details and an example.

User-Agent

This header identifies the browser or other client making the request and can be used to return different content to different types of browsers. Be wary of this use when dealing only with Web browsers; relying on a hard-coded list of browser versions and associated features can make for unreliable and hard-to-modify servlet code. Whenever possible, use something specific in the HTTP headers instead. For example, instead of trying to remember which browsers support gzip on which platforms, simply check the `Accept-Encoding` header.

However, the `User-Agent` header is quite useful for distinguishing among different *categories* of client. For example, Japanese developers might see whether the `User-Agent` is an Imode cell phone (in which case they would redirect to a chtml page), a Skynet cell phone (in which case they would redirect to a wml page), or a Web browser (in which case they would generate regular HTML).

Most Internet Explorer versions list a "Mozilla" (Netscape) version first in their `User-Agent` line, with the real browser version listed parenthetically. The Opera

browser does the same thing. This deliberate misidentification is done for compatibility with JavaScript; JavaScript developers often use the `User-Agent` header to determine which JavaScript features are supported. So, if you want to differentiate Netscape from Internet Explorer, you have to check for the string "MSIE" or something more specific, not just the string "Mozilla." Also note that this header can be easily spoofed, a fact that calls into question the reliability of sites that use this header to "show" market penetration of various browser versions.

See Section 5.5 (Differentiating Among Different Browser Types) for details and an example.