7.2 Understanding HTTP 1.1 Response Headers

Following is a summary of the most useful HTTP 1.1 response headers. A good understanding of these headers can increase the effectiveness of your servlets, so you should at least skim the descriptions to see what options are at your disposal. You can come back for details when you are ready to use the capabilities.

These headers are a superset of those permitted in HTTP 1.0. The official HTTP 1.1 specification is given in RFC 2616. The RFCs are online in various places; your best bet is to start at http://www.rfc-editor.org/ to get a current list of the archive sites. Header names are not case sensitive but are traditionally written with the first letter of each word capitalized.

Be cautious in writing servlets whose behavior depends on response headers that are available only in HTTP 1.1, especially if your servlet needs to run on the WWW "at large" rather than on an intranet—some older browsers support only HTTP 1.0. It is best to explicitly check the HTTP version with `request.getRequestProtocol` before using HTTP-1.1-specific headers.

Allow

The `Allow` header specifies the request methods (`GET`, `POST`, etc.) that the server supports. It is required for 405 (`Method Not Allowed`) responses. The default `service` method of servlets automatically generates this header for `OPTIONS` requests.

Cache-Control

This useful header tells the browser or other client the circumstances in which the response document can safely be cached. It has the following possible values.

- **`public`.** Document is cacheable, even if normal rules (e.g., for password-protected pages) indicate that it shouldn't be.
- **`private`.** Document is for a single user and can only be stored in private (nonshared) caches.
- **`no-cache`.** Document should never be cached (i.e., used to satisfy a later request). The server can also specify "`no-cache="header1,header2,...,headerN"`" to stipulate the headers that should be omitted if a cached response is later used. Browsers normally do not cache documents that were retrieved by requests that include form data. However, if a servlet generates different content for different requests even when the requests contain no form data, it is critical to tell the browser not to cache the response. Since older browsers use the `Pragma` header for this purpose, the typical servlet approach is to set *both* headers, as in the following example.

  ```
  response.setHeader("Cache-Control", "no-cache");
  response.setHeader("Pragma", "no-cache");
  ```

- **`no-store`.** Document should never be cached and should not even be stored in a temporary location on disk. This header is intended to prevent inadvertent copies of sensitive information.

- **`must-revalidate`.** Client must revalidate document with original server (not just intermediate proxies) each time it is used.

- **`proxy-revalidate`.** This is the same as `must-revalidate`, except that it applies only to shared caches.

- **`max-age=`*`xxx`*.** Document should be considered stale after *xxx* seconds. This is a convenient alternative to the `Expires` header but only works with HTTP 1.1 clients. If both `max-age` and `Expires` are present in the response, the `max-age` value takes precedence.

- **`s-max-age=`*`xxx`*.** Shared caches should consider the document stale after *xxx* seconds.

The `Cache-Control` header is new in HTTP 1.1.

Connection

A value of `close` for this response header instructs the browser not to use persistent HTTP connections. Technically, persistent connections are the default when the client supports HTTP 1.1 and does *not* specify a `Connection: close` request header (or when an HTTP 1.0 client specifies `Connection: keep-alive`). However, since persistent connections require a `Content-Length` response header, there is no reason for a servlet to explicitly use the `Connection` header. Just omit the `Content-Length` header if you aren't using persistent connections.

Content-Disposition

The `Content-Disposition` header lets you request that the browser ask the user to save the response to disk in a file of the given name. It is used as follows:

```
Content-Disposition: attachment; filename=some-file-name
```

This header is particularly useful when you send the client non-HTML responses (e.g., Excel spreadsheets as in Section 7.3 or JPEG images as in Section 7.5). `Content-Disposition` was not part of the original HTTP specification; it was defined later in RFC 2183. Recall that you can download RFCs by going to http://rfc-editor.org/ and following the instructions.

Content-Encoding

This header indicates the way in which the page was encoded during transmission. The browser should reverse the encoding before deciding what to do with the document. Compressing the document with gzip can result in huge savings in transmission time; for an example, see Section 5.4 (Sending Compressed Web Pages).

Content-Language

The `Content-Language` header signifies the language in which the document is written. 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 on language codes (you can access RFCs online at one of the archive sites listed at http://www.rfc-editor.org/).

Content-Length

This header indicates the number of bytes in the response. This information is needed only if the browser is using a persistent (keep-alive) HTTP connection. See the `Connection` header for determining when the browser supports persistent connections. If you want your servlet to take advantage of persistent connections when the browser supports them, your servlet should write the document into a `ByteArrayOutputStream`, look up its size when done, put that into the `Content-Length` field with `response.setContentLength`, then send the content by `byteArrayStream.writeTo(response.getOutputStream())`.

Content-Type

The `Content-Type` header gives the MIME (Multipurpose Internet Mail Extension) type of the response document. Setting this header is so common that there is a special method in `HttpServletResponse` for it: `setContentType`. MIME types are of the form *`maintype/subtype`* for officially registered types and of the form *`maintype/x-subtype`* for unregistered types. Most servlets specify `text/html`; they can, however, specify other types instead. This is important partly because servlets directly generate other MIME types (as in the Excel and JPEG examples of this chapter), but also partly because servlets are used as the glue to connect other applications to the Web. OK, so you have Adobe Acrobat to generate PDF, GhostScript to generate PostScript, and a database application to search indexed MP3 files. But you still need a servlet to answer the HTTP request, invoke the helper application, and set the `Content-Type` header, even though the servlet probably simply passes the output of the helper application directly to the client.

In addition to a basic MIME type, the `Content-Type` header can also designate a specific character encoding. If this is not specified, the default is `ISO-8859_1` (Latin). For example, the following instructs the browser to interpret the document as HTML in the `Shift_JIS` (standard Japanese) character set.

```
response.setContentType("text/html; charset=Shift_JIS");
```

Table 7.1 lists some of the most common MIME types used by servlets. RFC 1521 and RFC 1522 list more of the common MIME types (again, see http://www.rfc-editor.org/ for a list of RFC archive sites). However, new MIME types are registered all the time, so a dynamic list is a better place to look. The officially registered types are listed at http://www.isi.edu/in-notes/iana/assignments/media-types/media-types. For common unregistered types, http://www.ltsw.se/knbase/internet/mime.htp is a good source.

Table 7.1. Common MIME Types

Type	Meaning
application/msword	Microsoft Word document
application/octet-stream	Unrecognized or binary data
application/pdf	Acrobat (.pdf) file
application/postscript	PostScript file
application/vnd.lotus-notes	Lotus Notes file
application/vnd.ms-excel	Excel spreadsheet
application/vnd.ms-powerpoint	PowerPoint presentation
application/x-gzip	Gzip archive
application/x-java-archive	JAR file

application/x-java-serialized-object	Serialized Java object
application/x-java-vm	Java bytecode (.class) file
application/zip	Zip archive
audio/basic	Sound file in .au or .snd format
audio/midi	MIDI sound file
audio/x-aiff	AIFF sound file
audio/x-wav	Microsoft Windows sound file
image/gif	GIF image
image/jpeg	JPEG image
image/png	PNG image
image/tiff	TIFF image
image/x-xbitmap	X Windows bitmap image
text/css	HTML cascading style sheet
text/html	HTML document
text/plain	Plain text
text/xml	XML
video/mpeg	MPEG video clip
video/quicktime	QuickTime video clip

Expires

This header stipulates the time at which the content should be considered out-of-date and thus no longer be cached. A servlet might use this header for a document that changes relatively frequently, to prevent the browser from displaying a stale cached value. Furthermore, since some older browsers support `Pragma` unreliably (and `Cache-Control` not at all), an `Expires` header with a date in the past is often used to prevent browser caching. However, some browsers ignore dates before January 1, 1980, so do not use 0 as the value of the `Expires` header.

For example, the following would instruct the browser not to cache the document for more than 10 minutes.

```
long currentTime = System.currentTimeMillis();
long tenMinutes = 10*60*1000; // In milliseconds
response.setDateHeader("Expires",
                       currentTime + tenMinutes);
```

Also see the `max-age` value of the `Cache-Control` header.

Last-Modified

This very useful header indicates when the document was last changed. The client can then cache the document and supply a date by an `If-Modified-Since` request header in later requests. This request is treated as a conditional `GET`, with the document being returned only if the `Last-Modified` date is later than the one specified for `If-Modified-Since`. Otherwise, a 304 (`Not Modified`) status line is returned, and the client uses the cached document. If you set this header explicitly, use the

`setDateHeader` method to save yourself the bother of formatting GMT date strings. However, in most cases you simply implement the `getLastModified` method (see the lottery number servlet of Section 3.6, "The Servlet Life Cycle") and let the standard `service` method handle `If-Modified-Since` requests.

Location

This header, which should be included with all responses that have a status code in the 300s, notifies the browser of the document address. The browser automatically reconnects to this location and retrieves the new document. This header is usually set indirectly, along with a 302 status code, by the `sendRedirect` method of `HttpServletResponse`. See Sections 6.3 (A Servlet That Redirects Users to Browser-Specific Pages) and 6.4 (A Front End to Various Search Engines) for examples.

Pragma

Supplying this header with a value of `no-cache` instructs HTTP 1.0 clients not to cache the document. However, support for this header was inconsistent with HTTP 1.0 browsers, so `Expires` with a date in the past is often used instead. In HTTP 1.1, `Cache-Control: no-cache` is a more reliable replacement.

Refresh

This header indicates how soon (in seconds) the browser should ask for an updated page. For example, to tell the browser to ask for a new copy in 30 seconds, you would specify a value of 30 with

```
response.setIntHeader("Refresh", 30);
```

Note that `Refresh` does not stipulate continual updates; it just specifies when the *next* update should be. So, you have to continue to supply `Refresh` in all subsequent responses. This header is extremely useful because it lets servlets return partial results quickly while still letting the client see the complete results at a later time. For an example, see Section 7.4 (Persistent Servlet State and Auto-Reloading Pages).

Instead of having the browser just reload the current page, you can specify the page to load. You do this by supplying a semicolon and a URL after the refresh time. For example, to tell the browser to go to `http://host/path` after 5 seconds, you would do the following.

```
response.setHeader("Refresh", "5; URL=http://host/path/");
```

This setting is useful for "splash screens" on which an introductory image or message is displayed briefly before the real page is loaded.

Note that this header is commonly set indirectly by putting

```
<META HTTP-EQUIV="Refresh"
      CONTENT="5; URL=http://host/path/">
```

in the `HEAD` section of the HTML page, rather than as an explicit header from the server. That usage came about because automatic reloading or forwarding is something often desired by authors of static HTML pages. For servlets, however, setting the header directly is easier and clearer.

This header is not officially part of HTTP 1.1 but is an extension supported by both Netscape and Internet Explorer.

Retry-After

This header can be used in conjunction with a 503 (`Service Unavailable`) response to tell the client how soon it can repeat its request.

Set-Cookie

The `Set-Cookie` header specifies a cookie associated with the page. Each cookie requires a separate `Set-Cookie` header. Servlets should not use `response.setHeader("Set-Cookie", ...)` but instead should use the special-purpose `addCookie` method of `HttpServletResponse`. For details, see Chapter 8 (Handling Cookies). Technically, `Set-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.

WWW-Authenticate

This header is always included with a 401 (`Unauthorized`) status code. It tells the browser what authorization type (`BASIC` or `DIGEST`) and realm the client should supply in its `Authorization` header. For examples of the use of `WWW-Authenticate` and a discussion of the various security mechanisms available to servlets and JSP pages, see the chapters on Web application security in Volume 2 of this book.