

TRAINING & REFERENCE

murach's

Java
servlets and
JSP

(Chapter 15)

Andrea Steelman
Joel Murach

Mike Murach & Associates



2560 West Shaw Lane, Suite 101
Fresno, CA 93711-2765
(559) 440-9071 · (800) 221-5528

murachbooks@murach.com · www.murach.com

Copyright © 2003 Mike Murach & Associates. All rights reserved.

Section 4

Advanced servlet and JSP skills

This section contains six chapters that present other servlet and JSP skills that you may need for some of your web applications. Chapter 12 shows you how to send email from a servlet. Chapters 13 and 14 show you how to use a secure connection for sensitive data and how to restrict access to portions of a web application. Chapter 15 tells you more about working with the request and response objects of HTTP. Chapter 16 shows you how to use XML in your web applications. And chapter 17 provides a brief introduction to Enterprise JavaBeans.

Because each of these chapters is written as an independent module, you can read these chapters in whatever sequence you prefer. If, for example, you want to learn more about Enterprise JavaBeans, you can skip to chapter 17. Or, if you want to learn how to work with a secure connection, you can skip to chapter 13. Eventually, though, you should read all of the chapters in this section because they all provide useful capabilities.

How to work with HTTP requests and responses

When you write servlets and JSPs, the classes and methods of the servlet API shelter you from having to work directly with HTTP. Sometimes, though, you need to know more about HTTP requests and responses, and you need to use the methods of the servlet API to work with them. So that's what you'll learn in this chapter. Along the way, you'll get a better idea of how HTTP works.

An introduction to HTTP	438
An HTTP request and response	438
Common MIME types	440
Common HTTP request headers	442
Common HTTP status codes	444
Common HTTP response headers	446
How to work with the request	448
How to get request headers	448
How to display all request headers	450
The request headers for the IE and Netscape browsers	452
How to work with the response	454
How to set status codes	454
How to set response headers	454
Practical skills for working with HTTP	456
How to return a tab-delimited file as an Excel spreadsheet	456
How to control caching	456
How to encode the response with GZIP compression	458
How to require the File Download dialog box	460
Perspective	462

An introduction to HTTP

This topic introduces you to some of the most common headers and status codes that make up *Hypertext Transfer Protocol*, or *HTTP*. This protocol can be used to request a resource from a server, and it can be used to return a response from a server.

An HTTP request and response

Figure 15-1 shows the components of a typical HTTP request and a typical HTTP response. As you learn more about these components, you'll get a better idea of how HTTP requests and responses work.

The first line of an HTTP request is known as the *request line*. This line contains the request method, the request URL, and the request protocol. Typically, the request method is a Get or Post method, but other methods are also supported by HTTP. Similarly, the request protocol is usually HTTP 1.1, but some older browsers use HTTP 1.0.

After the request line, an HTTP request contains the *request headers*. These headers contain information about the client that's making the request. In this figure, the HTTP request contains eight request headers with one header per line, but a request can include more headers than that. Each request header begins with the name of the request header, followed by a colon and a space, followed by the value of the request header.

After the request headers, an HTTP request that uses the Post method may include a blank line followed by the parameters for the request. Unlike a Get request, a Post request doesn't include its parameters in the URL.

The first line of an HTTP response is known as the *status line*. This line specifies the version of HTTP that's being used, a *status code*, and a message that's associated with the status code.

After the status line, an HTTP response contains the *response headers*. These headers contain information about the server and about the response that's being returned to the client. Like request headers, each response header takes one line. In addition, each line begins with the name of the header, followed by a colon and a space, followed by the value of the header.

After the response headers, an HTTP response contains a blank line, followed by the *response entity*, or *response body*. In this figure, the response entity is an HTML document, but it could also be plain text, tab-delimited text, and so on.

To learn more about HTTP 1.1, you can use Adobe Acrobat to open the PDF file that's stored in the HTTPSpec directory of the CD that comes with this book. This PDF file contains a highly technical description of HTTP 1.1 including a list of headers and status codes. Or, you can view this document by going to www.rfc-editor.org and searching for 2616. Then, you can follow the links to other related HTTP documents, which may contain updated information.

An HTTP request

```
GET http://www.murach.com/email/join_email_list.html HTTP/1.1
referer: http://www.murach.com/murach/index.html
connection: Keep-Alive
user-agent: Mozilla/4.61 [en] (Win98; I)
host: www.murach.com
accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
accept-encoding: gzip
accept-language: en
cookie: emailCookie=jsmith%40hotmail.com; userID=39210
```

An HTTP response

```
HTTP/1.1 200 OK
date: Sat, 17 Aug 2002 10:32:54 GMT
server: Apache/1.3.6 (Unix) PHP/3.0.7
content-type: text/html
content-length: 201
last-modified: Fri, 16 Aug 2002 12:52:09 GMT

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
  <title>Chapter 4 - Email List application</title>
</head>
<body>
  <h1>Join our email list</h1>
</body>
</html>
```

Description

- *Hypertext Transfer Protocol*, or *HTTP*, is the primary protocol that's used to transfer data between a browser and a server. Two versions of HTTP exist: 1.0 and 1.1. Since HTTP 1.1 is a superset of the HTTP 1.0, all HTTP 1.0 request headers are also available in HTTP 1.1.
- The first line of an HTTP request is known as the *request line*. This line specifies the request method, the URL of the request, and the version of HTTP.
- After the first line of a request, the browser sends *request headers* that give information about the browser and its request.
- The first line of an HTTP response is known as the *status line*. This line specifies the HTTP version, a *status code*, and a brief description associated with the status code.
- After the first line of a response, the server sends *response headers* that give information about the response. Then, it sends the *response entity*, or *response body*. The body of a response is typically HTML, but it can also be other types of data.
- To view a technical description of the HTTP 1.1 specification, you can use Adobe Acrobat to open the PDF file that's stored in the HTTPSpec directory of the CD that comes with this book. This PDF file includes a list of headers and status codes. Or, you can view this document by going to www.rfc-editor.org and searching for 2616.

Figure 15-1 An HTTP request and response

Common MIME types

Figure 15-2 shows some of the most common *Multipurpose Internet Mail Extension*, or *MIME*, types that are used by HTTP. You can use them in the accept header of a request or the content-type header of a response.

To specify an officially registered MIME type, you can use this format:

type/subtype

To specify a MIME type that isn't officially registered, you can use this format:

type/x-subtype

Although the "text/plain" MIME type is the default MIME type for a servlet, the most commonly used MIME type is the "text/html" type. Later in this chapter, you'll see examples of how MIME types can be used in HTTP requests and responses.

If you want to learn more about MIME types, you can use Adobe Acrobat to open the PDF file that's stored in the MIMESpec directory on the CD that comes with this book. This PDF file contains a highly technical description of MIME types. Or, you can go to www.rfc-editor.org and search for 1341. Then, you can follow the links to newer MIME types.

Common MIME types

Type/Subtype	Description
text/plain	Plain text document
text/html	HTML document
text/css	HTML cascading style sheet
text/xml	XML document
image/gif	GIF image
image/jpeg	JPEG image
image/png	PNG image
image/tiff	TIFF image
image/x-xbitmap	Window bitmap image
application/msword	Microsoft Word document
application/vnd.ms-excel	Microsoft Excel spreadsheet
application/pdf	Adobe Acrobat file
application/postscript	PostScript file
application/zip	Zip file
application/x-java-archive	Jar file
application/x-gzip	Gzip file
application/octet-stream	Binary data
audio/basic	A sound file (usually in the *.au or *.snd format)
video/mpeg	MPEG video clip

Two web sites that have lists of other MIME types

www.isi.edu/in-notes/iana/assignments/media-types/media-types
www.ltsw.se/knbase/internet/mime.htm

Description

- The *Multipurpose Internet Mail Extension* types, or *MIME* types, provide standards for the various types of data that can be transferred across the Internet.
- MIME types can be included in the accept header of a request or the content-type header of a response.
- For more technical information about MIME types, you can check www.rfc-editor.org and search for RFC 1341.

Common HTTP request headers

Figure 15-3 lists some of the most common HTTP request headers. When you work with HTTP, you should be aware that some older browsers don't support HTTP 1.1. So if your web application needs to support these older browsers, you should only use headers that were specified in the HTTP 1.0 version. Today, however, most web browsers support HTTP 1.1 so you can usually use HTTP 1.1 headers.

Most of the time, a web browser automatically sets these request headers when it makes a request. Then, when the server receives the request, it can check these headers to learn about the browser. In addition, though, you can write servlets that set some of these request headers. For example, chapter 7 shows how to use the servlet API to set the cookie header. And chapter 14 shows how to use the servlet container to automatically set the authorization header.

Common HTTP request headers

Name	Description
accept	Specifies the preferred order of MIME types that the browser can accept. The “*/*” type indicates that the browser can handle any MIME type.
accept-encoding	Specifies the types of compression encoding that the browser can accept.
accept-charset	Specifies the character sets that the browser can accept. Although the Internet Explorer doesn’t usually return this header, Netscape usually does.
accept-language	Specifies the standard language codes for the languages that the browser prefers. The standard language code for English is “en” or “en-us”.
authorization	Identifies the authorization level for the browser. When you use container-managed security as described in chapter 14, the servlet container automatically sets this header.
connection	Indicates the type of connection that’s being used by the browser. In HTTP 1.0, a value of “keep-alive” means that the browser can use a persistent connection that allows it to accept multiple files with a single connection. In HTTP 1.1, this type of connection is the default.
cookie	Specifies any cookies that were previously sent by the current server. In chapter 7, you learned how to use the servlet API to work with this header.
host	Specifies the host and port of the machine that originally sent the request. This header is optional in HTTP 1.0 and required in HTTP 1.1.
pragma	A value of “no-cache” indicates that any servlet that’s forwarding requests shouldn’t cache this page.
referer	Indicates the URL of the referring web page. The spelling error was made by one of the original authors of HTTP and is now part of the protocol.
user-agent	Indicates the type of browser. Although both Internet Explorer and Netscape identify themselves as “Mozilla”, the Internet Explorer always includes “MSIE” somewhere in the string.

Figure 15-3 HTTP request headers

Common HTTP status codes

Figure 15-4 summarizes the five categories of status codes. Then, this figure lists some of the most common status codes. For successful requests, the server typically returns a 200 (OK) status code. However, if the server can't find the requested file, it typically returns the infamous 404 (Not Found) status code. Or, if the server encounters an error while trying to retrieve the file, it may return the equally infamous 500 (Internal Server Error) status code.

Status code summary

Number	Type	Description
100-199	Informational	The request was received and is being processed.
200-299	Success	The request was successful.
300-399	Redirection	Further action must be taken to fulfill the request.
400-499	Client errors	The client has made a request that contains an error.
500-599	Server errors	The server has encountered an error.

Status codes

Number	Name	Description
200	OK	The default status indicating that the response is normal.
301	Moved Permanently	The requested resource has been permanently moved to a new URL.
302	Found	The requested resource resides temporarily under a new URL.
400	Bad Request	The request could not be understood by the server due to bad syntax.
401	Unauthorized	The request requires authentication. The response must include a <code>www-authenticate</code> header. If you use container-managed security as described in chapter 14, the web server automatically returns this status code when appropriate.
404	Not Found	The server could not find the requested URL.
405	Method Not Allowed	The method specified in the request line is not allowed for the requested URL.
500	Internal Server Error	The server encountered an unexpected condition that prevented it from fulfilling the request.

Figure 15-4 HTTP status codes

Common HTTP response headers

Figure 15-5 lists some of the most common HTTP response headers. Most of the time, the web server automatically sets these response headers when it returns the response. However, there are times when you may want to use Java code to set response headers that control the response sent by your web server.

For example, you can use the cache-control header to control how your web server caches a response. To do that, you can use the cache-control values specified in this figure to turn off caching, to use a private cache, to use a public cache, to specify when a response must be revalidated, or to increase the duration of the cache. You'll see an example of this later on in this chapter.

Common HTTP response headers

Name	Description
cache-control	Controls when and how a browser caches a page. For more information, see figure 15-10 and the list of possible values shown below.
content-disposition	Can be used to specify that the response includes an attached binary file. For an example, see figure 15-12.
content-length	Specifies the length of the body of the response in bytes. This allows the browser to know when it's done reading the entire response and is necessary for the browser to use a persistent, keep-alive connection.
content-type	Specifies the MIME type of the response document. You can use the "maintype/subtype" format shown earlier in this chapter to specify the MIME type.
content-encoding	Specifies the type of encoding that the response uses. Encoding a document with GZIP can enhance performance. For an example, see figure 15-11.
expires	Specifies the time that the page should no longer be cached.
last-modified	Specifies the time when the document was last modified.
location	Works with status codes in the 300s to specify the new location of the document.
pragma	Turns off caching for older browsers when it is set to a value of "no-cache".
refresh	Specifies the number of seconds before the browser should ask for an updated page.
www-authenticate	Works with the 401 (Unauthorized) status code to specify the authentication type and realm. If you use container-managed security as described in chapter 14, the servlet container automatically sets this header when necessary.

Values for the cache-control header

Name	Description
public	The document can be cached in a public, shared cache.
private	The document can only be cached in a private, single-user cache.
no-cache	The document should never be cached.
no-store	The document should never be cached or stored in a temporary location on the disk.
must-revalidate	The document must be revalidated with the original server (not a proxy server) each time it is requested.
proxy-revalidate	The document must be revalidated on the proxy server but not on the original server.
max-age=x	The document must be revalidated after x seconds for private caches.
s-max-age=x	The document must be revalidated after x seconds for shared caches.

Figure 15-5 HTTP response headers

How to work with the request

This topic shows how to use the methods of the request object to get the data that's contained in an HTTP request.

How to get request headers

You can use the first group of methods in figure 15-6 to get any of the headers in an HTTP request. The `getHeader` method lets you return the value of any header. The `getIntHeader` and `getDateHeader` methods make it easier to work with headers that contain integer and date values. And the `getHeaderNames` method returns an Enumeration object that contains the names of all of the headers for the request.

You can use the second group of methods to get the request headers more easily. For example, this statement:

```
int contentLength = request.getIntHeader("Content-Length");
```

returns the same value as this statement:

```
int contentLength = request.getContentLength();
```

The first example uses the `getHeader` method to return a string that includes all of the MIME types supported by the browser that made the request. Then, it uses the `indexOf` method within an if statement to check if a particular MIME type exists in the string. If it does, the code calls a method that returns a PNG image if the browser supports that type. Otherwise, the code calls a method that returns a GIF image.

The second example uses the `getHeader` method to return a string that identifies the type of browser that made the request. Since the Internet Explorer always includes the letters "MSIE" in its string, this example uses an `indexOf` method within an if statement to check if the string contains the letters "MSIE". If so, it calls a method that executes some code that's specific to the Internet Explorer. Otherwise, it calls a method that executes some code that's specific to the Netscape browser.

Although you may never need to write code that checks the MIME types or the browser type, these examples illustrate a general concept that you can use to check any request header. First, you use the `getHeader`, `getIntHeader`, or `getDateHeader` methods to return a header. Then, you can use an if statement to check the header. For a String object, you can use the `indexOf` method to check if a substring exists within the string. For int values and Date objects, you can use other comparison operators.

General methods for working with request headers

```
String getHeader(String headerName)
int getIntHeader(String headerName)
Date getDateHeader(String headerName)
Enumeration getHeaderNames()
```

Convenience methods for working with request headers

```
String getContentType()
int getContentLength()
Cookie[] getCookies()
String getAuthType()
String getRemoteUser()
```

An example that checks the MIME types accepted by the browser

```
String mimeType = request.getHeader("Accept");
if (mimeType.indexOf("image/png") > -1)
    returnPNG();
else
    returnGIF();
```

An example that checks the browser type

```
String browser = request.getHeader("User-Agent");
if (browser.indexOf("MSIE") > -1)
    doIECode();
else
    doNetscapeCode();
```

Description

- All of these methods can be called from the request object to return information about the HTTP request.
- The general header methods provide a generic way to access any HTTP header.
- The convenience header methods provide a specialized way to access commonly used HTTP headers.
- For more information about using the `getCookies` method to work with the cookie header, see chapter 7.
- For more information about using the servlet container to automatically check the authorization header, see chapter 14.
- For more information about any of these methods, you can look up the `HttpServletRequest` interface in the documentation for the servlet API.

How to display all request headers

Figure 15-7 shows a JSP that displays all of the request headers for a request. If you're developing a web application that needs to check other request headers, you can use this JSP to quickly view the request headers for all of the different browsers that your web application supports. Then, you can write the code that checks the request headers and works with them.

This JSP begins by defining the table that will contain the request headers. In the first row of the table, the first column is the Name column, and the second column is the Value column.

After the first row of the table, this page imports the `java.util` package, so the page can work with the Enumeration class. Then, it uses the `getHeaderNames` method of the request object to return an Enumeration object that contains the names of all of the request headers. After that, a while loop cycles through all of the header names. To do that, this loop uses the `hasMoreElements` method to check if more elements exist, and it uses the `nextElement` method to return the current header name and move to the next header name. Once the loop has returned the name of the header, the `getHeader` method of the request object uses this name to return the value of the header. Last, the loop displays the name and value for the current header in a row.

The JSP code that displays all request headers

```
<!doctype html public "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
  <title>Chapter 14 - HTTP requests and responses</title>
</head>
<body>

<h1>Request Headers</h1>

<table cellpadding="5" border="1">

  <tr>
    <td align="right"><b>Name</td>
    <td><b>Value</td>
  </tr>
  <%= page import="java.util.*" %>
  <%
    Enumeration headerNames = request.getHeaderNames();
    while (headerNames.hasMoreElements()){
      String name = (String) headerNames.nextElement();
      String value = request.getHeader(name);
    %>
    <tr>
      <td align="right"><%= name %></td>
      <td><%= value %></td>
    </tr>
  <% } %>

</table>

</body>
</html>
```

Description

- This JSP displays all the request headers for a request.
- In the next figure, you can see how this JSP is rendered for two different browsers.

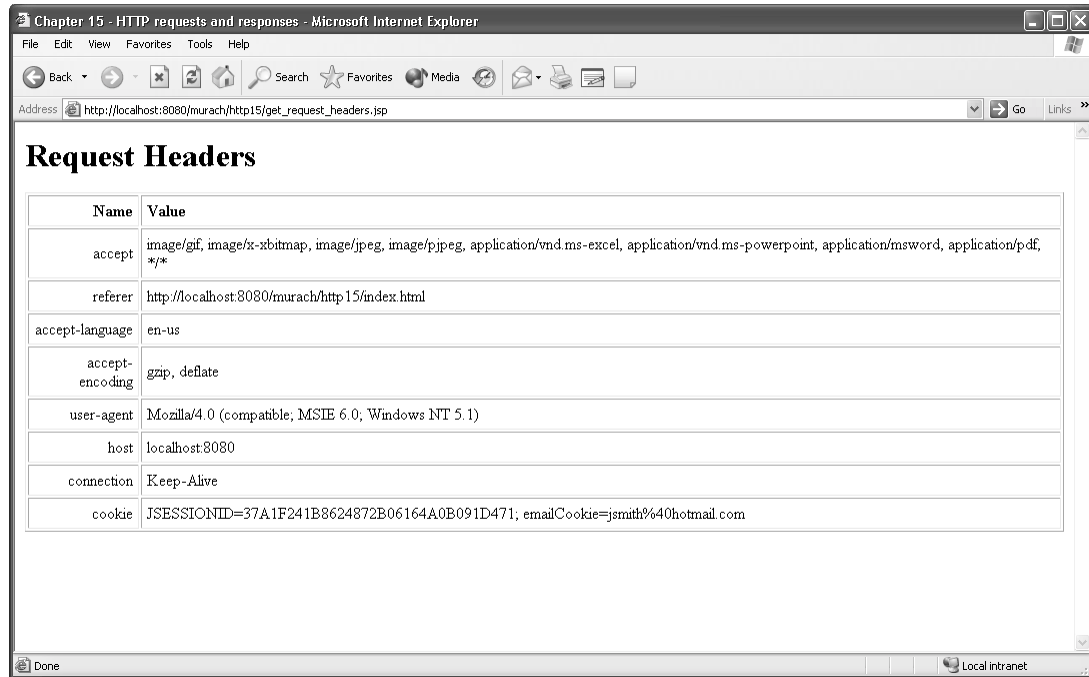
The request headers for the IE and Netscape browsers

Since the request headers and their values vary depending on the client that makes the request, figure 15-8 shows how the JSP in the last figure looks when displayed by different clients. First, it shows how the JSP will look when requested by version 6.0 of the Internet Explorer. Then, it shows how the JSP will look when requested by version 7.0 of the Netscape browser. If you compare the values sent by each of these browsers, you'll see some of the differences between them.

For example, the accept header for IE6 indicates that it prefers documents in Microsoft Word or Microsoft Excel formats. Of course, this is what you would expect from a Microsoft product. Since the Netscape browser supports the `*/*` type, it also supports these formats. However, by not specifying these formats in its accept header, the Netscape browser indicates that it doesn't prefer the Microsoft formats.

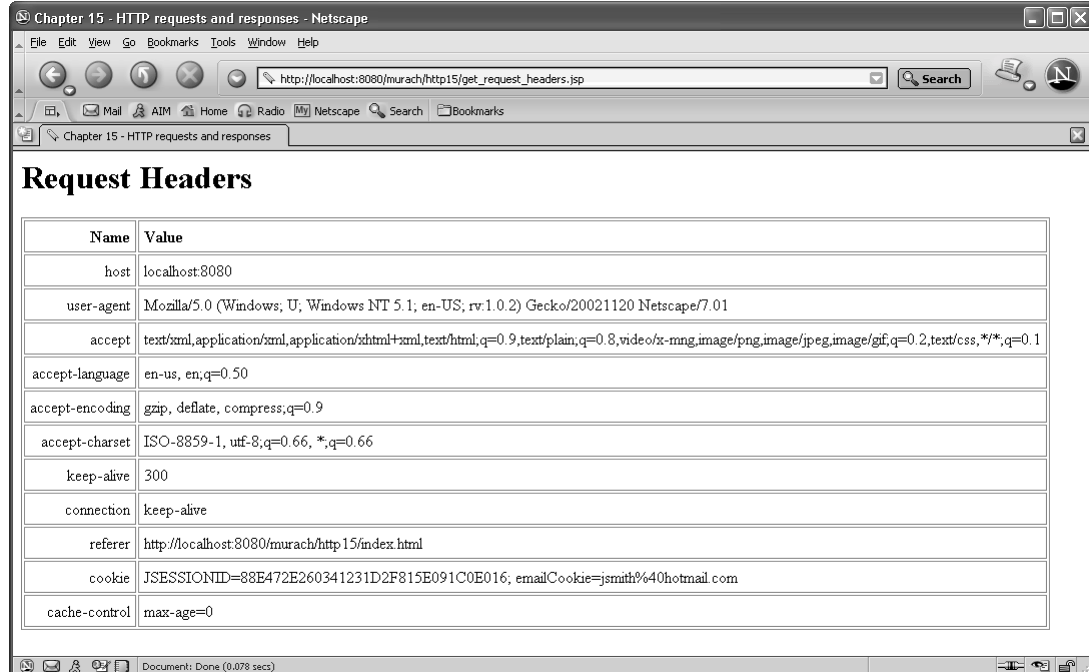
If you compare the user-agent headers, you'll see that the Netscape browser identifies itself as a "Mozilla" browser while the IE browser identifies itself as being compatible with "Mozilla". However, the IE browser includes the "MSIE" string that indicates that it's the Microsoft Internet Explorer. In addition, if you check the accept-charset header, you'll see that the Netscape browser sends this header while the IE browser doesn't.

All request headers sent by Internet Explorer 6.0



Name	Value
accept	image/gif, image/x-bitmap, image/jpeg, image/pjpeg, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, application/pdf, */*
referer	http://localhost:8080/murach/http15/index.html
accept-language	en-us
accept-encoding	gzip, deflate
user-agent	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
host	localhost:8080
connection	Keep-Alive
cookie	JSESSIONID=37A1F241B8624872B06164A0B091D471; emailCookie=jsmith%40hotmail.com

All request headers sent by Netscape 7.0



Name	Value
host	localhost:8080
user-agent	Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.0.2) Gecko/20021120 Netscape/7.01
accept	text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,video/x-mng,image/png,image/jpeg,image/gif;q=0.2,text/css;*/q=0.1
accept-language	en-us, en;q=0.50
accept-encoding	gzip, deflate, compress;q=0.9
accept-charset	ISO-8859-1, utf-8;q=0.66, */q=0.66
keep-alive	300
connection	keep-alive
referer	http://localhost:8080/murach/http15/index.html
cookie	JSESSIONID=88E472E260341231D2F815E091C0E016; emailCookie=jsmith%40hotmail.com
cache-control	max-age=0

Figure 15-8 Typical request headers for the IE and Netscape browsers

How to work with the response

Figure 15-9 shows how to use the fields and methods of the response object to set the data that's contained in an HTTP response.

How to set status codes

Most of the time, the web server automatically sets the status code for an HTTP response. However, if you need to set the status code, you can use the `setStatus` method. To specify the value for this code, you can use either an integer value or one of the fields of the response object. For example, this figure shows two ways to specify the 404 (Not Found) status code.

How to set response headers

Like status codes, the web server usually sets the headers of an HTTP response. However, if you need to set a response header, this figure shows six methods that you can use to set response headers. To start, you can use the `setHeader`, `setIntHeader`, and `setDateHeader` methods to set all response headers that accept strings, integers, or dates. Here, the `setDateHeader` accepts a long value that represents the date in milliseconds since January 1, 1970 00:00:00 GMT.

On the other hand, if you're working with commonly used headers, such as the content-type or content-length headers, you can use the `setContentType` and `setContentLength` methods. And you can use the `addCookie` method to add a value to the cookie header as described in chapter 7.

The examples show how to work with response headers. The first statement uses the `setContentType` method to set the value of the content-type header to the "text/html" MIME type. The second statement uses the `setContentLength` method to set the content-length header to 403 bytes, although you usually won't need to set this header.

The third statement uses the `setHeader` method to set the pragma header to "no-cache" to turn off caching for older browsers. The fourth statement uses the `setIntHeader` method to set the refresh header to 60 seconds. As a result, the browser will request an updated page in 1 minute. Last, the fifth statement uses the `setDateHeader` to set the expires header so caching for the page will expire after 1 hour. To do that, this statement calls the `getTime` method from a `Date` object named `currentDate` to return the current time in milliseconds. Then, it adds 3,600,000 milliseconds to that date (1000 milliseconds times 60 seconds times 60 minutes equals one hour).

The main method for setting the status codes

```
void setStatus(int code)
```

How status codes map to fields of the response object

Code	HttpServletResponse field
200 (OK)	SC_OK
404 (Not Found)	SC_NOT_FOUND
XXX (Xxx Xxx)	SC_XXX_XXX

Examples that set the status code

```
response.setStatus(404);
response.setStatus(response.SC_NOT_FOUND);
```

General methods for setting response headers

```
void setHeader(String headerName, String headerValue)
void setIntHeader(String headerName, int headerValue)
void setDateHeader(String headerName, long headerValue)
```

Convenience methods for setting response headers

```
void setContentType(String mimeType)
void setContentLength(int lengthInBytes)
void addCookie(Cookie cookie)
```

Examples that set response headers

```
response.setContentType("text/html");
response.setContentLength(403);
response.setHeader("pragma", "no-cache");
response.setIntHeader("refresh", 60);
response.setDateHeader("expires", currentDate.getTime() + 60 * 60 * 1000);
```

Description

- For more information about these methods and fields, you can look up the HttpServletResponse interface in the documentation for the servlet API.

Practical skills for working with HTTP

Now that you understand how to use the servlet API to work with HTTP requests and responses, you're ready to learn some practical skills for working with HTTP. These will illustrate when you might want to work directly with the requests and responses.

How to return a tab-delimited file as an Excel spreadsheet

Most modern browsers can use Microsoft Excel to read tab-delimited text. To display tab-delimited text as a spreadsheet, then, you can create some tab-delimited text, set the Content-Type response header to the "application/vnd.ms-excel" MIME type, and return the tab-delimited text to the browser. Then, the browser will open the tab-delimited text in Excel as shown in figure 15-10.

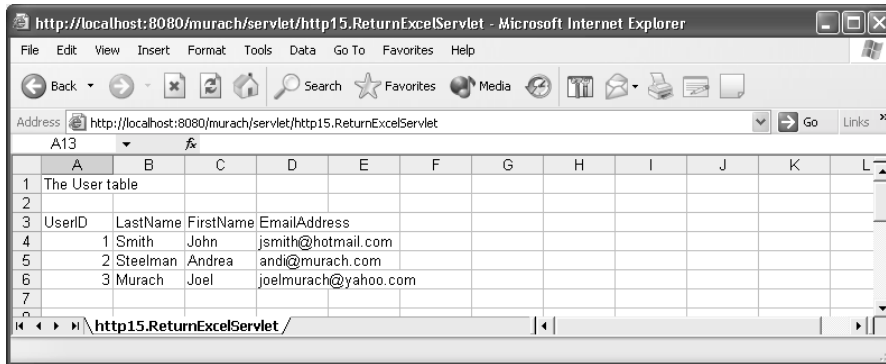
The code in this figure starts by storing the column headings for the spreadsheet in a StringBuffer object named report. Then, it retrieves all of the columns and rows from the User table in the Murach database and stores this data in tab-delimited format in the StringBuffer object.

When the code finishes storing the tab-delimited data in the StringBuffer object, the first statement in the highlighted code uses the `setContentType` method to set the content-type header of the response object. Here, the value in the header indicates that the response body contains data that's intended to be opened with Microsoft Excel. Then, the last two statements get a `PrintWriter` object that's used to return the StringBuffer object to the client.

How to control caching

The second statement in the highlighted code uses the `setHeader` method to set the cache-control response header so the document that's returned won't be cached. Otherwise, the server might automatically cache the response. Although it's usually more efficient to allow a document to be cached, preventing caching makes sure that the user updates the data with every visit.

An Excel spreadsheet displayed within the Internet Explorer



Code that returns a tab-delimited file as an Excel spreadsheet

```
String query = "SELECT * FROM User ORDER BY UserID";
String d = "\t";
StringBuffer report = new StringBuffer("The User table\n\n"
    + "UserID" + d
    + "LastName" + d
    + "FirstName" + d
    + "EmailAddress" + "\n");

try{
    Statement statement = connection.createStatement();
    ResultSet results = statement.executeQuery(query);
    while (results.next()){
        report.append(results.getInt("UserID") + d
            + results.getString("LastName")+ d
            + results.getString("FirstName") + d
            + results.getString("EmailAddress") + "\n");
    }
    results.close();
    statement.close();
}
catch(SQLException e){
    System.out.println("SQLException: " + e.getMessage());
}
response.setContentType("application/vnd.ms-excel");
response.setHeader("cache-control", "no-cache");

PrintWriter out = response.getWriter();
out.println(report);
```

Description

- To view the complete code for this servlet, open the ReturnExcelServlet class that's stored in the murach\WEB-INF\classes\http15 directory.

Figure 15-10 How to return a tab-delimited file as an Excel spreadsheet

How to encode the response with GZIP compression

Figure 15-11 shows how to encode a response with GZIP compression. Since encoding and decoding a document with compression can improve performance for large documents, you may occasionally want to use this technique. For example, if the table named `User` contains a large amount of data, it might make sense to encode the tab-delimited document that's returned in the last figure with GZIP compression. To do that, you just add the code shown in this figure.

This code checks a request header named `accept-encoding` and sets a response header named `content-encoding`. Here, the first statement uses the `getHeader` method to return the `accept-encoding` request header. This request header is a string that contains the types of encoding that the browser supports. Then, the second statement declares a `PrintWriter` object.

If the browser supports GZIP encoding, the statements within the `if` block create a `PrintWriter` object that uses a `GZIPOutputStream` to compress the output stream. In addition, the third statement uses the `setHeader` method to set the `content-encoding` response header to GZIP. That way, the browser knows to use GZIP to decompress the stream before trying to read it. On the other hand, if the browser doesn't support GZIP encoding, the `getWriter` method of the response object is used to return a normal `PrintWriter` object. Either way, the `PrintWriter` object is used to return the report to the browser.

Version 1.4 of the J2SE API includes the `GZIPOutputStream` class in the `java.util.zip` package. So to use this class, you need to import this package. You can get more information about this class by looking it up in the documentation for the J2SE API.

How to encode a response with GZIP compression

```
String encodingString = request.getHeader("accept-encoding");
PrintWriter out;

if ((encodingString != null)
    && (encodingString.indexOf("gzip") > -1)){

    OutputStream outputStream = response.getOutputStream();
    out = new PrintWriter(
        new GZIPOutputStream(outputStream), false);
    response.setHeader("content-encoding", "gzip");
}
else{
    out = response.getWriter();
}
out.println(report);
```

Description

- The GZIPOutputStream class is included in the java.util.zip package of version 1.4 of the J2SE.
- The getOutputStream method of the response object returns a binary output stream. The getWriter method of the response object returns a text output stream.
- To view the complete code for this servlet, open the ReturnGZIPServlet class that's stored in the murach\WEB-INF\classes\http15 directory.

How to require the File Download dialog box

When coding a web application, you can code an HTML link that points to a downloadable file. For example, let's say you code a link that points to a *Portable Document Format*, or *PDF* file. When a user clicks on this link, most systems will automatically start Adobe's Acrobat Reader and attempt to display the PDF file within the Acrobat Reader.

Although this is adequate for some applications, the Acrobat Reader doesn't indicate how long it will take to open the file. As a result, for a large PDF file, the user may be left staring at a blank screen for several minutes while the document downloads. In that case, you might want to display a File Download dialog box like the one shown in figure 15-12. That way, the user will have the choice of opening the PDF file or saving the file to disk, and the user will see a dialog box that indicates the progress of the download.

The code in this figure begins by showing how to code an HTML link to a MP3 sound file. On many systems, clicking on a link like this automatically launches an audio player that plays the sound file. If that's not what you want, you can code an HTML link to a JSP that forces the File Download dialog box to be displayed by setting a content-disposition header.

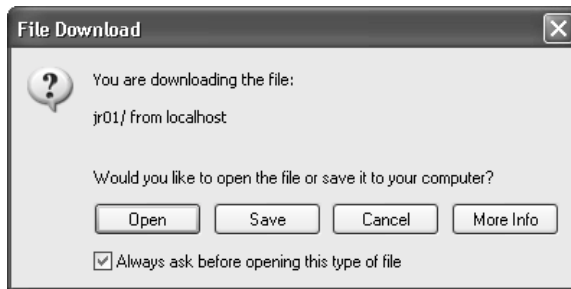
The code for the JSP sets two response headers and creates a `FileInputStream` that's used to read the specified file and convert it into a binary stream. To start, the JSP uses a page directive to import the `java.io` package. Then, the first two statements within the scriptlet get the path and name of the file that were sent as request parameters. After that, the third statement in the scriptlet sets the content-type response header to indicate that the response will contain generic binary data, and the fourth statement sets the content-disposition header to indicate that the response contains an attached file. This will force the File Download dialog box to be displayed. Once the response headers have been set, the scriptlet uses a `PrintWriter` object to read each byte of the specified file and write it to the response.

When using a JSP like this, you must make sure that the response doesn't include any bytes besides the binary data that's stored in the file. That's why the ending tag for the page directive ends on the same line as the beginning tag of the scriptlet like this:

```
%><%
```

Otherwise, the response would begin with the new line character that separates the page directive from the scriptlet, and the browser wouldn't be able to understand the response.

The File Download dialog box



An HTML link to an MP3 sound file

```
<a href="c:/tomcat/webapps/murach/sound/jr01/filter.mp3">Filter</a>
```

An HTML link to the download.jsp file

```
<a href=
"download.jsp?name=filter.mp3&path=c:/tomcat/webapps/murach/sound/jr01/">
Filter</a><br>
```

The download.jsp file

```
<%@ page import="java.io.*"
%><%
String path = request.getParameter("path");
String name = request.getParameter("name");

response.setContentType("application/octet-stream");
response.setHeader("content-disposition", "attachment; filename=" + name);

FileInputStream in = new FileInputStream(path + name);
int i;
while ((i = in.read()) != -1)
    out.write(i);
in.close();

%>
```

Description

- To force a File Download dialog box to be displayed, set the content-disposition header as shown above.
- To view the code for this JSP, open the download.jsp file that's stored in the murach\http15 directory.

Note

- If the File Download dialog box is displayed twice on your system, you can deselect the "Always ask before opening this type of file" option. Then, the File Download dialog box will only be displayed once.

Perspective

The goals of this chapter have been (1) to give you a better idea of what HTTP requests and responses consist of, and (2) to show you how you can use Java to work with HTTP requests and responses. If this chapter has been successful, you should now be able to use Java code to check the values in the headers of an HTTP request and also to set the status code and values of the headers of an HTTP response. Most of the time, though, the servlet API will shield you from having to work directly with HTTP.

Summary

- An HTTP request consists of a *request line* followed by *request headers*, while an HTTP response consists of a *status line* followed by *response headers* and then by a *response body*. The headers specify the attributes of a request or a response.
- The *Multipurpose Internet Mail Extension (MIME)* types provide standards for various types of data that can be transferred across the Internet.
- You can use the get methods of the request object to get the values of request headers, and you can use the set methods of the response object to set the values of response headers. This is useful for some applications.

Terms

Hypertext Transfer Protocol (HTTP)	response entity
request line	response body
request header	Multipurpose Internet Mail Extension (MIME)
status line	Portable Document Format (PDF)
status code	
response header	

Objectives

- Use the get and set methods of the request and response objects whenever you need them.
- Describe the components of an HTTP request and an HTTP response.
- Describe a case in which you would need to know the values of one or more request headers.
- Describe a case in which you would need to set the values of one or more response headers.

Exercise 15-1 Run the sample applications

1. Run the first three applications in the `http15` directory to see how the request headers can be returned in different formats. Then, view the code for each of the JSPs that return the request headers so you can see how the different MIME types are returned.
2. Run the application that displays the users in the User table in an Excel spreadsheet. Then, compare this with the application that uses GZIP compression to return the spreadsheet. View the servlet code for these applications in the `http15` package to see how these applications work.
3. Run the application that uses an HTML link to download a MP3 file. Then, run the application that uses a JSP to download the same file. This time, the application should display a dialog box similar to the one in figure 15-12. Perform the same task for the last two applications that download PDF files.

Exercise 15-2 Enhance the Download application

Enhance the Download application presented in chapter 7 so it displays a File Download dialog box for all sound files. To do this, modify each of the product download JSPs so the hyperlinks refer to the download JSP in the `http15` directory.