Table of Contents

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

Chapter 3. Basic Web Concepts

Java can do a lot more than create flashy web pages. Nonetheless, many of your programs will be applets on web pages, servlets running on the server, or web services that need to talk to other web servers and clients. Therefore, it's important to have a solid understanding of the interaction between web servers and web browsers.

The Hypertext Transfer Protocol (HTTP) is a standard that defines how a web client talks to a server and how data is transferred from the server back to the client. The architecture and design of the HTTP protocol is Representational State Transfer (REST). HTTP can be used to transfer data in essentially any format, from TIFF pictures to Microsoft Word documents to DBase files. However, far and away the most common format for data transferred over the Web and in some sense the Web's native format is the Hypertext Markup Language (HTML). HTML is a simple standard for describing the semantic value of textual data. You can say "this is a header", "this is a list item", "this deserves emphasis", and so on, but you can't specify how headers, lists, and other items are formatted: formatting is up to the browser. HTML is a "hypertext markup language" because it includes a way to specify links to other documents identified by URLs. A URL is a way to unambiguously identify the location of a resource on the Internet. To understand network programming, you'll need to understand URLs, HTML, and HTTP in somewhat more detail than the average web page designer.

3.1. URIs

A Uniform Resource Identifier (URI) is a string of characters in a particular syntax that identifies a resource. The resource identified may be a file on a server, but it may also be an email address, a news message, a book, a person's name, an Internet host, the current stock price of Sun Microsystems, or something else. An absolute URI is made up of a scheme for the URI and a scheme-specific part, separated by a colon, like this:

```
scheme:scheme-specific-part
```


Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

The syntax of the scheme-specific part depends on the scheme being used. Current schemes include:

data
: Base64-encoded data included directly in a link; see RFC 2397

file
: A file on a local disk

ftp
: An FTP server

http
: A World Wide Web server using the Hypertext Transfer Protocol

gopher
: A Gopher server

mailto
: An email address

news
: A Usenet newsgroup

telnet
: A connection to a Telnet-based service

urn
: A Uniform Resource Name

In addition, Java makes heavy use of nonstandard custom schemes such as *rmi*, *jndi*, and *doc* for various purposes. We'll look at the mechanism behind this in Chapter 16, when we discuss protocol handlers.

There is no specific syntax that applies to the scheme-specific parts of all URIs. However, many have a hierarchical form, like this:

```
//authority/path?query
```

The *authority* part of the URI names the authority responsible for resolvin g the rest of the URI. For instance, the URI http://www.ietf.org/rfc/rfc2396.txt has the scheme *http* and the authority www.ietf.org. This means the server at www.ietf.org is responsible for mapping the path */rfc/rfc2396.txt* to a resource. This URI does not have a query part. The URI http://www.powells.com/cgi-bin/biblio?inkey=62-1565928709-0 has the scheme *http*, the authority www.powells.com, the path /biblio, and the query inkey=62-1565928709-0. The URI *urn:isbn:156592870* has the scheme *urn* but doesn't follow the hierarchical *//authority/path?query* form for scheme-specific parts.

Although most current examples of URIs use an Internet host as an authority, future schemes may not. However, if the authority is an Internet host, optional usernames and ports may also be provided to make the authority more specific. For example, the URI *ftp://mp3:mp3@ci43198-a.ashvil1.nc.home.com:33/VanHalen-Jump.mp3* has the authority *mp3:mp3@ci43198-a.ashvil1.nc.home.com:33*. This authority has the username *mp3*, the password *mp3*, the host *ci43198-a.ashvil1.nc.home.com*, and the port *33*. It has the scheme *ftp* and the path */VanHalen-Jump.mp3*. (In most cases, including the password in the URI is a

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

big security hole unless, as here, you really do want everyone in the universe to know the password.)

The path (which includes its initial */*) is a string that the authority can use to determine which resource is identified. Different authorities may interpret the same path to refer to different resources. For instance, the path */index.html* means one thing when the authority is www.landoverbaptist.org and something very different when the authority is www.churchofsatan.com. The path may be hierarchical, in which case the individual parts are separated by forward slashes, and the . and .. operators are used to navigate the hierarchy. These are derived from the pathname syntax on the Unix operating systems where the Web and URLs were invented. They conveniently map to a filesystem stored on a Unix web server. However, there is no guarantee that the components of any particular path actually correspond to files or directories on any particular filesystem. For example, in the URI http://www.amazon.com/exec/obidos/ISBN%3D1565924851/cafeaulaitA/002-3777605-3043449, all the pieces of the hierarchy are just used to pull information out of a database that's never stored in a filesystem. *ISBN%3D1565924851* selects the particular book from the database by its ISBN number, *cafeaulaitA* specifies who gets the referral fee if a purchase is made from this link, and *002-3777605-3043449* is a session key used to track the visitor's path through the site.

Some URIs aren't at all hierarchical, at least in the filesystem sense. For example, *snews://secnews.netscape.com/netscape.devs-java* has a path of */netscape.devs-java*. Although there's some hierarchy to the newsgroup names indicated by the . between *netscape* and *netscape.devs-java*, it's not visible as part of the URI.

The scheme part is composed of lowercase letters, digits, and the plus sign, period, and hyphen. The other three parts of a typical URI (authority, path, and query) should each be composed of the ASCII alphanumeric characters; that is, the letters A-Z, a-z, and the digits 0-9. In addition, the punctuation characters - _ . ! ~ * ' may also be used. All other characters, including non-ASCII alphanumerics such as á and , should be escaped by a percent sign (%) followed by the hexadecimal code for the character. For instance, á would be encoded as %E1. A URL so transformed is said to have been "x-www-form-urlencoded".

This process assumes that the character set is the Latin 1. The URI and URL specifications don't actually say what character set should be used, which means most software tends to use the local default character set. Thus, URLs containing non-ASCII characters aren't very interoperable across different platforms and languages. With the Web becoming more international and less English daily, this situation has become increasingly problematic. Work is ongoing to define Internationalized Resource Identifiers (IRIs) that can use the full range of Unicode. At the time of this writing, the IRI draft specification indicates that non-ASCII characters should be encoded by first converting them to UTF-8, then percent-escaping each byte of the UTF-8, as specified above. For instance, the Greek letter is Unicode code point

3C0. In UTF-8, this letter is encoded as the three bytes E0, A7, 80. Thus in a URL it would be encoded as %E0%A7%80.

Punctuation characters such as / and @ must also be encoded with percent escapes if they are used in any role other than what's specified for them in the scheme-specific part of a particular URL. For example, the forward slashes in the URI http://www.cafeaulait.org/books/javaio/ do not need to be encoded as *%2F* because they serve to delimit the hierarchy as specified for the *http* URI scheme. However, if a filename includes a / character—for instance, if the last directory were named *Java I/O* instead of *javaio* to more closely match the name of the book—the URI would have to be written as http://www.cafeaulait.org/books/Java%20I%2FO/. This is not as farfetched as it might sound to Unix or Windows users. Mac filenames frequently include a forward slash. Filenames on many platforms often contain characters that need to be encoded, including @, $, +, =, and many more.

3.1.1. URNs

There are two types of URIs: Uniform Resource Locators (URLs) and Uniform Resource Names (URNs). A URL is a pointer to a particular resource on the Internet at a particular location. For example, http://www.oreilly.com/catalog/javanp3/ is one of several URLs for the book *Java Network Programming*. A URN is a name for a particular resource but without reference to a particular location. For instance, *urn:isbn:1565928709* is a URN referring to the same book. As this example shows, URNs, unlike URLs, are not limited to Internet resources.

The goal of URNs is to handle resources that are mirrored in many different locations or that have moved from one site to another; they identify the resource itself, not the place where the resource lives. For instance, when given a URN for a particular piece of software, an FTP program should get the file from the nearest mirror site. Given a URN for a book, a browser might reserve the book at the local library or order a copy from a bookstore.

A URN has the general form:

```
urn:namespace:resource_name
```

The `namespace` is the name of a collection of certain kinds of resources maintained by some authority. The `resource_name` is the name of a resource within that collection. For instance, the URN *urn:ISBN:1565924851* identifies a resource in the *ISBN* namespace with the identifier *1565924851*. Of all the books published, this one selects the first edition of *Java I/O*.

The exact syntax of resource names depends on the namespace. The ISBN namespace expects to see strings composed of 10 or 13 characters, all of which are digits—with the single

exception that the last character may be the letter *X* (either upper- or lowercase) instead. Furthermore, ISBNs may contain hyphens that are ignored when comparing. Other namespaces will use very different syntaxes for resource names. The IANA is responsible for handing out namespaces to different organizations, as described in RFC 3406. Basically, you have to submit an Internet draft to the IETF and publish an announcement on the urn-nid mailing list for public comment and discussion before formal standardization.

3.1.2. URLs

A URL identifies the location of a resource on the Internet. It specifies the protocol used to access a server (e.g., FTP, HTTP), the name of the server, and the location of a file on that server. A typical URL looks like http://www.ibiblio.org/javafaq/javatutorial.html. This specifies that there is a file called *javatutorial.html* in a directory called *javafaq* on the server *www.ibiblio.org*, and that this file can be accessed via the HTTP protocol. The syntax of a URL is:

```
protocol://username@hostname:port/path/filename?query#fragment
```

Here the protocol is another word for what was called the scheme of the URI. (*Scheme* is the word used in the URI RFC. *Protocol* is the word used in the Java documentation.) In a URL, the protocol part can be *file*, *ftp*, *http*, *https*, *gopher*, *news*, *telnet*, *wais*, or various other strings (though not *urn*).

The *hostname* part of a URL is the name of the server that provides the resource you want, such as *www.oreilly.com* or *utopia.poly.edu*. It can also be the server's IP address, such as 204.148.40.9 or 128.238.3.21. The *username* is an optional username for the server. The *port* number is also optional. It's not necessary if the service is running on its default port (port 80 for HTTP servers).

The *path* points to a particular directory on the specified server. The path is relative to the document root of the server, not necessarily to the root of the filesystem on the server. As a rule, servers that are open to the public do not show their entire filesystem to clients. Rather, they show only the contents of a specified directory. This directory is called the document root, and all paths and filenames are relative to it. Thus, on a Unix server, all files that are available to the public might be in */var/public/html*, but to somebody connecting from a remote machine, this directory looks like the root of the filesystem.

The filename points to a particular file in the directory specified by the path. It is often omitted —in which case, it is left to the server's discretion what file, if any, to send. Many servers send an index file for that directory, often called *index.html* or *Welcome.html*. Some send a list of

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

the files and folders in the directory, as shown in Figure 3-1. Others may send a 403 Forbidden error message, as shown in Figure 3-2.

Figure 3-1. A web server configured to send a directory list when no index file exists

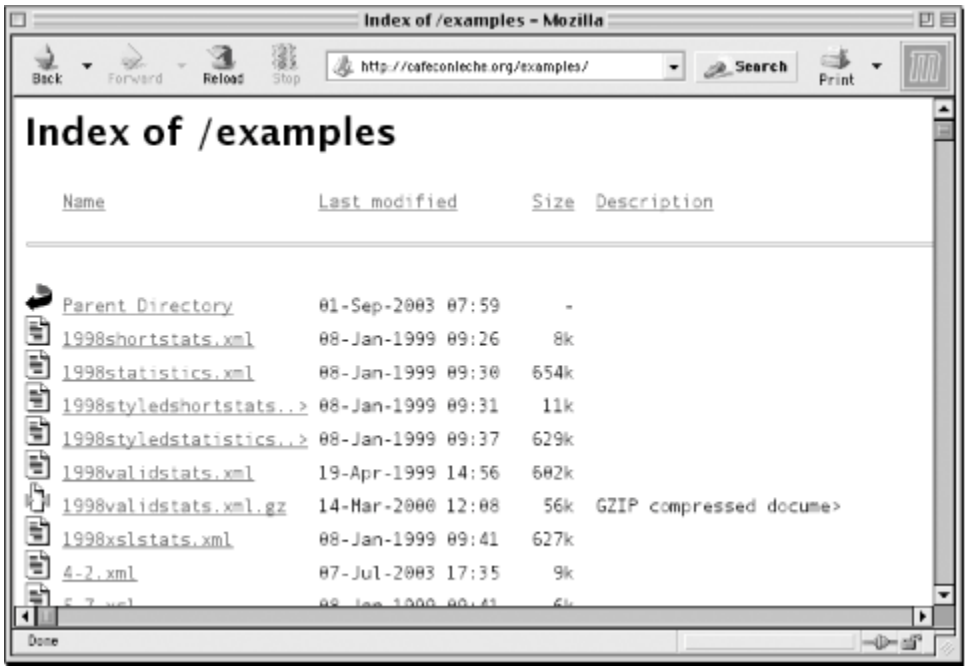
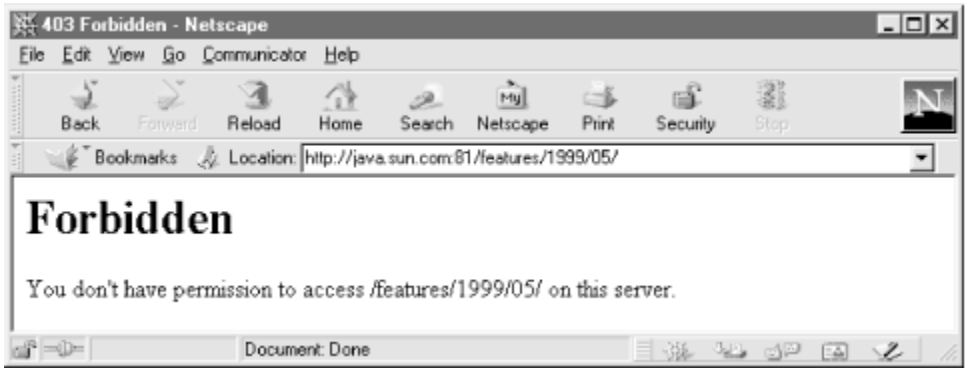


Figure 3-2. A web server configured to send a 403 error when no index file exists



The *query* string provides additional arguments for the server. It's commonly used only in *http* URLs, where it contains form data for input to programs running on the server.

Finally, the *fragment* references a particular part of the remote resource. If the remote resource is HTML, the fragment identifier names an anchor in the HTML document. If the remote resource is XML, the fragment identifier is an XPointer. Some documents refer to the fragment part of the URL as a "section"; Java documents rather unaccountably refer to the

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

fragment identifier as a "Ref". A named anchor is created in an HTML document with a tag, like this:

```
<A NAME="xtocid1902914">Comments</A>
```

This tag identifies a particular point in a document. To refer to this point, a URL includes not only the document's filename but the named anchor separated from the rest of the URL by a #:

```
http://www.cafeaulait.org/javafaq.html#xtocid1902914
```

Technically, a string that contains a fragment identifier is a URL reference, not a URL. Java, however, does not distinguish between URLs and URL references.

3.1.3. Relative URLs

A URL tells the web browser a lot about a document: the protocol used to retrieve the document, the name of the host where the document lives, and the path to that document on the host. Most of this information is likely to be the same for other URLs that are referenced in the document. Therefore, rather than requiring each URL to be specified in its entirety, a URL may inherit the protocol, hostname, and path of its parent document (i.e., the document in which it appears). URLs that aren't complete but inherit pieces from their parent are called *relative* URLs. In contrast, a completely specified URL is called an *absolute URL*. In a relative URL, any pieces that are missing are assumed to be the same as the corresponding pieces from the URL of the document in which the URL is found. For example, suppose that while browsing http://www.ibiblio.org/javafaq/javatutorial.html you click on this hyperlink:

```
<a href="javafaq.html">
```

The browser cuts *javatutorial.html* off the end of http://www.ibiblio.org/javafaq/javatutorial.html to get http://www.ibiblio.org/javafaq/. Then it attaches *javafaq.html* onto the end of http://www.ibiblio.org/javafaq/ to get http://www.ibiblio.org/javafaq/javafaq.html. Finally, it loads that document.

If the relative link begins with a /, then it is relative to the document root instead of relative to the current file. Thus, if you click on the following link while browsing http://www.ibiblio.org/javafaq/javatutorial.html:

```
<a href="/boutell/faq/www_faq.html">
```

the browser would throw away */javafaq/javatutorial.html* and attach */boutell/faq/www_faq.html* to the end of http://www.ibiblio.org to get http://www.ibiblio.org/boutell/faq/www_faq.html.

Relative URLs have a number of advantages. First—and least important—they save a little typing. More importantly, relative URLs allow a single document tree to be served by multiple protocols: for instance, both FTP and HTTP. The HTTP might be used for direct surfing, while the FTP could be used for mirroring the site. Most importantly of all, relative URLs allow entire trees of documents to be moved or copied from one site to another without breaking all the internal links.

3.2. HTML, SGML, and XML

HTML is the primary format used for Web documents. As I said earlier, HTML is a simple standard for describing the semantic content of textual data. The idea of describing a text's semantics rather than its appearance comes from an older standard called the Standard Generalized Markup Language (SGML). Standard HTML is an instance of SGML. SGML was invented in the mid-1970s by Charles Goldfarb, Edward Mosher, and Raymond Lorie at IBM. SGML is now an International Standards Organization (ISO) standard, specifically ISO 8879:1986.

SGML and, by inheritance, HTML are based on the notion of design by meaning rather than design by appearance. You don't say that you want some text printed in 18-point type; you say that it is a top-level heading (<H1> in HTML). Likewise, you don't say that a word should be placed in italics. Rather, you say it should be emphasized (<EM> in HTML). It is left to the browser to determine how to best display headings or emphasized text.

The tags used to mark up the text are case-insensitive. Thus, <STRONG> is the same as is the same as <Strong> is the same as <StrONg>. Some tags have a matching end-tag to define a region of text. An end-tag is the same as the start-tag, except that the opening angle bracket is followed by a /. For example: <STRONG>this text is strong</STRONG>; <EM>this text is emphasized</EM>. The entire text from the beginning of the start-tag to the end of the end-tag is called an *element*. Thus, <STRONG>this text is strong</STRONG> is a STRONG element.

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

HTML elements may nest but they should not overlap. The first line in the following example is standard-conforming. The second line is not, though many browsers accept it nonetheless:

```
<STRONG><EM>Jack and Jill went up the hill</EM></STRONG>
<STRONG><EM>to fetch a pail of water</STRONG></EM>
```

Some elements have additional attributes that are encoded as name-value pairs on the start-tag. The `<H1>` tag and most other paragraph-level tags may have an `ALIGN` attribute that says whether the header should be centered, left-aligned, or right-aligned. For example:

```
<H1 ALIGN=CENTER> This is a centered H1 heading </H1>
```

The value of an attribute may be enclosed in double or single quotes, like this:

```
<H1 ALIGN="CENTER"> This is a centered H1 heading </H1>
<H2 ALIGN='LEFT'> This is a left-aligned H2 heading </H2>
```

Quotes are required only if the value contains embedded spaces. When processing HTML, you need to be prepared for attribute values that do and don't have quotes.

There have been several versions of HTML over the years. The current standard is HTML 4.0, most of which is supported by current web browsers, with occasional exceptions. Furthermore, several companies, notably Netscape, Microsoft, and Sun, have added nonstandard extensions to HTML. These include blinking text, inline movies, frames, and, most importantly for this book, applets. Some of these extensions—for example, the `<APPLET>` tag—are allowed but deprecated in HTML 4.0. Others, such as Netscape's notorious `<BLINK>`, come out of left field and have no place in a semantically-oriented language like HTML.

HTML 4.0 may be the end of the line, aside from minor fixes. The W3C has decreed that HTML is getting too bulky to layer more features on top of. Instead, new development will focus on XML, a semantic language that allows page authors to create the elements they need rather than relying on a few fixed elements such as `P` and `LI`. For example, if you're writing a web page with a price list, you would likely have an `SKU` element, a `PRICE` element, a `MANUFACTURER` element, a `PRODUCT` element, and so forth. That might look something like this:

```
<PRODUCT MANUFACTURER="IBM">
  <NAME>Lotus Smart Suite</NAME>
  <VERSION>9.8</VERSION>
  <PLATFORM>Windows</PLATFORM>
  <PRICE CURRENCY="US">299.95</PRICE>
  <SKU>D05WGML</SKU>
</PRODUCT>
```


Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

This looks a lot like HTML, in much the same way that Java looks like C. There are elements and attributes. Tags are set off by < and >. Attributes are enclosed in quotation marks, and so forth. However, instead of being limited to a finite set of tags, you can create all the new and unique tags you need. Since no browser can know in advance all the different elements that may appear, a *stylesheet* is used to describe how each of the items should be displayed.

XML has another advantage over HTML that may not be obvious from this simple example. HTML can be quite sloppy. Elements are opened but not closed. Attribute values may or may not be enclosed in quotes. The quotes may or may not be present. XML tightens all this up. It lays out very strict requirements for the syntax of a well-formed XML document, and it requires that browsers reject all malformed documents. Browsers may not attempt to fix the problem and make a best-faith effort to display what they think the author meant. They must simply report the error. Furthermore, an XML document may have a Document Type Definition (DTD), which can impose additional constraints on valid documents. For example, a DTD may require that every `PRODUCT` element contain exactly one `NAME` element. This has a number of advantages, but the key one here is that XML documents are far easier to parse than HTML documents. As a programmer, you will find it much easier to work with XML than HTML.

XML can be used both for pure XML pages and for embedding new kinds of content in HTML and XHTML. For example, the Mathematical Markup Language, MathML, is an XML application for including mathematical equations in web pages. SMIL, the Synchronized Multimedia Integration Language, is an XML application for including timed multimedia such as slide shows and subtitled videos on web pages. More recently, the W3C has released several versions of XHTML. This language uses the familiar HTML vocabulary (`p` for paragraphs, `tr` for table rows, `img` for pictures, and so forth) but requires the document to follow XML's stricter rules: all attribute values must be quoted; every start-tag must have a matching end-tag; elements can nest but cannot overlap; etc. For a lot more information about XML, see *XML in a Nutshell* by Elliotte Rusty Harold and W. Scott Means (O'Reilly).

3.3. HTTP

HTTP is the standard protocol for communication between web browsers and web servers. HTTP specifies how a client and server establish a connection, how the client requests data from the server, how the server responds to that request, and finally, how the connection is closed. HTTP connections use the TCP/IP protocol for data transfer. For each request from client to server, there is a sequence of four steps:

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

Making the connection

The client establishes a TCP connection to the server on port 80, by default; other ports may be specified in the URL.

Making a request

The client sends a message to the server requesting the page at a specified URL. The format of this request is typically something like:

```
GET /index.html HTTP/1.0
```

`GET` specifies the operation being requested. The operation requested here is for the server to return a representation of a resource. `/index.html` is a relative URL that identifies the resource requested from the server. This resource is assumed to reside on the machine that receives the request, so there is no need to prefix it with `http://www.thismachine.com/`. `HTTP/1.0` is the version of the protocol that the client understands. The request is terminated with two carriage return/linefeed pairs (`\r\n\r\n` in Java parlance), regardless of how lines are terminated on the client or server platform.

Although the `GET` line is all that is required, a client request can include other information as well. This takes the following form:

```
Keyword: Value
```

The most common such keyword is `Accept`, which tells the server what kinds of data the client can handle (though servers often ignore this). For example, the following line says that the client can handle four MIME media types, corresponding to HTML documents, plain text, and JPEG and GIF images:

```
Accept: text/html, text/plain, image/gif, image/jpeg
```

`User-Agent` is another common keyword that lets the server know what browser is being used, allowing the server to send files optimized for the particular browser type. The line below says that the request comes from Version 2.4 of the Lynx browser:

```
User-Agent: Lynx/2.4 libwww/2.1.4
```

All but the oldest first-generation browsers also include a `Host` field specifying the server's name, which allows web servers to distinguish between different named hosts served from the same IP address. Here's an example:

```
Host: www.cafeaulait.org
```

Finally, the request is terminated with a blank line—that is, two carriage return/linefeed pairs, `\r\n\r\n`. A complete request might look like this:

```
GET /index.html HTTP/1.0
Accept: text/html, text/plain, image/gif, image/jpeg
User-Agent: Lynx/2.4 libwww/2.1.4
Host: www.cafeaulait.org
```

In addition to `GET`, there are several other request types. `HEAD` retrieves only the header for the file, not the actual data. This is commonly used to check the modification date of a file, to see whether a copy stored in the local cache is still valid. `POST` sends form data to the server, `PUT` uploads a resource to the server, and `DELETE` removes a resource from the server.

The response

The server sends a response to the client. The response begins with a response code, followed by a header full of metadata, a blank line, and the requested document or an error message. Assuming the requested document is found, a typical response looks like this:

```
HTTP/1.1 200 OK
Date: Mon, 15 Sep 2003 21:06:50 GMT
Server: Apache/2.0.40 (Red Hat Linux)
Last-Modified: Tue, 15 Apr 2003 17:28:57 GMT
Connection: close
Content-Type: text/html; charset=ISO-8859-1
Content-length: 107

<html>
<head>
<title>
A Sample HTML file
</title>
</head>
<body>
The rest of the document goes here
</body>
</html>
```


Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

The first line indicates the protocol the server is using (`HTTP/1.1`), followed by a response code. `200 OK` is the most common response code, indicating that the request was successful. Table 3-1 is a complete list of the response codes used by HTTP 1.0; HTTP 1.1 adds many more to this list. The other header lines identify the date the request was made in the server's time frame, the server software (Apache 2.0.40), the date this document was last modified, a promise that the server will close the connection when it's finished sending, the MIME content type, and the length of the document delivered (not counting this header)—in this case, 107 bytes.

Closing the connection

Either the client or the server or both close the connection. Thus, a separate network connection is used for each request. If the client reconnects, the server retains no memory of the previous connection or its results. A protocol that retains no memory of past requests is called *stateless*; in contrast, a *stateful* protocol such as FTP can process many requests before the connection is closed. The lack of state is both a strength and a weakness of HTTP.

Table 3-1. HTTP 1.0 response codes

Response code	Meaning
2xx Successful	Response codes between 200 and 299 indicate that the request was received, understood, and accepted.
200 OK	This is the most common response code. If the request used `GET` or `POST`, the requested data is contained in the response along with the usual headers. If the request used `HEAD`, only the header information is included.
201 Created	The server has created a data file at a URL specified in the body of the response. The web browser should now attempt to load that URL. This is sent only in response to `POST` requests.
202 Accepted	This rather uncommon response indicates that a request (generally from `POST`) is being processed, but the processing is not yet complete so no response can be returned. The server should return an HTML page that explains the situation to the user, provides an estimate of when the request is likely to be completed, and, ideally, has a link to a status monitor of some kind.
204 No Content	The server has successfully processed the request but has no information to send back to the client. This is usually the result of a poorly written form-processing program that accepts data but does not return a response to the user indicating that it has finished.
3xx Redirection	Response codes from 300 to 399 indicate that the web browser needs to go to a different page.
300 Multiple Choices	The page requested is available from one or more locations. The body of the response includes a list of locations from which the user or web browser can pick the most appropriate one. If the server prefers one of these locations, the URL of this choice is included in a `Location` header, which web browsers can use to load the preferred page.

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

Response code	Meaning
301 Moved Permanently	The page has moved to a new URL. The web browser should automatically load the page at this URL and update any bookmarks that point to the old URL.
302 Moved Temporarily	This unusual response code indicates that a page is temporarily at a new URL but that the document's location will change again in the foreseeable future, so bookmarks should not be updated.
304 Not Modified	The client has performed a `GET` request but used the `If-Modified-Since` header to indicate that it wants the document only if it has been recently updated. This status code is returned because the document has not been updated. The web browser will now load the page from a cache.
4xx Client Error	Response codes from 400 to 499 indicate that the client has erred in some fashion, although the error may as easily be the result of an unreliable network connection as of a buggy or nonconforming web browser. The browser should stop sending data to the server as soon as it receives a 4xx response. Unless it is responding to a `HEAD` request, the server should explain the error status in the body of its response.
400 Bad Request	The client request to the server used improper syntax. This is rather unusual, although it is likely to happen if you're writing and debugging a client.
401 Unauthorized	Authorization, generally username and password controlled, is required to access this page. Either the username and password have not yet been presented or the username and password are invalid.
403 Forbidden	The server understood the request but is deliberately refusing to process it. Authorization will not help. One reason this occurs is that the client asks for a directory listing but the server is not configured to provide it, as shown in Figure 3-1.
404 Not Found	This most common error response indicates that the server cannot find the requested page. It may indicate a bad link, a page that has moved with no forwarding address, a mistyped URL, or something similar.
5xx Server Error	Response codes from 500 to 599 indicate that something has gone wrong with the server, and the server cannot fix the problem.
500 Internal Server Error	An unexpected condition occurred that the server does not know how to handle.

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

Response code	Meaning
501 Not Implemented	The server does not have the feature that is needed to fulfill this request. A server that cannot handle `POST` requests might send this response to a client that tried to `POST` form data to it.
502 Bad Gateway	This response is applicable only to servers that act as proxies or gateways. It indicates that the proxy received an invalid response from a server it was connecting to in an effort to fulfill the request.
503 Service Unavailable	The server is temporarily unable to handle the request, perhaps as a result of overloading or maintenance.

HTTP 1.1 more than doubles the number of responses. However, a response code from 200 to 299 always indicates success, a response code from 300 to 399 always indicates redirection, one from 400 to 499 always indicates a client error, and one from 500 to 599 indicates a server error.

HTTP 1.0 is documented in the informational RFC 1945; it is not an official Internet standard because it was primarily developed outside the IETF by early browser and server vendors. HTTP 1.1 is a proposed standard being developed by the W3C and the HTTP working group of the IETF. It provides for much more flexible and powerful communication between the client and the server. It's also a lot more scalable. It's documented in RFC 2616. HTTP 1.0 is the basic version of the protocol. All current web servers and browsers understand it. HTTP 1.1 adds numerous features to HTTP 1.0, but doesn't change the underlying design or architecture in any significant way. For the purposes of this book, it will usually be sufficient to understand HTTP 1.0.

The primary improvement in HTTP 1.1 is *connection reuse*. HTTP 1.0 opens a new connection for every request. In practice, the time taken to open and close all the connections in a typical web session can outweigh the time taken to transmit the data, especially for sessions with many small documents. HTTP 1.1 allows a browser to send many different requests over a single connection; the connection remains open until it is explicitly closed. The requests and responses are all asynchronous. A browser doesn't need to wait for a response to its first request before sending a second or a third. However, it remains tied to the basic pattern of a client request followed by a server response. Each request and response has the same basic form: a header line, an HTTP header containing metadata, a blank line, and then the data itself.

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

There are a lot of other, smaller improvements in HTTP 1.1. Requests include a `Host` header field so that one web server can easily serve different sites at different URLs. Servers and browsers can exchange compressed files and particular byte ranges of a document, both of which decrease network traffic. And HTTP 1.1 is designed to work much better with proxy servers. HTTP 1.1 is a superset of HTTP 1.0, so HTTP 1.1 web servers have no trouble interacting with older browsers that only speak HTTP 1.0, and vice versa.

3.4. MIME Media Types

MIME is an open standard for sending multipart, multimedia data through Internet email. The data may be binary, or it may use multiple ASCII and non-ASCII character sets. Although MIME was originally intended just for email, it has become a widely used technique to describe a file's contents so that client software can tell the difference between different kinds of data. For example, a web browser uses MIME to tell whether a file is a GIF image or a printable PostScript file.

Officially, MIME stands for Multipurpose Internet Mail Extensions, which is the expansion of the acronym used in RFC 2045. However, you will hear other versions—most frequently Multipart Internet Mail Extensions and Multimedia Internet Mail Extensions.

MIME supports more than 100 predefined types of content. Content types are classified at two levels: a type and a subtype. The type shows very generally what kind of data is contained: is it a picture, text, or movie? The subtype identifies the specific type of data: GIF image, JPEG image, TIFF image. For example, HTML's content type is `text/html`; the type is `text`, and the subtype is `html`. The content type for a GIF image is `image/gif`; the type is `image`, and the subtype is `gif`. Table 3-2 lists the more common defined content types. On most systems, a simple text file maintains a mapping between MIME types and the application used to process that type of data; on Unix, this file is called *mime.types*. The most current list of registered MIME types is available from http://www.iana.org/assignments/media-types/. For more on MIME, see the *comp.mail.mime* FAQ at http://www.uni-giessen.de/faq/archiv/mail.mime-faq.part1-9/.

Web servers use MIME to identify the kind of data they're sending. Web clients use MIME to identify the kind of data they're willing to accept. Most web servers and clients understand at least two MIME text content types, `text/html` and `text/plain`, and two image formats, `image/gif` and `image/jpeg`. More recent browsers also understand `application/`

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

`xml` and several other image formats. Java relies on MIME types to pick the appropriate content handler for a particular stream of data.

Table 3-2. Predefined MIME content types

Type	Subtype	Description
text		The document represents printable text.
	calendar	Calendaring and scheduling information in the iCalendar format; see RFC 2445.
	css	A Cascading Style Sheet used for HTML and XML.
	directory	Address book information such as name, phone number, and email address; used by Netscape vCards; defined in RFCs 2425 and 2426.
	enriched	A very simple HTML-like language for adding basic font and paragraph-level formatting such as bold and italic to email; used by Eudora; defined in RFC 1896.
	html	Hypertext Markup Language as used by web browsers.
	plain	This is supposed to imply raw ASCII text. However, some web servers use `text/plain` as the default MIME type for any file they can't recognize. Therefore, anything and everything, most notably *.class* byte code files, can get identified as a `text/plain` file.
	richtext	An HTML-like markup for encoding formatting into pure ASCII text. It's never really caught on, in large part because of the popularity of HTML.
	rtf	An incompletely defined Microsoft format for word processing files.
	sgml	The Standard Generalized Markup Language; ISO standard 8879:1986.
	tab-separated-values	The interchange format used by many spreadsheets and databases; records are separated by linebreaks and fields by tabs.
	xml	The W3C standard Extensible Markup Language. For various technical reasons, `application/xml` should be used instead, but often isn't.

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

Type	Subtype	Description
multipart		Multipart MIME messages encode several different files into one message.
	mixed	Several message parts intended for sequential viewing.
	alternative	The same message in multiple formats so a client may choose the most convenient one.
	digest	A popular format for merging many email messages into a single digest; used by many mailing lists and some FAQ lists.
	parallel	Several parts intended for simultaneous viewing.
	byteranges	Several separately contiguous byte ranges; used in HTTP 1.1.
	encrypted	One part for the body of the message and one part for the information necessary to decode the message.
	signed	One part for the body of the message and one part for the digital signature.
	related	Compound documents formed by aggregating several smaller parts.
	form-data	Form responses.
message		An email message.
	external-body	Just the headers of the email message; the message's body is not included but exists at some other location and is referenced, perhaps by a URL.
	http	An HTTP 1.1 request from a web client to a web server.
	news	A news article.

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

Type	Subtype	Description
	partial	Part of a longer email message that has been split into multiple parts to allow transmission through email gateways.
	rfc822	A standard email message including headers.
image		Two-dimensional pictures.
	cgm	A Computer Graphics Metafile format image. CGM is ISO standard 8632:1992 for device-independent vector graphics and bitmap images.
	g3fax	The standard for bitmapped fax images.
	gif	A Graphics Interchange Format image.
	jpeg	The Joint Photographic Experts Group file format for bitmapped images with lossy compression.
	png	A Portable Network Graphics Format image. The format was developed at the W3C as a modern replacement for GIF that supports 24-bit color and is not encumbered by patents.
	tiff	The Tagged Image File format from Adobe.
audio		Sound.
	basic	8-bit ISDN -law encoded audio with a single channel and a sample rate of eight kilohertz. This is the format used by *.au* and *.snd* files and supported by the `java.applet.AudioClip` class.
video		Video.
	mpeg	The Motion Picture Experts Group format for video data with lossy compression.
	quicktime	Apple's proprietary QuickTime movie format. Before being included in a MIME message, QuickTime files must be "flattened".

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

Type	Subtype	Description
model		3-D images.
	vrml	A Virtual Reality Modeling Language file, a format for 3-D data on the Web.
	iges	The Initial Graphics Exchange Specification for interchanging documents between different CAD programs.
	mesh	The mesh structures used in finite element and finite difference methods.
application		Binary data specific to some application.
	octet-stream	Unspecified binary data, which is usually saved into a file for the user. This MIME type is sometimes used to serve *.class* byte code files.
	java	A nonstandard subtype sometimes used to serve *.class* byte code files.
	postscript	Adobe PostScript.
	dca-rft	IBM's Document Content Architecture-Richly Formatted Text.
	mac-BinHex40	A means of encoding the two forks of a Macintosh document in a single ASCII file.
	pdf	An Adobe Acrobat file.
	zip	A zip compressed file.
	macwriteii	A MacWrite II word-processing document.
	msword	A Microsoft Word document.
	xml+xhtml	An XHTML document
	xml	An Extensible Markup Language document.

A MIME-compliant program is not required to understand all these different types of data; it just needs to recognize what it can and cannot handle. Many programs—Netscape Navigator, for example—use various helper programs to display types of content they themselves don't understand.

MIME allows you to define additional nonstandard subtypes by using the prefix `x-`. For example, the content type `application/x-tex` has the MIME type `application` and the nonstandard subtype `x-tex` for a TeX document. These x-types are not guaranteed to be understood by any program other than the one that created them. Indeed, two programs may use the same x-type to mean two completely different things, or different programs may use different x-types to mean the same thing. However, many nonstandard types have come into common use; some of the more common ones are listed in Table 3-3.

Table 3-3. X-types

Type	X-subtype	Description
application		Subtypes of an application; the name of the subtype is usually a file format name or an application name.
	x-aiff	SGI's AIFF audio data format.
	x-bitmap	An X Windows bitmap image.
	x-gzip	Data compressed in the GNU gzip format.
	x-dvi	A TeX DVI document.
	x-framemaker	A FrameMaker document.
	x-latex	A LaTeX document.
	x-macBinHex40	Identical to `application/mac-BinHex40`, but older software may use this x-type instead.
	x-mif	A FrameMaker MIF document.
	x-sd	A session directory protocol announcement, used to announce MBONE events.

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

Type	X-subtype	Description
	x-shar	A shell archive; the Unix equivalent of a Windows or Macintosh self-extracting archive. Software shouldn't be configured to unpack shell archives automatically, because a shell archive can call any program the user who runs it has the rights to call.
	x-tar	A tar archive.
	x-gtar	A GNU tar archive.
	x-tcl	A tool command language (TCL) program. You should never configure your web browser or email program to automatically run programs you download from the web or receive in email messages.
	x-tex	A TeX document.
	x-texinfo	A GNU texinfo document.
	x-troff	A troff document.
	x-troff-man	A troff document written with the *man* macros.
	x-troff-me	A troff document that should be processed using the *me* macros.
	x-troff-ms	A troff document that should be processed using the *ms* macros.
	x-wais-source	A WAIS source.
	x-www-form-urlencoded	A string that has been encoded like a URL, with + replacing spaces and % escapes replacing non-alphanumeric characters that aren't separators.
audio		
	x-aiff	The same as `application/x-aiff`: an AIFF audio file.

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

Type	X-subtype	Description
	x-mpeg	The MP3 sound format.
	x-mpeg.mp3	The MP3 sound format.
	x-wav	The Windows WAV sound format.
image		
	x-fits	The FITS image format used primarily by astronomers.
	x-macpict	A Macintosh PICT image.
	x-pict	A Macintosh PICT image.
	x-macpaint	A MacPaint image.
	x-pbm	A portable bitmap image.
	x-portable-bitmap	A portable bitmap image.
	x-pgm	A PGM image.
video		
	x-msvideo	A Microsoft AVI Video for Windows.
	x-sgi-movie	A Silicon Graphics movie.

3.5. Server-Side Programs

These days many web pages are not served from static files on the hard drive. Instead, the server generates them dynamically to meet user requests. The content may be pulled from a database or generated algorithmically by a program. Indeed, the actual page delivered to

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

the client may contain data combined from several different sources. In Java, such server-side programs are often written using servlets or Java Server Pages (JSP). They can also be written with other languages, such as C and Perl, or other frameworks, such as ASP and PHP. The concern in this book is not so much with how these programs are written as with how your programs communicate with them. One advantage to HTTP is that it really doesn't matter how the other side of the connection is written, as long as it speaks the same basic HTTP protocol.

The simplest server-side programs run without any input from the user. From the viewpoint of the client, these programs are accessed like any other web page and aren't of much concern to this book. The difference between a web page produced by a program that takes no input and a web page written in static HTML is all on the server side. When writing clients, you don't need to know or care whether the web server is feeding you a file or the output of some program it ran. Your interface to the server is the same in either case.

A slightly more complex server-side program processes user input from HTML forms. A web form is essentially just a way of collecting input from the user, dividing it into neat pieces, and passing those pieces to some program on the server. A client written in Java can perform the same function, either by asking the user for input in its own GUI or by providing its own unique information.

HTTP provides a standard, well understood and well supported means for Java applets and applications to talk to remote systems; therefore, I will cover how to use Java to both receive and send data to the server. There are other ways for Java programs to talk to servers, including Remote Method Invocation (RMI) and SOAP. However, RMI is slow and SOAP is quite complex. By way of contrast, HTTP is mature, robust, better supported across multiple platforms and web servers, and better understood in the web development community.

Example 3-1 and Figure 3-3 show a simple form with two fields that collects a name and an email address. The values the user enters in the form are sent back to the server when the user presses the "Submit Query" button. The program to run when the form data is received is */cgi/reg.pl*; the program is specified in the `ACTION` attribute of the `FORM` element. The URL in this parameter is usually a relative URL, as it is in this example.

Example 3-1. A simple form with input fields for a name and an email address

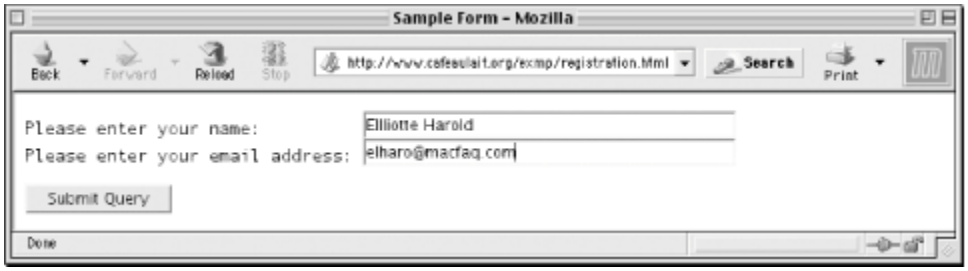
```
<HTML>
<HEAD>
<TITLE>Sample Form</TITLE>
</HEAD>
```


Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

```
    <BODY>

    <FORM METHOD=GET ACTION="/cgi/reg.pl">
    <PRE>
    Please enter your name:          <INPUT NAME="username" SIZE=40>
    Please enter your email address: <INPUT NAME="email" SIZE=40>
    </PRE>
    <INPUT TYPE="SUBMIT">
    </FORM>
    </BODY>
    </HTML>
```

Figure 3-3. A simple form



The web browser reads the data the user types and encodes it in a simple fashion. The name of each field is separated from its value by the equals sign (=). Different fields are separated from each other by an ampersand (&). Each field name and value is x-www-form-url-encoded; that is, any non-ASCII or reserved characters are replaced by a percent sign followed by hexadecimal digits giving the value for that character in some character set. Spaces are a special case because they're so common. Instead of being encoded as %20, they become the + sign. The plus sign itself is encoded as %2b. For example, the data from the form in Figure 3-3 is encoded as:

```
username=Elliotte+Harold&email=elharo%40macfaq.com
```

This is called the *query string*.

There are two methods by which the query string can be sent to the server: `GET` and `POST`. If the form specifies the `GET` method, the browser attaches the query string to the URL it sends to the server. Forms that specify `POST` send the query string on an output stream. The form in Example 3-1 uses `GET` to communicate with the server, so it connects to the server and sends the following command:

```
GET /cgi/reg.pl?username=Elliotte+Harold&email=elharo%40macfaq.com HTTP/1.0
```

The server uses the path component of the URL to determine which program should handle this request. It passes the query string's set of name-value pairs to that program, which normally takes responsibility for replying to the client.

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

With the POST method, the web browser sends the usual headers and follows them with a blank line (two successive carriage return/linefeed pairs) and then sends the query string. If the form in Example 3-1 used POST, it would send this to the server:

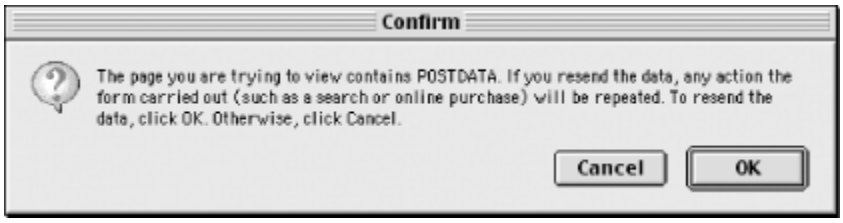
```
POST /cgi-bin/register.pl HTTP 1.0
Content-type: application/x-www-form-urlencoded
Content-length: 65

username=Elliotte+Harold&email=elharo%40metalab.unc.edu
```

There are many different form tags in HTML that produce pop-up menus, radio buttons, and more. However, although these input widgets appear different to the user, the format of data they send to the server is the same. Each form element provides a name and an encoded string value.

Because GET requests include all necessary information in the URL, they can be bookmarked, linked to, spidered, googled, and so forth. The results of a POST request cannot. This is deliberate. GET is intended for noncommital actions, like browsing a static web page. POST is intended for actions that commit to something. For example, adding items to a shopping cart should be done with GET, because this action doesn't commit; you can still abandon the cart. However, placing the order should be done with POST because that action makes a commitment. This is why browsers ask you if you're sure when you go back to a page that uses POST (as shown in Figure 3-4). Reposting data may buy two copies of a book and charge your credit card twice.

Figure 3-4. Repost confirmation



In practice, POST is vastly overused on the web today. Any safe operation that does not commit the user to anything should use GET rather than POST. Only operations that commit the user should use POST.

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004