tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. This is the foundation for data communication for the World Wide Web (i.e. internet) since 1990. HTTP is a generic and stateless protocol which can be used for other purposes as well using extensions of its request methods, error codes, and headers.

This tutorial is based on RFC-2616 specification, which defines the protocol referred to as HTTP/1.1. HTTP/1.1 is a revision of the original HTTP (HTTP/1.0). A major difference between HTTP/1.0 and HTTP/1.1 is that HTTP/1.0 uses a new connection for each request/response exchange, whereas HTTP/1.1 connection may be used for one or more request/response exchanges.

Audience

This tutorial has been prepared for computer science graduates and web developers to help them understand the basic-to-advanced level concepts related to Hypertext Transfer Protocol (HTTP).

Prerequisites

Before proceeding with this tutorial, it is good to have a basic understanding of web concepts, web browsers, web servers, client and server architecture based softwares.

Copyright & Disclaimer

Table of Contents

1. OVERVIEW

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. This is the foundation for data communication for the World Wide Web (i.e. internet) since 1990. HTTP is a generic and stateless protocol which can be used for other purposes as well using extensions of its request methods, error codes, and headers.

Basically, HTTP is a TCP/IP based communication protocol that is used to deliver data (HTML files, image files, query results, etc.) on the World Wide Web. The default port is TCP 80, but other ports can be used as well. It provides a standardized way for computers to communicate with each other. HTTP specification specifies how clients' request data will be constructed and sent to the server, and how the servers respond to these requests.

Basic Features

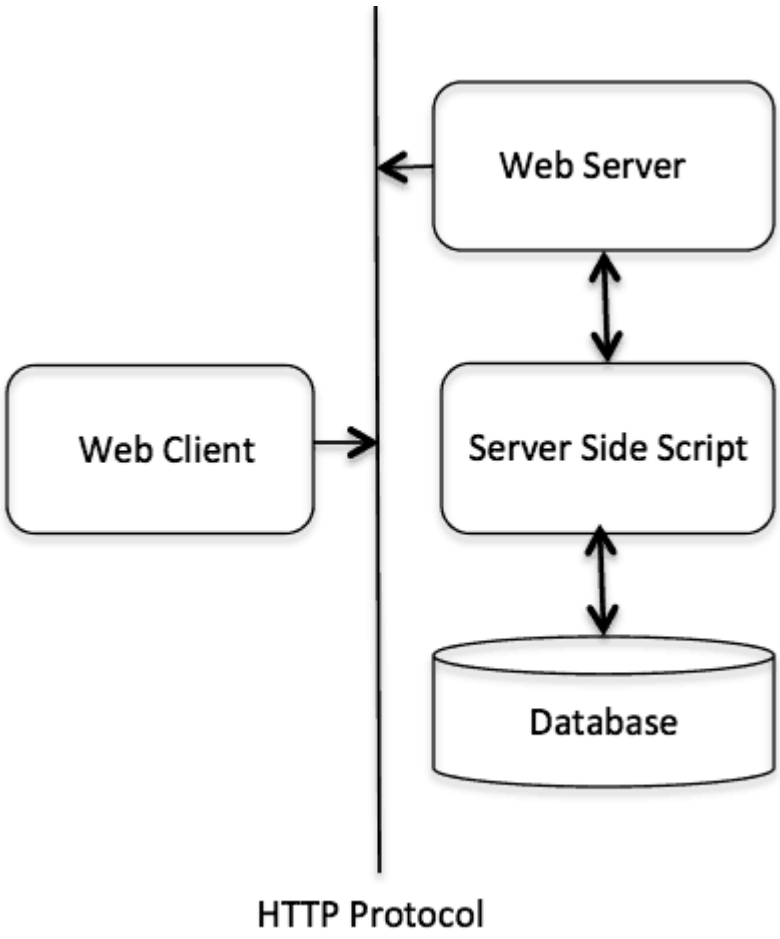
There are three basic features that make HTTP a simple but powerful protocol:

- **HTTP is connectionless:** The HTTP client, i.e., a browser initiates an HTTP request and after a request is made, the client disconnects from the server and waits for a response. The server processes the request and re-establishes the connection with the client to send a response back.
- **HTTP is media independent:** It means, any type of data can be sent by HTTP as long as both the client and the server know how to handle the data content. It is required for the client as well as the server to specify the content type using appropriate MIME-type.
- **HTTP is stateless:** As mentioned above, HTTP is connectionless and it is a direct result of HTTP being a stateless protocol. The server and client are aware of each other only during a current request. Afterwards, both of them forget about each other. Due to this nature of the protocol, neither the client nor the browser can retain information between different requests across the web pages.

HTTP/1.0 uses a new connection for each request/response exchange, whereas HTTP/1.1 connection may be used for one or more request/response exchanges.

Basic Architecture

The following diagram shows a very basic architecture of a web application and depicts where HTTP sits:



The HTTP protocol is a request/response protocol based on the client/server based architecture where web browsers, robots and search engines, etc. act like HTTP clients, and the Web server acts as a server.

Client

The HTTP client sends a request to the server in the form of a request method, URI, and protocol version, followed by a MIME-like message containing request modifiers, client information, and possible body content over a TCP/IP connection.

Server

The HTTP server responds with a status line, including the message's protocol version and a success or error code, followed by a MIME-like message containing server information, entity meta-information, and possible entity-body content.

2. PARAMETERS

This chapter is going to list down few of the important HTTP Protocol Parameters and their syntax the way they are used in the communication. For example, format for date, format of URL, etc. This will help you in constructing your request and response messages while writing HTTP client or server programs. You will see the complete usage of these parameters in subsequent chapters while learning the message structure for HTTP requests and responses.

HTTP Version

HTTP uses a **<major>.<minor>** numbering scheme to indicate versions of the protocol. The version of an HTTP message is indicated by an HTTP-Version field in the first line. Here is the general syntax of specifying HTTP version number:

```
HTTP-Version   = "HTTP" "/" 1*DIGIT "." 1*DIGIT
```

Example

```
HTTP/1.0


or


HTTP/1.1
```

Uniform Resource Identifiers

Uniform Resource Identifiers (URI) are simply formatted, case-insensitive string containing name, location, etc. to identify a resource, for example, a website, a web service, etc. A general syntax of URI used for HTTP is as follows:

```
URI = "http:" "//" host [ ":" port ] [ abs_path [ "?" query ]]
```

Here if the **port** is empty or not given, port 80 is assumed for HTTP and an empty **abs_path** is equivalent to an **abs_path** of "/". The characters other than those in the **reserved** and **unsafe** sets are equivalent to their ""%" HEX HEX" encoding.

Example

The following three URIs are equivalent:

```
http://abc.com:80/~smith/home.html
http://ABC.com/%7Esmith/home.html
http://ABC.com:/%7esmith/home.html
```

Date/Time Formats

All HTTP date/time stamps MUST be represented in Greenwich Mean Time (GMT), without exception. HTTP applications are allowed to use any of the following three representations of date/time stamps:

```
Sun, 06 Nov 1994 08:49:37 GMT  ; RFC 822, updated by RFC 1123
Sunday, 06-Nov-94 08:49:37 GMT ; RFC 850, obsoleted by RFC 1036
Sun Nov  6 08:49:37 1994       ; ANSI C's asctime() format
```

Character Sets

We use character sets to specify the character sets that the client prefers. Multiple character sets can be listed separated by commas. If a value is not specified, the default is the US-ASCII.

Example

Following are the valid character sets:

```
US-ASCII


or


ISO-8859-1


or
```

```
ISO-8859-7
```

Content Encodings

A content encoding value indicates that an encoding algorithm has been used to encode the content before passing it over the network. Content coding are primarily used to allow a document to be compressed or otherwise usefully transformed without losing the identity.

All content-coding values are case-insensitive. HTTP/1.1 uses content-coding values in the Accept-Encoding and Content-Encoding header fields which we will see in the subsequent chapters.

Example

Following are the valid encoding schemes:

```
Accept-encoding: gzip

or

Accept-encoding: compress

or

Accept-encoding: deflate
```

Media Types

HTTP uses Internet Media Types in the **Content-Type** and **Accept** header fields in order to provide open and extensible data typing and type negotiation. All the Media-type values are registered with the Internet Assigned Number Authority (IANA). The general syntax to specify media type is as follows:

```
media-type     = type "/" subtype *( ";" parameter )
```

The type, subtype, and parameter attribute names are case-insensitive.

Example

```
Accept: image/gif
```

Language Tags

HTTP uses language tags within the **Accept-Language** and **Content-Language** fields. A language tag is composed of one or more parts: a primary language tag and a possibly empty series of subtags:

```
language-tag  = primary-tag *( "-" subtag )
```

Whitespaces are not allowed within the tags and all tags are case-insensitive.

Example

Example tags include:

```
 en, en-US, en-cockney, i-cherokee, x-pig-latin
```

where any two-letter primary-tag is an ISO-639 language abbreviation and any two-letter initial subtag is an ISO-3166 country code.

3. MESSAGES

HTTP is based on the client-server architecture model and a stateless request/response protocol that operates by exchanging messages across a reliable TCP/IP connection.

An HTTP "client" is a program (Web browser or any other client) that establishes a connection to a server for the purpose of sending one or more HTTP request messages. An HTTP "server" is a program (generally a web server like Apache Web Server or Internet Information Services IIS, etc.) that accepts connections in order to serve HTTP requests by sending HTTP response messages.

HTTP makes use of the Uniform Resource Identifier (URI) to identify a given resource and to establish a connection. Once the connection is established, **HTTP messages** are passed in a format similar to that used by the Internet mail [RFC5322] and the Multipurpose Internet Mail Extensions (MIME) [RFC2045]. These messages include **requests** from client to server and **responses** from server to client which will have the following format:

```
HTTP-message   = <Request> | <Response> ; HTTP/1.1 messages
```

HTTP requests and HTTP responses use a generic message format of RFC 822 for transferring the required data. This generic message format consists of the following four items.

```
A Start-line

Zero or more header fields followed by CRLF

An empty line (i.e., a line with nothing preceding the CRLF) indicating the
end of the header fields

Optionally a message-body
```

In the following sections, we will explain each of the entities used in an HTTP message.

Message Start-Line

A start-line will have the following generic syntax:

```
start-line = Request-Line | Status-Line
```

We will discuss Request-Line and Status-Line while discussing HTTP Request and HTTP Response messages respectively. For now, let's see the examples of start line in case of request and response:

```
GET /hello.htm HTTP/1.1     (This is Request-Line sent by the client)


HTTP/1.1 200 OK             (This is Status-Line sent by the server)
```

Header Fields

HTTP header fields provide required information about the request or response, or about the object sent in the message body. There are four types of HTTP message headers:

- **General-header:** These header fields have general applicability for both request and response messages.
- **Request-header:** These header fields have applicability only for request messages.
- **Response-header:** These header fields have applicability only for response messages.
- **Entity-header:** These header fields define meta-information about the entity-body or, if no body is present, about the resource identified by the request.

All the above-mentioned headers follow the same generic format and each of the header field consists of a name followed by a colon (**:**) and the field value as follows:

```
message-header = field-name ":" [ field-value ]
```

Following are the examples of various header fields:

```
User-Agent: curl/7.16.3 libcurl/7.16.3 OpenSSL/0.9.7l zlib/1.2.3

Host: www.example.com

Accept-Language: en, mi

Date: Mon, 27 Jul 2009 12:28:53 GMT
```

```
Server: Apache
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
ETag: "34aa387-d-1568eb00"
Accept-Ranges: bytes
Content-Length: 51
Vary: Accept-Encoding
Content-Type: text/plain
```

Message Body

The message body part is optional for an HTTP message but if it is available, then it is used to carry the entity-body associated with the request or response. If entity body is associated, then usually **Content-Type** and **Content-Length** headers lines specify the nature of the body associated.

A message body is the one which carries the actual HTTP request data (including form data and uploaded, etc.) and HTTP response data from the server (including files, images, etc.). Shown below is the simple content of a message body:

```
<html>
<body>
<h1>Hello, World!</h1>
</body>
</html>
```

4. REQUESTS

An HTTP client sends an HTTP request to a server in the form of a request message which includes the following format:

```
A Request-line

Zero or more header (General|Request|Entity) fields followed by CRLF

An empty line (i.e., a line with nothing preceding the CRLF) indicating the
end of the header fields

Optionally a message-body
```

The following sections explain each of the entities used in an HTTP request message.

Request-Line

The Request-Line begins with a method token, followed by the Request-URI and the protocol version, and ending with CRLF. The elements are separated by space SP characters.

```
Request-Line = Method SP Request-URI SP HTTP-Version CRLF
```

Let's discuss each of the parts mentioned in the Request-Line.

Request Method

The **request method** indicates the method to be performed on the resource identified by the given **Request-URI**. The method is case-sensitive and should always be mentioned in uppercase. The following table lists all the supported methods in HTTP/1.1.