MI3103
Pengenalan Protokol HTTP

Fadjar Fathurrahman

2018

Daftar Isi

1 Overview

HTTP adalah singkatan dari Hypertext Transfer Protocol. HTTP merupakan protokol jaringan tingkat aplikasi untuk sistem informasi terdistribusi, kolaboratif, dan hypermedia. Protokol ini merupakan fondasi bagi komunikasi data untuk World Wide Web (W3, internet) sejak tahun 1990. HTTP juga dapat digunakan untuk keperluan selain World Wide Web. HTTP dispesifikasikan dalam RFC-2616, yang mendefinisikan protokol yang dikenal sekarang sebagai HTTP/1.1, yang merupakan revisi dari HTTP awal (HTTP/1.0). Perbedaan penting antara HTTP/1.1 dan HTTP/1.0 adalah HTTP/1.0 menggunakan koneksi baru untuk tiap pertukaran permintaan (*request*) dan respon (*response*), sedangkan pada HTTP/1.1 satu koneksi dapat digunakan untuk pertukaran lebih dari satu *request/response*.

HTTP adalah protokol komunikasi yang berbasis TCP/IP, yang digunakan untuk mengirimkan data seperti file HTML, CSS, Javascript, gambar, dan sebagainya melalui Web. Port default untuk protokol TCP adalah 80, meskipun port lain juga dapat digunakan. HTTP menyediakan cara standard bagi komputer untuk berkomunikasi satu dengan yang lainnya. HTTP menspesifikasikan bagaimanan data permintaan dari klien dibangung dan dikirimkan ke server, serta bagaimana server merespon permintaan tersebut.

Fitur dasar HTTP:

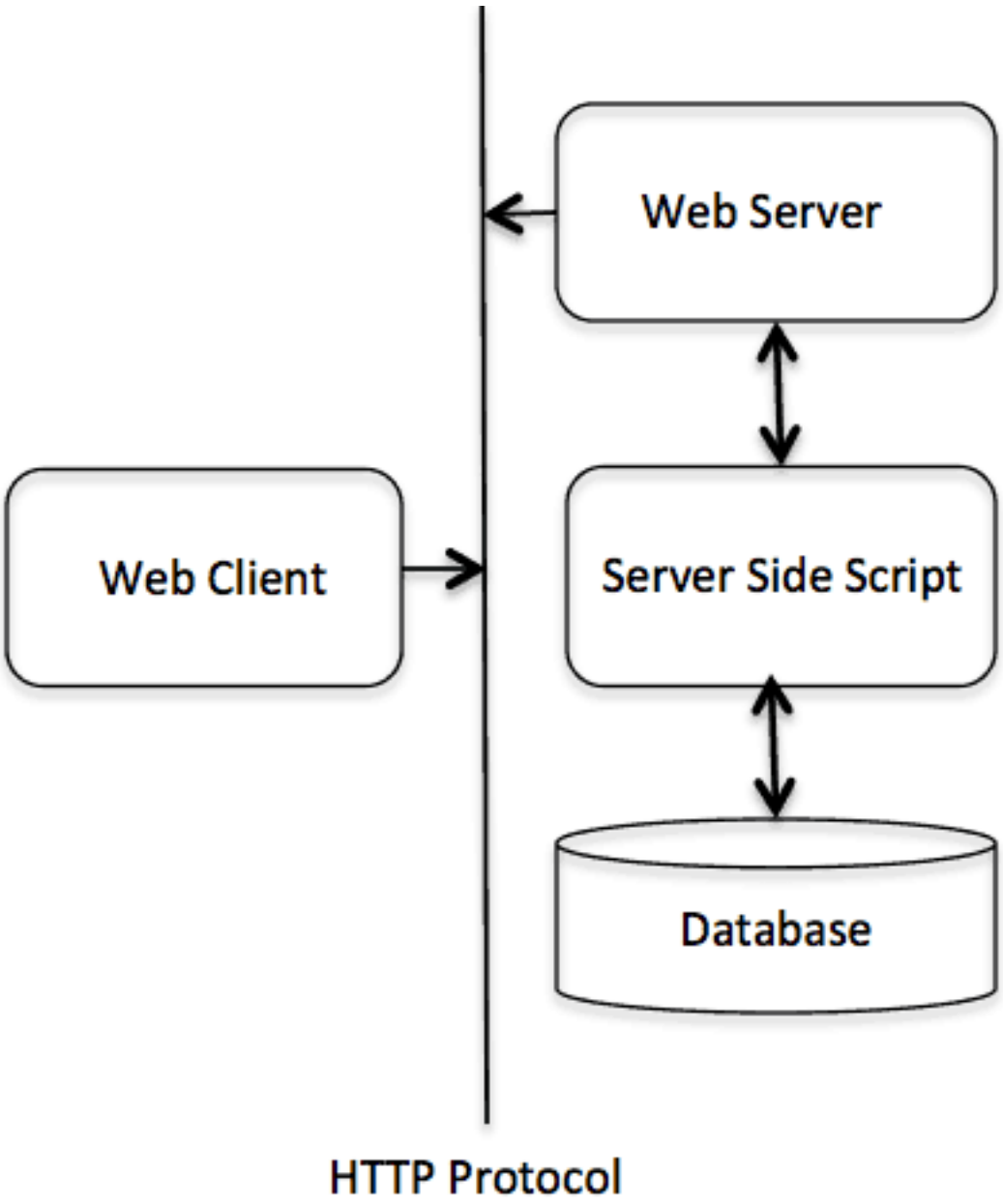
- Connectionless: Klien HTTP menginisiasi permintaan HTTP dan setelah itu klien melepas koneksi dengan server dan menunggu respon dari server. Proses server menerima permintaan dan membangun kembali koneksi dengan klien untuk mengirimkan respon.
- Media-independent: tipe data apapun dapat dikirimkan melalui HTTP asalkan klien dan server mengetahui bagaimana cara menangani data tersebut.
- 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, where as 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:

HTTP Architecture



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 HTTP 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.

2.1 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
```

2.2 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.

Examples: 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
```

2.3 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
```

2.4 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
- ISO-8859-1
- ISO-8859-7

2.5 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.

Examples: The following are valid encoding schemes:

```
Accept-encoding: gzip
Accept-encoding: compress
Accept-encoding: deflate
```

2.6 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
```

2.7 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 )
```

White spaces are not allowed within the tag and all tags are case-insensitive. 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 HTTP - 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.

3.1 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)
```

3.2 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
```

3.3 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>
```

Next two chapters will make use of above explained concepts to prepare HTTP Requests and HTTP Responses.

4 HTTP - Requests

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

4.1 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.

4.2 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 lists all the supported methods in HTTP/1.1.

1. GET: The GET method is used to retrieve information from the given server using a given URI. Requests using GET should only retrieve data and should have no other effect on the data.
2. HEAD: Same as GET, but it transfers the status line and the header section only.
3. POST: A POST request is used to send data to the server, for example, customer information, file upload, etc. using HTML forms.
4. PUT: Replaces all the current representations of the target resource with the uploaded content.
5. DELETE: Removes all the current representations of the target resource given by URI.
6. CONNECT: Establishes a tunnel to the server identified by a given URI.
7. OPTIONS: Describe the communication options for the target resource.
8. TRACE: Performs a message loop back test along with the path to the target resource.

4.3 Request-URI

The Request-URI is a Uniform Resource Identifier and identifies the resource upon which to apply the request. Following are the most commonly used forms to specify an URI:

```
Request-URI = "*" | absoluteURI | abs_path | authority
```

The asterisk * is used when an HTTP request does not apply to a particular resource, but to the server itself, and is only allowed when the method used does not necessarily apply to a resource. For example:

```
OPTIONS * HTTP/1.1
```

The `absoluteURI` is used when an HTTP request is being made to a proxy. The proxy is requested to forward the request or service from a valid cache, and return the response. For example:

```
GET http://www.w3.org/pub/WWW/TheProject.html HTTP/1.1
```

The most common form of `Request-URI` is that used to identify a resource on an origin server or gateway. For example, a client wishing to retrieve a resource directly from the origin server would create a TCP connection to port 80 of the host `www.w3.org` and send the following lines:

```
GET /pub/WWW/TheProject.html HTTP/1.1
Host: www.w3.org
```

Note that the absolute path cannot be empty; if none is present in the original URI, it MUST be given as "/" (the server root).

4.4 Request Header Fields

We will study General-header and Entity-header in a separate chapter when we will learn HTTP header fields. For now, let's check what Request header fields are.

The request-header fields allow the client to pass additional information about the request, and about the client itself, to the server. These fields act as request modifiers.Here is a list of some important Request-header fields that can be used based on the requirement:

- Accept-Charset
- Accept-Encoding
- Accept-Language
- Authorization
- Expect
- From
- Host
- If-Match
- If-Modified-Since
- If-None-Match
- If-Range
- If-Unmodified-Since
- Max-Forwards
- Proxy-Authorization
- Range
- Referer
- TE
- User-Agent

You can introduce your custom fields in case you are going to write your own custom Client and Web Server.

4.5 Examples of Request Message

Now let's put it all together to form an HTTP request to fetch `hello.htm` page from the web server running on `tutorialspoint.com`

```
GET /hello.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

Here we are not sending any request data to the server because we are fetching a plain HTML page from the server. Connection is a general-header, and the rest of the headers are request headers. The following example shows how to send form data to the server using request message body:

```
POST /cgi-bin/process.cgi HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Content-Type: application/x-www-form-urlencoded
Content-Length: length
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive

licenseID=string&content=string&/paramsXML=string
```

Here the given URL `/cgi-bin/process.cgi` will be used to process the passed data and accordingly, a response will be returned. Here content-type tells the server that the passed data is a simple web form data and length will be the actual length of the data put in the message body. The following example shows how you can pass plain XML to your web server:

```
POST /cgi-bin/process.cgi HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Content-Type: text/xml; charset=utf-8
Content-Length: length
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive

<?xml version="1.0" encoding="utf-8"?>
<string xmlns="http://clearforest.com/">string</string>
```

5 HTTP - Responses

After receiving and interpreting a request message, a server responds with an HTTP response message:

```
    A Status-line

    Zero or more header (General|Response|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
```

5.1 Message Status-Line

A Status-Line consists of the protocol version followed by a numeric status code and its associated textual phrase. The elements are separated by space SP characters.

```
Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase CRLF
```

5.2 HTTP Version

A server supporting HTTP version 1.1 will return the following version information:

```
HTTP-Version = HTTP/1.1
```

5.3 Status Code

The Status-Code element is a 3-digit integer where first digit of the Status-Code defines the class of response and the last two digits do not have any categorization role. There are 5 values for the first digit:

- 1xx: Informational. It means the request was received and the process is continuing.
- 2xx: Success. It means the action was successfully received, understood, and accepted.
- 3xx: Redirection. It means further action must be taken in order to complete the request.
- 4xx: Client Error. It means the request contains incorrect syntax or cannot be fulfilled.
- 5xx: Server Error. It means the server failed to fulfill an apparently valid request.

HTTP status codes are extensible and HTTP applications are not required to understand the meaning of all registered status codes. A list of all the status codes has been given in a separate chapter for your reference.

5.4 Response Header Fields

We will study General-header and Entity-header in a separate chapter when we will learn HTTP header fields. For now, let's check what Response header fields are.

The response-header fields allow the server to pass additional information about the response which cannot be placed in the Status-Line. These header fields give information about the server and about further access to the resource identified by the Request-URI.

- Accept-Ranges
- Age
- ETag
- Location
- Proxy-Authenticate
- Retry-After
- Server
- Vary
- WWW-Authenticate

You can introduce your custom fields in case you are going to write your own custom Web Client and Server.

5.5 Examples of Response Message

Now let's put it all together to form an HTTP response for a request to fetch the `hello.htm` page from the web server running on `tutorialspoint.com`

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
Content-Length: 88
Content-Type: text/html
Connection: Closed

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

The following example shows an HTTP response message displaying error condition when the web server could not find the requested page:

```
HTTP/1.1 404 Not Found
Date: Sun, 18 Oct 2012 10:36:20 GMT
Server: Apache/2.2.14 (Win32)
Content-Length: 230
Connection: Closed
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html>
<head>
   <title>404 Not Found</title>
</head>
<body>
   <h1>Not Found</h1>
   <p>The requested URL /t.html was not found on this server.</p>
</body>
</html>
```

Following is an example of HTTP response message showing error condition when the web server encountered a wrong HTTP version in the given HTTP request:

```
HTTP/1.1 400 Bad Request
Date: Sun, 18 Oct 2012 10:36:20 GMT
Server: Apache/2.2.14 (Win32)
Content-Length: 230
Content-Type: text/html; charset=iso-8859-1
Connection: Closed

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html>
<head>
   <title>400 Bad Request</title>
</head>
<body>
   <h1>Bad Request</h1>
   <p>Your browser sent a request that this server could not understand.</p>
   <p>The request line contained invalid characters following the protocol string.</p>
</body>
</html>
```