HTTP://
Little Book

It May help you save a lot of time

Liu Chuanjun

HTTP A Little Book

Reco

This book is for sale at http://leanpub.com/httplittlebook

This version was published on 2018-08-13

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Contents

Foreword

Today, more and more software is built on HTTP protocols. And I feel its value bit by bit in my several software project.So I think that it will help improve software development efficiency if I master it fully and systematic. At the beginning of this year, I've dicided to research it fully. I bought a set of books, put them in my car, my office and my home. I'v been trying to made anything about HTTP clear,whether it is useful or not in my work currently,and if any text is not clear enough to understand ,I will digging into source code of Nodejs's HTTP module or ask in `Stackoverflow.com`.The more I study it, the clearer my architecture of HTTP is.

One day, I heard a voice saying to myself, "it must be a interesting thing to write something about it",because:

1. It used by a lot of people
2. Existing text is too complicated to understand

HTTP prototols was originly very simple, but if you follow the strict text of the RFC,it will seem to be very diffcult. So, I can write a book with Best-First princle.

In the first chapter to I would introduce the HTTP architecture through the sample code. In the following chapters, I will break down the HTTP protocols into small part,including request and response processes. Against something are particularly difficult, there would be a few command lines and a small piece of Node.js source code. So, this is an HTTP book, written by a programmer,and written for programmers. I hope this book looks straightforward enough to help you save a lot of time for study it.

Now it is in front of you.

Introduction

If we know that a website has a page which url is `http://example.com/hello.htm`, then I can open a browser in my computer and typing this url,after several seconds ,you will see a HTML page rendered in the browser.

Under such a great representation, if we want to know the interactive dialogue between the browser and the server underhood, we must to learn the HTTP protocol. It is this protocol that specifies how to package a client's request as an HTTP request message and send it to the server. This protocol also specifies that a response from server is packaged into an HTTP response message and sent back to the browser.

In more detail ,the browser software opens the server connection and send those text:

```
1 GET /hello.htm HTTP/1.1
2 User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
3 Host: example.com
4 Accept-Language: en-us
5 Accept-Encoding: gzip, deflate
6 Connection: Keep-Alive
```

After receiving this request message from client, by parsing the first line ,the server can know that the browser has sent a GET request and wants the hello.htm resource in the root directory. The protocol version is 1.1.

The server can also obtain more client information according to the information called the header fields between the second line and the blank line. For example, if the server parse this line `Accept-Language:en-us`, the server would know the browser can accept the response which content language is english. Then the

server would find this resource in specified path and gives a response:

```
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>
```

Then the browser received this response, parse the first line, which indicates the protocol version is 1.1, and parse the status code is 200, which indicates the request is successful. and parsing form the second line to the blank line, get more message metadata, such as `Content-Type: text/html` and `Content-Length: 88`, indicates that the body is carrying an html file, and the file's length is 88 bytes. Then get the contents of the HTML file after the first blanket line, finally ,the browser render the HTML file to the user.

And according to the head line of `Connection: Closed`, which indicated the connection should be closed after the browser have been received the whole response content.

Request message

For general purposes, a request message is constructed in this way:

- A request line. The first line of the request message is the request line. It indicates the request method, resource identifier, and HTTP version used.
- The header fields.

A zero or more line header field followed by a CRLF. In the case

```
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: example.com
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

All of them are called the header fields. The header field is separated into two parts by a colon, the left side is called the header field name, and the right side is called the header field value. For example, `Host: example.com`, the value of the header field `Host` is `example.com`. The name of the header field is in a very long list, and their values are also different.

- A blank line (CRLF). Indication header field completed
- Optional message body. in this case it has no message body.

The request method may be GET,as well as POST, HEAD, PUT, DELETE, CONNECT, OPTIONS, TRACE.

GET means that the browser want to request a resource with the specified URL. POST means that the browser want to update a resource, the new data of the resource is provided by the message subject. PUT means that the browser want to create a resource with the specified URL. The new data for the resource is provided by the message subject. DELETE indicates that the browser want to delete a resource with the specified URL.

the other request methods ,such as CONNECT,OPTIONS,TRACE will be explained in later chapters.

Because the request method is the most critical message field, it is very convenient to classify the request message directly according to the request method. In this case, the message using the GET method is directly reduced to a GET request; accordingly, there are naturally POST requests, PUT requests, and DELETE requests. and many more.

Response Message

Response message is composited as follows:

- A status line. It consists of a protocol version, a status code and status description text. Such as `HTTP/1.1 200 OK`. Status code 200 indicates success.
- A blank line (CRLF). this blank line indicates header field section is complete. In this case, follow lines are header fields:

  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
- Optional message body. In the case which is the content of `hello.htm` file

There are 5 groups of status codes, which are in the range of 100-199, 200-299, 300-399, 400-499, 500-599.

- 200-299 successful. Indicates that the client request is correct and was successfully executed.
- 300-399 redirect. Indicates that the client request is correct, but the location of the current request resource is elsewhere. Please redirect your resource location again to initiate a new request.

- 400-499 Client side error. Indicates that the client's request is incorrect, either the format is unrecognizable, or the URL is too long, and so on.
- 500-599 server side error. Indicates that the client's request is correct, but the server was unable to complete the request for its own reasons.
- 100-199 Information alert. There are only 2 status codes in this series, but they are more confusing. It will be specifically explained late.

Because the status code is the most critical response message field, selecting different status codes often means different header fields and body, so it is very convenient to classify the response messages directly according to the status code. In this case, the status code is a series of 100 response messages, which can be simplified to a type 100 response. Correspondingly, there are naturally type 200 responses, type 300 responses, type 400 responses, and type 500 responses.

Hands-on

Create a web server using Node.js. Initiate the raw GET request via Netcat and review the raw response.

Server side:

```
var express = require('express');
var app = express();
app.get('/', function (req, res) {
  res.send('<a href="/test204">204</a> <a href="/test205"\
>205</a> <a href="/test300">300</a>');
});
var server = app.listen(3000, function () {
  var host = server.address().address;
  var port = server.address().port;
  console.log('Example app listening at http://%s:%s', ho\
st, port);
});
```

Execute node app.js and enter the row request on console ,then to see it's raw response:

```
$ nc localhost 3000
GET /

HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 80
ETag: W/"50-41mmSL16PW+Zt5VLKLE2/Q"
Date: Thu, 03 Dec 2015 08:54:23 GMT
Connection: close
```

or get HTTP/1.0,then view it's response:

```
$ nc localhost 3000
GET / HTTP/1.0

HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 80
ETag: W/"50-41mmSL16PW+Zt5VLKLE2/Q"
Date: Thu, 03 Dec 2015 08:52:45 GMT
Connection: close

<a href="/test204">204</a> <a href="/test205">205</a> <a \
href="/test300">300</a>
```

Hands-on environment

As a programmer,I'd like to type some code to verify what you have learned.Sometimes, code can express confusing concepts better than descriptive text.So, I will use some of source code to express it here.

The language I chose was Node.js because it was small and expressive,and it was easy to write some code and run it in a few of steps.

I often work remotely, when I am using a Mac Book Air,if I work inside my company, I used a Windows PC. The cross-platform feature of Node.js allows me to switch environments freely without being annoyed by detail differences between operating systems.

if using command-line tools can do a bunch of things directly,and I will use it. For example, Netcat is a universal saber of HTTP protocol.I will use it to test my web server.

Node.js and Express.js

I uses Node.js with Express.js as the web server development environment.You need to install Node.js ,then install Express.js with the npm command to run the code which I gave.

Installing Node is different on different platforms, but if you like installer ,Whether it's Windows or Mac, you can install it by click a button which name is `next` a few of times. Once Node.js is installed, npm is a built-in Node package management tool, you can install Express.js by this command :

```
1 $npm i express
```

then you will prepared your environment.

Netcat

I'd like do many jobs by the command line, not by the graphical user interface. So when I must make HTTP test, I would prefer Netcat than browsers. This is also good for the reader, because it can reduce the burden of eyeball recognition - the command is to minimize the output of the information, and the use of a graphical browser, requires a large number of screenshots which are almost inevitably contains a lot of information that is not related to the current problem.

Therefore, you need to know this command and other small command line tools. Of course, they are very simple usually.Here you can find more information about netcat http://nc110.sourceforge.net