Technical University of Moldova
Faculty of Computers and Information Technologies
Specialty Computer Science

Report

Laboratory #5

HTTP protocol: mean of distributed data transmission

Author:
Cristian Cartofeanu

Lecturer:
Dumitru Ciorbă

January 21, 2016

Contents

1 Objective

The goal of the laboratory study lies on the HTTP protocol in the context of data distribution and the use of HTTP methods in implementing the interaction between the client and the server applications.

The primary objectives:

- Creation of a server with concurent processing of HTTP requests
- Implement GET/POST methods for processing the requests

2 HTTP protocols

The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web.
Hypertext is structured text that uses logical links (hyperlinks) between nodes containing text. HTTP is the protocol to exchange or transfer hypertext.
The standards development of HTTP was coordinated by the Internet Engineering Task Force (IETF) and the World Wide Web Consortium (W3C), culminating in the publication of a series of Requests for Comments (RFCs). The first definition of HTTP/1.1, the version of HTTP in common use, occurred in RFC 2068 in 1997, although this was obsoleted by RFC 2616 in 1999.

2.1 HTTP session

An HTTP session is a sequence of network request-response transactions. An HTTP client initiates a request by establishing a Transmission Control Protocol (TCP) connection to a particular port on a server (typically port 80, occasionally port 8080; see List of TCP and UDP port numbers). An HTTP server listening on that port waits for a client's request message. Upon receiving the request, the server sends back a status line, such as "HTTP/1.1 200 OK", and a message of its own. The body of this message is typically the requested resource, although an error message or other information may also be returned.

2.2 Request methods

HTTP defines methods (sometimes referred to as verbs) to indicate the desired action to be performed on the identified resource. What this resource represents, whether pre-existing data or data that is generated dynamically, depends on the implementation of the server. Often, the resource corresponds to a file or the output of an executable residing on the server. The HTTP/1.0 specification defined the GET, POST and HEAD methods and the HTTP/1.1 specification added 5 new methods: OPTIONS, PUT, DELETE, TRACE and CONNECT. By being specified in these documents their semantics are well known and can

be depended upon. Any client can use any method and the server can be configured to support any combination of methods. If a method is unknown to an intermediate it will be treated as an unsafe and non-idempotent method. There is no limit to the number of methods that can be defined and this allows for future methods to be specified without breaking existing infrastructure.
We focused on 2 particular methods:

- **GET** - The GET method requests a representation of the specified resource. Requests using GET should only retrieve data and should have no other effect. (This is also true of some other HTTP methods.) The W3C has published guidance principles on this distinction, saying, "Web application design should be informed by the above principles, but also by the relevant limitations."
- **POST** - The POST method requests that the server accept the entity enclosed in the request as a new subordinate of the web resource identified by the URI. The data POSTed might be, for example, an annotation for existing resources; a message for a bulletin board, newsgroup, mailing list, or comment thread; a block of data that is the result of submitting a web form to a data-handling process; or an item to add to a database.

3 Implementation description

In order to run our application we will need to install several tools for that:

- NodeJS as a JavaScript environment.
- Node.js modules: http; dispatcher; underscore; url;

In this practical inplementation I will be talking about how we can set up a simple Node.js HTTP server from a beginner's perspective. Node is a fantastic candidate for creating web servers, which are lightweight and can handle a great number of simultaneous requests. This means if youre interested in learning to build scalable web applications this is a great beginner's guide.
Loading the http Module
Node.js is shipped with several core modules out of the box, which we will be using to set up our own http server. The http module makes it simple to create an http server via its simple but powerful api.

```
//How to require/import the HTTP module
var http = require('http');
```

Defining the Handler Function We need a function which will handle all requests and reply accordingly. This is the point of entry for your server application, you can reply to requests as per your business logic.

```
//We need a function which handles requests and send response
function handleRequest(request, response){
    response.end('It Works!! Path Hit: ' + request.url);
}
```

Creating and Starting the Server

```
//Create a server
var server = http.createServer(handleRequest);

//Lets start our server
server.listen(PORT, function(){
    //Callback triggered when server is successfully listening.
    console.log("Server listening on: http://localhost:%s", PORT);
        });
```

Adding in a Dispatcher Now that we have a basic HTTP Server running, it's time we implement some real functionality. Your server should respond differently to different URL paths. This means we need a dispatcher. Dispatcher is kind of router which helps in calling the desired request handler code for each particular URL path. Now lets add a dispatcher to our program. First we will install a dispatcher module, in our case httpdispatcher. There are many modules available but lets install a basic one for demo purposes:
If youre unaware, npm is a package manager which provides a central repository for custom open sourced modules for Node.js and JavaScript. npm makes it simple to manage modules, their versions and distribution. We used the 'npm install' command to install the required module in our project.
Now we will require the dispatcher in our program, add the following line on the top:

```
> npm install httpdispatcher
```

If youre unaware, npm is a package manager which provides a central repository for custom open sourced modules for Node.js and JavaScript. npm makes it simple to manage modules, their versions and distribution. We used the 'npm install' command to install the required module in our project.
For importing the dispatcher module just add the following line on the top:

```
var dispatcher = require('httpdispatcher');
```

Here is how we use our dispatcher in our handleRequest function:

```
//Lets use our dispatcher
function handleRequest(request, response){
    try {
        //log the request on console
        console.log(request.url);
        //Disptach
        dispatcher.dispatch(request, response);
    } catch(err) {
```

```
        console.log(err);
    }
}
```

Here is how we define some routes. Routes define what should happen when a specific URL is requested through the browser (such as /employee or /employees).

```
        //For all your static (js/css/images/etc.)
        //set the directory name        (relative path).
        dispatcher.setStatic('resources');

        //A sample GET request
        dispatcher.onGet("/page1", function(req, res) {
        res.writeHead(200, {'Content-Type': 'text/plain'});
        res.end('Page One');
        });

        //A sample POST request
        dispatcher.onPost("/post1", function(req, res) {
        res.writeHead(200, {'Content-Type': 'text/plain'});
        res.end('Got Post Data');
        });
```

Before running our server we also need to create some nodes that will serve us by sending post request to our server. Here again we will be using `http` module. Now lets run what we have to see some Node's magic. First we need to start our http server an after we will gona start the nodes that will post some information in our server using post method. After we can try the following URL paths to extract some specific informationinformation:

- `http://localhost:8080/employee?id=1`
- `http://localhost:8080/employees`
- `http://localhost:8080/employees?offset=1&limit=4`

You can use a browser to do a GET request just by entering the URL in the address bar or a tool like Postman that also permits post request.

4 Conclusion

During this laboratory work I have learned about http protocol in detail while working around with GET/POST method for data processing.

5 References

[2] Casciaro Mario, Node.js Design Patterns, 2014, Packt Publishing, ISBN-13: 978-1783287314

[3] Rohit Rai., Socket.IO Real-time Web Application Development, 2013, Packt Publishing, ISBN: 978-1-78216-078-6, p. 50

[4] https://github.com/CristianChris/HTTP-protocol-mean-of-distributed-data-transmission