**JavaScript Basics #9: Network and HTTP**

If you’ve made it to this article, congratulations, we are done with the frontend basics of JavaScript. It is almost time for us to dive into the backend of web development. However, before we do that, I’d like to first talk about how the internet actually works.

The network, to put it simply, is multiple computers connected together. These computers can send information to each other. And if this network extends to the entire planet, it becomes what we call the internet.

Table of Contents

* [Network Protocols](https://www.ericsdevblog.com/index.php/2021/11/javascript-basics-9-network-and-http/#Network_Protocols)
* [HTTP Methods](https://www.ericsdevblog.com/index.php/2021/11/javascript-basics-9-network-and-http/#HTTP_Methods)
  + [The GET Method](https://www.ericsdevblog.com/index.php/2021/11/javascript-basics-9-network-and-http/#The_GET_Method)
  + [The POST Method](https://www.ericsdevblog.com/index.php/2021/11/javascript-basics-9-network-and-http/#The_POST_Method)
  + [The DELETE Method](https://www.ericsdevblog.com/index.php/2021/11/javascript-basics-9-network-and-http/#The_DELETE_Method)
  + [The HEAD Method](https://www.ericsdevblog.com/index.php/2021/11/javascript-basics-9-network-and-http/#The_HEAD_Method)
  + [THE PUT Method](https://www.ericsdevblog.com/index.php/2021/11/javascript-basics-9-network-and-http/#THE_PUT_Method)
* [HTML Form and HTTP](https://www.ericsdevblog.com/index.php/2021/11/javascript-basics-9-network-and-http/#HTML_Form_and_HTTP)
* [JavaScript and HTTP](https://www.ericsdevblog.com/index.php/2021/11/javascript-basics-9-network-and-http/#JavaScript_and_HTTP)

**NETWORK PROTOCOLS**

When a computer (server) sends data and resources, it has to follow a certain protocol, so that the computer (client) that receives the resources knows how to read them. There are protocols for sending and receiving emails, sharing files, and even controlling another computer over the internet. We don’t have time to introduce all of them, so in this article, we’ll focus on HTTP, HTTPS the TCP protocol.

The TCP protocol is one of the most commonly used internet communication protocols, in fact, a lot of other protocols are created on top of it. It works as follows: one computer must always be listening, waiting for other computers to start talking to it.

This computer has different “listeners”, and they can listen for different kinds of communications at the same time, to make sure these listeners don’t interrupt each other, each of them will take up one position (port) on that computer. For example, when we receive emails, that email is sent to us using the SMTP protocol, which is created based on the TCP protocol. By default, our computer will always be listening on port 25 for emails.

For another computer to send data to the target computer, it needs to “talk” to the target computer through the correct port. If the target machine can be reached and is listening on that port, a connecting will be established, and the data transfer can begin. In this case, the computer that is listening is called the client, and the computer doing the talking is called the server.

The Hypertext Transfer Protocol (HTTP) is a protocol for retrieving named resources. It means that the client would first make a request to the server, asking for some resources. The resources are usually web pages, images or CSS/JavaScript files. If the server is “OK” with that request, it would return a 200 OK message back to the client, and start transferring the files.

The HTTP request sent by the client usually looks like this:

# Start with HTTP method (we'll discuss this in detail later), followed by the name of

# the resource, and the version of the protocol

GET /index.html HTTP/1.1

# You can also specify other information here

Host: example.com

Accept-Language: en

And the response looks like this:

# Start by the 200 OK message

HTTP/1.1 200 OK

# Some extra info here

Date: Sat, 09 Oct 2010 14:28:02 GMT

Server: Apache

Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT

ETag: "51142bc1-7449-479b075b2891b"

Accept-Ranges: bytes

Content-Length: 29769

Content-Type: text/html

# The requested resource

<!DOCTYPE html... (here come the 29769 bytes of the requested web page)

Of course, when you are surfing the internet, you never actually had to do this manually, the browser does everything automatically for you when you type in the uniform resource locator (URL), which specifies the protocol, host, and the path to the resource you want.

http://example.com/2020/03/16/13\_browser.html

| | | |

protocol server path

The HTTPS protocol works exactly the same, except it is encrypted. It uses something called the transport layer security (TLS) protocol to make sure that the communication between the client and the server is secure. The server has a private key and the client has a public key, the connecting could only be established if the two keys match each other.

**HTTP METHODS**

Since we are focusing on web development, in this article, we’ll only discuss the HTTP protocol in detail. Notice that from our previous example, when we send an HTTP request, the request starts with a keyword GET, which is called an HTTP method. There are six other methods besides GET, and each of them serves a different purpose.

**The GET Method**

The GET method is the most commonly used HTTP request method. It is used to request data and resources from the server. When you send a GET request, the query parameters are sent in the URL in name/value pairs like this:

http://example.com/2020/03/16/13\_browser.html?name1=value1&name2=value2

Note that the question mark (?) marks the beginning of the string of parameters, and the ampersand divides two different parameters.

**The POST Method**

The POST method is used to send data to the server, either adding a new resource or updating an existing resource. The parameters are stored in the body of the HTTP request.

POST /index.html HTTP/1.1

Host: example.com

name1=value1&name2=value2

**The DELETE Method**

This one is very intuitive, it deletes a resource from the server.

**The HEAD Method**

The HEAD method works just like the GET method. Except the HTTP response sent from the server will only contain the head and not the body. Meaning if the server is ok with the request, it will give you a 200 OK response but not the resource you requested. You can only retrieve the resource with the GET method. This is very useful when testing if the server works.

**THE PUT Method**

The PUT method is similar to the POST method, with one small difference. When you POST a resource that already exists on the server, this action would not cause any difference, it would always produce the same result. The PUT method, however, will duplicate that resource, every time you make the request.

**HTML FORM AND HTTP**

Now that we know what an HTTP request would look like, it is time to talk about how to send a request. The most common way of doing that is through HTML forms. It allows the user to fill out Information and submit them as parameters. Here is an example:

<form method="GET" action="example/message.html">

<p>Name: <input type="text" name="name"></p>

<p>Message:<br><textarea name="message"></textarea></p>

<p><button type="submit">Send</button></p>

</form>

Let’s first look at the <form> tag. The method attribute specifies the HTTP method we are going to use. In this case, it’s GET, which means the parameters will be embedded inside the URL. The action specifies the domain and the path to the file we are requesting. Usually, the server will perform some actions to that file based on the parameters you send, and return you a customized file.

If you look inside the <form> element, notice that the user input elements (both <input> and <textarea>) have name attribute. This defines the name of the parameter, which is a name/value pair. The corresponding value of that name would be the user input. This name is very important, you have to make sure that when you are coding the backend, the names are consistent.

When you push the “Send” button, the HTTP request would look like this:

GET /example/message.html?name=Jean&message=Yes%3F HTTP/1.1

**JAVASCRIPT AND HTTP**

Besides HTML forms, JavaScript can also be used to send HTTP request. It can be done using the fetch() method like this:

fetch("path/to/resource").then((response) => {

// Get the returned response status (200 OK)

console.log(response.status);

// Get the header of the response

console.log(response.headers.get("Content-Type"));

});

By default, the fetch() method uses GET method to make the request, you can change that by specifying the method.

fetch("path/to/resource", {method: "POST"}).then(...);

Or adding extra information in the header, and add parameters in the body like this:

fetch("path/to/resource", {

method: "POST",

headers: {

"Content-type": "application/x-www-form-urlencoded; charset=UTF-8",

},

body: "name1=val1&name2=val2",

}).then(...);

However, using JavaScript to make HTTP request does raise some security concerns. Because the user and the programmer aren’t usually the same person, they might not have the same interest in mind. Obviously, you don’t want a random web page to access your bank with credentials stored in your browser. This is why most browsers forbid JavaScript from making HTTP requests by default.

This can be very annoying because it is possible that the JavaScript code wants to access another domain for a legitimate reason. To solve this problem, the servers can include in the response saying that it is OK for the request to come from another domain.

Access-Control-Allow-Origin: \*