Week 2.2

Created	December 28, 2023 1:57 PM
Created by	A Aditya Kulkarni
Tags	Empty

Node.js runtime | HTTP

Week 2.2

What are we learning today

- What is ECMAScript
- What is Javascript
- What is Node.js
- What is Bun
- Node.js and its runtime
- How to create an HTTP server with Node.js

What is ECMAScript

- ECMAScript is a standard for scripting languages, such as Javascript, that defines the syntax, semantics, and features of the language1.
- ECMAScript is developed by the ECMA International organization, and the latest version is ECMAScript 20211.
- ECMAScript is also known as ES, and each version has a number or a year as a suffix, such as ES6 or ES20151.
- ECMAScript defines the core features of Javascript, such as variables, data types, operators, expressions, statements, functions, objects, arrays, etc2.

What is Javascript

- Javascript is a scripting language that follows the ECMAScript standard, but also has some additional features, such as the `document` and `window` objects, that are specific to web browsers3.
- Javascript is a high-level, interpreted, dynamic, and multi-paradigm language that supports object-oriented, functional, and imperative programming styles3.
- Javascript is mainly used for creating dynamic and interactive web pages, but can also be used for other purposes, such as server-side programming, desktop applications, mobile applications, etc3.
- Javascript is one of the most popular and widely used programming languages in the world, and is supported by all major web browsers3.

What is Node.js

- Node.js is a runtime environment that allows Javascript to run outside of a web browser, on a computer or a server operating system4.

- Node.js is built on the Google V8 Javascript engine, which is a fast and powerful engine that compiles Javascript code to native machine code4.
- Node.js uses a single-threaded, non-blocking, event-driven model, which means that it can handle multiple concurrent requests without waiting for I/O operations to finish4.
- Node.js provides a low-level I/O API, which gives access to the file system, network, streams, buffers, etc4.
- Node.js also has a built-in module system, which allows developers to organize and reuse code, and a large ecosystem of third-party modules, which can be installed using the npm package manager4.
- Node.js is mainly used for creating scalable and performant web applications, such as APIs, microservices, real-time applications, etc4.

What is Bun

- Bun is a new runtime environment that aims to replace Node.js for backend programming with Javascript.
- Bun is written in Zig, which is a fast and safe low-level language that can interoperate with C.
- Bun claims to be significantly faster, more memory-efficient, and more secure than Node.js, by using a different architecture and design.
- Bun is still in development and not yet ready for production use, but it has some promising features, such as native HTTP/3 support, zero-cost abstractions, and automatic concurrency.

Node.js and its runtime

- Node.js is a runtime environment that executes Javascript code on a computer or a server operating system4.
- A runtime environment is a software layer that provides the necessary services and resources for a program to run.
- A runtime environment typically consists of three components: a compiler or interpreter, a library, and an API.
- A compiler or interpreter is a program that translates the source code of a program into executable code that can run on a specific platform.

- A library is a collection of pre-written code that provides common functionality and features for a program.
- An API is a set of rules and specifications that define how a program can interact with other programs or systems.
- Node.js uses the Google V8 Javascript engine as its compiler or interpreter, which converts Javascript code into native machine code4.
- Node.js provides a core library that includes modules for various tasks, such as file system, network, crypto, etc4.
- Node.js also provides an API that exposes the low-level I/O functionality and the event loop mechanism to the Javascript code4.

How to create an HTTP server with Node.js

- HTTP (Hypertext Transfer Protocol) is a protocol that defines how web browsers and web servers communicate over the internet.
- An HTTP server is a program that listens for and responds to HTTP requests from web browsers or other clients.
- Node.js provides a built-in module called `http` that allows creating and running an HTTP server with Javascript.
- To create an HTTP server with Node.js, we need to do the following steps:
  1. Import the `http` module using the `require` function.
  2. Create a server object using the `http.createServer` method, and pass a callback function that handles the incoming requests and sends the responses.
  3. Start the server using the `server.listen` method, and specify the port number and the hostname where the server will run.
  4. Optionally, add event listeners for the server object to handle different events, such as `request`, `connection`, `close`, etc.
- Here is an example of a simple HTTP server that returns a plain text message to the client:

```
// Import the http module const http = require('http'); // Create a server ob
ject const server = http.createServer((req, res) => { // Write the response h
eaders res.writeHead(200, {'Content-Type': 'text/plain'}); // Write the respo
nse body res.write('Hello, this is a Node.js HTTP server'); // End the respon
se res.end(); }); // Start the server on port 3000 and localhost server.liste
n(3000, 'localhost', () => { console.log('Server is running at http://localho
st:3000'); });
```

- To test the server, we can open a web browser and navigate to `http://localhost:3000`, or use a command-line tool like `curl` to send an HTTP request to the server.

HTTP Protocol

What are the common methods you can send to your BE server?

- HTTP methods are verbs that indicate the type of action that the client wants to perform on the server's resource1.
- There are many HTTP methods, but the most common ones are:
  - GET: used to request a representation of a resource, without modifying it1. For example, `GET /users` can return a list of users.
  - POST: used to create a new resource or submit data to the server1. For example, `POST /users` can create a new user with the data provided in the request body.
  - PUT: used to replace or update an existing resource with the data provided in the request body1. For example, `PUT /users/1` can update the user with id 1 with the new data.
  - DELETE: used to delete an existing resource1. For example, `DELETE /users/1` can delete the user with id 1.

What are the common status codes the backend responds with?

- HTTP status codes are three-digit numbers that indicate the result of the server's processing of the client's request2.
- There are five classes of HTTP status codes, each with a different meaning:
  - 1xx: informational responses, indicating that the request was received and is being processed2. For example, 100 Continue means that the server has received the request headers and the client can proceed to send the request body.
  - 2xx: successful responses, indicating that the request was successfully completed2. For example, 200 OK means that the request was fulfilled and the response body contains the requested resource.
  - 3xx: redirection responses, indicating that the client needs to perform additional actions to complete the request2. For example, 301 Moved Permanently means that the requested resource has been moved to a new URL and the client should use that URL from now on.
  - 4xx: client error responses, indicating that the request was invalid or cannot be fulfilled by the server2. For example, 404 Not Found means that the requested resource does not exist on the server.
  - 5xx: server error responses, indicating that the server encountered an error while processing the request2. For example, 500 Internal Server Error means that the server encountered an unexpected condition that prevented it from fulfilling the request.

Why do we need status codes? Why can't we just return in the body something like success: true/false

- Status codes are useful because they provide a standardized and concise way of communicating the outcome of the request to the client3.
- Status codes can also help the client to handle different scenarios and take appropriate actions, such as retrying, redirecting, displaying an error message, etc3.
- Returning only success: true/false in the body would not be sufficient, because it would not convey the reason for the success or failure, or the details of the response3.

- For example, if the client requests a resource that does not exist, returning success: false would not tell the client whether the resource was deleted, moved, or never existed in the first place. A status code of 404 would be more informative and helpful.

Why do we need so many types of request methods? Why can't just one work?

- Having different types of request methods allows the client to express the intent and semantics of the request more clearly and accurately4.
- Different request methods also have different properties and implications, such as idempotency, safety, cacheability, etc4.
- Idempotency means that repeating the same request multiple times will have the same effect as making it once4. For example, GET and PUT are idempotent, but POST is not.
- Safety means that the request does not modify the state of the server4. For example, GET and HEAD are safe, but POST, PUT, and DELETE are not.
- Cacheability means that the response can be stored and reused by intermediaries, such as proxies or browsers4. For example, GET and HEAD are cacheable, but POST, PUT, and DELETE are not.
- Having different request methods allows the server and intermediaries to optimize the performance, reliability, and security of the communication4.

Why do we need body/headers/query params, why can't just one work?

- Body, headers, and query params are different ways of passing information from the client to the server in an HTTP request5.
- Body is the part of the request that contains the data or content that the client wants to send to the server5. For example, the body can contain a JSON object with the user's details, or a file to be uploaded.
- Headers are key-value pairs that provide additional information or metadata about the request or the response5. For example, headers can specify the content type, the authorization, the cache control, etc.

- Query params are key-value pairs that are appended to the URL after a question mark5. For example, query params can be used to filter, sort, or paginate the results of a GET request.
- We need different ways of passing information because they have different purposes and characteristics5.
- Body is mainly used for sending large or complex data that cannot fit in the URL or the headers5. Body is also only supported by some request methods, such as POST and PUT5.
- Headers are mainly used for sending small or common data that are relevant to the request or the response as a whole5. Headers are also standardized and have predefined meanings and formats5.
- Query params are mainly used for sending small or optional data that are relevant to the resource being requested5. Query params are also visible in the URL and can be modified by the user or bookmarked5.

How do I create a HTTP server of my own?

- To create a HTTP server of your own, you need to have a computer or a server that can run a program that listens for and handles HTTP requests.
- You also need to have a domain name that can be resolved to the IP address of your computer or server, or use a dynamic DNS service that can update the IP address automatically.
- You also need to have a web framework or a library that can simplify the process of creating and running a HTTP server, such as Express for Node.js, Flask for Python, or Rails for Ruby.
- You also need to have a web application or a website that can provide the content or the functionality that you want to offer to the clients.
- You also need to configure your firewall and router to allow incoming HTTP traffic on port 80 (or 443 for HTTPS) and forward it to your computer or server.
- Here is an example of how to create a simple HTTP server using Express for Node.js:

```
// Import the express module const express = require('express'); // Create an express app const app = express(); // Define a route handler for the root path app.get('/', (req, res) => { // Send a plain text response res.send('Hello, this is a HTTP server'); }); // Start the server on port 3000 app.listen(3000, () => { console.log('Server is running at http://localhost:3000'); });
```

How do I expose it over the internet like chatgpt.com?

- To expose your HTTP server over the internet, you need to have a public IP address that can be reached by other computers on the internet.
- You also need to have a domain name that can be mapped to your public IP address, or use a service that can provide you a subdomain and proxy the requests to your server.
- You also need to have a SSL certificate that can encrypt the communication between your server and the clients, and use HTTPS instead of HTTP.
- You also need to have a reliable and fast internet connection that can handle the traffic and bandwidth of your server.
- Here are some possible ways to expose your HTTP server over the internet:
    - Use a cloud hosting service, such as AWS, Azure, or Google Cloud, that can