NodeJS

Contents

1. Definition

- **Node.js** is an open-source server environment, server-side JavaScript runtime environment. It allows developers to run JavaScript code outside of a web browser and on the server.
- **Node.js** runs the V8 JavaScript engine, the core of Google Chrome, outside of the browser.
- **Node.js** is free.
- **Node.js** uses an event-driven, non-blocking I/O model.
- **Node.js** runs on various platforms (Windows, Linux, Unix, Mac OS X, ...)
- **Node.js** provides a large ecosystem of modules and libraries, making it easier for developers to build server-side applications with JavaScript.

2. Use cases

Node.js is a versatile technology that can be used for a wide range of use cases and there are various libraries and frameworks available to support each use case. Here are some common **Node.js** use cases along with associated libraries or frameworks:

2.1. Web servers

Node.js is widely used in web servers because of its non-blocking I/O model and event-driven architecture. It allows developers to build scalable and efficient web servers that can handle a large number of concurrent connections.

- Express.js: A fast and minimalist web framework that allows you to build web applications and APIs.
- Koa.js: A modern, lightweight web framework designed for high-performance web applications.
- Electronjs: The Electron framework lets you write cross-platform desktop applications using JavaScript, HTML, and CSS. It is based on **Node.js** and Chromium and is used by Visual Studio Code and many other apps.

2.2. Real-time applications

Due to its event-driven architecture, **Node.js** is well-suited for building real-time applications such as chat applications, collaborative editing tools, and multiplayer games.

- Socket.io: A library that enables real-time, bidirectional communication between clients and servers.
- Sails.js: A full-featured MVC *(Model-View-Controller)* framework that includes real-time capabilities.

2.3. API development:

Node.js is commonly used to build RESTful APIs and microservices, providing a lightweight and efficient backend for front-end applications or mobile apps.

- Restify: A framework specifically designed for building REST APIs.
- Hapi.js: A powerful framework for building APIs and websites that includes support for developer-friendly features like input validation and authentication.

2.4. Command-line tools:

Node.js provides a rich set of APIs for interacting with the file system, network, and operating system, making it an excellent choice for building command-line tools and scripts.

- Commander.js: A feature-rich library for building command-line interfaces (CLIs) with **Node.js**.
- Inquirer.js: A library for creating interactive command-line interfaces with a wide range of user prompts.

2.5. Data streaming:

Node.js is particularly effective in handling streaming data, such as real-time analytics, file uploads/downloads, and audio/video processing.

- Async.js: A utility library that provides powerful functions for handling asynchronous operations.
- Fastify: A performant and low-overhead web framework suitable for building efficient applications, including data processing tasks.

2.6. IoT applications:

With its lightweight footprint, event-driven architecture, and support for asynchronous programming, **Node.js** is well-suited for building IoT *(Internet of Things)* applications and controlling embedded devices.

- Johnny-Five: A JavaScript robotics framework for **Node.js** that supports a wide range of devices and platforms.
- Cylon.js: A web-based JavaScript robotics framework for **Node.js** that provides a simple, unified API for interacting with various physical devices.

3. Special things about nodejs

Node.js is a powerful, open-source, server-side runtime environment that allows developers to build scalable applications using JavaScript. Here are some special things about Node.js:

3.1. Event-driven architecture and Non-blocking I/O model

One of the most remarkable and standout features of Node.js is undoubtedly its event-driven architecture and non-blocking I/O model. These features offer several notable benefits such as scalability and responsiveness that significantly enhance the efficiency and performance of applications.

3.2. Single Language for Frontend and Backend

With Node.js, developers can use JavaScript both on the server side and the client side, which leads to code reusability, reduced complexity, and faster development.

3.3. NPM (Node Package Manager)

Node.js has a built-in package manager called NPM, which hosts thousands of open-source packages and modules. NPM makes it easy for developers to find, install, and manage dependencies for their projects, greatly accelerating the development process.

3.4. Large Ecosystem and Active Community

Node.js has a vibrant and active community that constantly contributes to its growth. This has resulted in a wide array of libraries, frameworks, and toolsets that enhance the capabilities of Node.js. This large ecosystem greatly reduces the development time and effort required for building applications.

3.5. Cross-platform Compatibility

Node.js can run on various platforms, including Windows, macOS, and Linux, making it highly flexible and versatile.

4. Functionality

4.1. File system:

To handle file operations like creating, reading, deleting, etc., Nodejs provides an inbuilt module called FS (File System).

- Common use for the File System module:
  - The **fs.readFile()** method is used to read files

    ```
    const fs = require('fs);
    fs.readFile('Fcode.html);
    ```
  - The **fs.appendFile()** method appends specified content to a file. If the file does not exist, the file will be created:

    ```
    const fs = require('fs);
    fs.appendFile('Fcode.txt', 'Adding text to file');
    ```
  - The **fs.appendFile()** method appends the specified content at the end of the specified file:
    - The **fs.open()** method takes a FileSystem flag as the second argument, the specified file is opened for writing. If the file does not exist, an empty file is created:

      ```
      const fs = require('fs);
      fs.open('Fcode.txt', 'w');
      ```
    - The **fs.writeFile()** method replaces the specified file and content if it exists. If the file does not exist, a new file, containing the specified content, will be created:

      ```
      const fs = require('fs);
      fs.writeFile('Fcode.txt', 'Hello and smile');
      ```
    - The **fs.writeFile()** method replaces the specified file and content:

      ```
      const fs = require('fs);
      fs.writeFile('Fcode.txt', 'Hello to my crew');
      ```
    - The **fs.rename()** method renames the specified file:

      ```
      const fs = require('fs);
      fs.rename('FIn4.txt', 'Fcode.txt');
      ```
    - The **fs.unlink()** method deletes the specified file:

      ```
      const fs = require('fs);
      fs.unlink('Membername.txt');
      ```

4.2. Net, HTTP/HTTPS

4.2.1. HTTP *(Hypertext Transfer Protocol)*

- HTTP is like a language that your web browser and the website's server use to talk to each other. It's how you get information from the server onto your browser.
- Imagine if everyone spoke English, and a hacker who knows English could easily understand any information you send. That's how HTTP works—everything is in plain text.
- When you visit a website your browser sends a request to the server, and the server responds with the page you see.
- Features:
  - Plain text communication.
  - Used for sending HTML documents, images, and videos to your browser.
  - Operates at the application layer of networking.

4.2.2. HTTPS *(Hypertext Transfer Protocol Secure)*

- HTTPS is like a secret language. It encrypts the communication between your browser and the server so that hackers (hopefully) can't understand it.

- When you access a bank's website using HTTPS, your data is protected. Even if a hacker intercepts it, they won't understand the encrypted conversation.
- When you visit a secure site (like your online banking), the URL starts with "https://" (e.g., your bank's website).
- Features:
  - Encrypted communication.
  - Boosts your site's ranking on Google.
  - Protects against phishing attacks.
  - Uses SSL certificates for security.

4.2.3. Main Differences:

- Encryption:
  - HTTP: No encryption layer.
  - HTTPS: Enabled encryption.
- Data Protection:
  - HTTP: Data is not secure.
  - HTTPS: Data is protected.
- Google Ranking:
  - HTTP: No ranking boost.
  - HTTPS: Boosts your ranking.
- Phishing Protection:
  - HTTP: No protection.
  - HTTPS: Guards against phishing.

4.3. Promise, async, await

4.3.1. Event loop

- Nodejs event loop is a semi-infinite loop, polling and blocking on the OS until some in a set of file descriptors are ready. The loop exits when it no longer has any event to wait for
- The event loop uses epoll on Linux, kqueue on MacOS and BSD for polling
- The ways Nodejs handles polling could be categorized into three cases:
  - Pollable file descriptors: can be directly waited on, including sockets (net, dgram, http, https, tls, child process pipes, stdin, stdout, stderr)
  - Time: the next timeout can be directly waited on
  - Others:
    - Including fs.*, dns.lookup(), crypto.randomBytes(), cryto.pbkdf2(), ...
    - Using uv thread pool. The blocking call is made by a thread, and when it completes, readiness is signaled back to the event loop using either and eventfd or a self-pipe (self-pipe is a pipe, where one end is written to by a thread or signal handler, and the other end is polled in the loop)

4.3.2. Promise

Nodejs promise provides high-level APIs to add functions to be executed when events occur in the event loop

4.3.3. Async, await

- The async function declaration creates a binding of a new async function to a given name. The await keyword is permitted within the function body, enabling asynchronous, promise-based behavior to be written in a cleaner style and avoiding the need to configure promise chains explicitly.
- Async, await enables the use of ordinary try/catch blocks around asynchronous code instead of .-catch in promise chains

4.4. Worker threads

- The worker thread module implements a form of threading that provides parallelism in nodejs
- Worker threads are not OS threads. They are distinct child processes, which means they can't directly access the execution context of their parents.
- Communication between the main application and worker threads is facilitated by an event-based messaging system
- Worker threads are most suitable for CPU-bound operations, consisting of image editing, video editing, cryptography, and complex mathematical operations,...
- Example use cases of worker thread module:

```
const {
    Worker, isMainThread, parentPort, workerData,
} = require('node:worker_threads');

if (isMainThread) {
    const data = {
        "editor.suggest.snippetsPreventQuickSuggestions": false,
        "editor.suggest.matchOnWordStartOnly": false,
        "editor.foldingImportsByDefault": true,
        "editor.inlineSuggest.enabled": true,
        "editor.suggest.localityBonus": true,
        "editor.suggestSelection": "first",
        "editor.accessibilitySupport": "off",
        "editor.stickyScroll.enabled": true,
        "editor.smoothScrolling": true,
    }

    const worker = new Worker(__filename, {
        workerData: data,
    });
    let result;
    worker.on('message', (data) => {
        result = data;
    })
    worker.on('exit', () => console.log(result))
} else {
    const data = workerData;
    parentPort.postMessage(JSON.stringify(data));
}
```

4.5. C/C++ addons

4.5.1. V8 Engine

- V8 is Google's open-source high-performance JavaScript and WebAssembly engine, written in C++. It is used in Chrome and in Node.js, among others.
- V8 is at the core of Node.js.

4.5.2. Why do we need C++?

- You can use existing, proven, and efficient algorithms or libraries already written for C/C++.
- You can develop applications that need hardware-level or OS-level operations.
- You can run CPU-intensive operations much faster in C++ than JavaScript.

4.5.3. C++ addons

- Addons are dynamically linked shared objects written in C++. The require() function can load addons as ordinary Node.js modules. Addons provide an interface between JavaScript and C/C++ libraries.

- There are three options for implementing addons: Node-API, nan, or direct use of internal V8, libuv, and Node.js libraries.

4.6. WASI

- WebAssembly *(abbreviated Wasm)* is a binary instruction format for a stack-based virtual machine. Wasm is designed as a portable compilation target for programming languages, enabling deployment on the web for client and server applications.
- WebAssembly System Interface *(WASI)*
  - WASI is a modular system interface for WebAssembly. As described in the initial announcement, it's focused on security and portability.
  - Source Code: *lib/wasi.js*
  - The WASI API provides an implementation of the WebAssembly System Interface specification. WASI gives WebAssembly applications access to the underlying operating system via a collection of POSIX-like functions.

5. Comparison

5.1. Javascript/NodeJs vs Golang

- Programming Language's type:
  - JavaScript: scripting language, object-oriented programming (OOP).
  - Go compiled language, procedural programming.
- Ability to run:
  - JavaScript/Node.js: runs on many platforms through the JS engine.
  - Go: compiles directly to machine code for multiple platforms.
- Performance:
  - JavaScript/Node.js: slower because it requires interpretation.
  - Go: faster thanks to direct compilation.
- Multi-threading:
  - JavaScript/Node.js: js is single-threaded, Node.js supports multi-threading but handles inefficiently.
  - Go: supports efficient multi-threading in m:n model.
- Frontend – Backend
  - JavaScript/Node.js: developing a true client-server system works really well.
  - Go: focuses more on the backend, especially for developing high-performance concurrent services on the server.

⟹ In general, Go is specifically designed for server-side, higher performance but more complex. JavaScript is simple and easy for beginners to use.

5.2. Javascript/NodeJs vs C#

- Programming Languages's type
  - JavaScript is a scripting language designed for the web environment.
  - C# is a fully-featured object-oriented programming language.
- Execution Environment
  - JavaScript/Node.js runs on the JavaScript V8 Engine.
  - C# runs on Microsoft's .NET Framework/Core.
- Performance
  - JavaScript/Node.js is slower because it requires code interpretation.
  - C# is faster because it compiles directly to machine code.
- Application Scope
  - JavaScript/Node.js is popular for web/server-side programming.
  - C#: More versatile, can be used to develop many different types of applications.

⟹ In general, C# is widely used to develop applications on Windows such as desktop, web, mobile, games, and business applications. Js is strong in the web field.