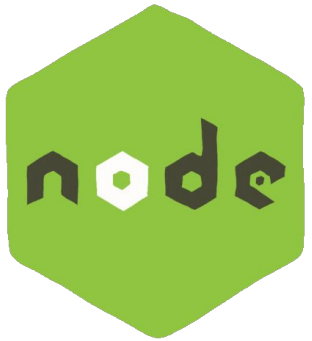
Node.js



It changed the game...

Agenda

Node.js

- Quick Intro
- Node.js: The Beginning
- What Is Node.js?
- Why Use Node.js?
- Installing Node.js
- Node Modules
- Node Loaders
- Node Packages
- Node Package Manager

What is this Node.js?

Node.js is a platform for building applications

- It allows you to run JavaScript outside of the browser.
- Which means you can build desktop apps, too!
- It's really fast but... the reason for that is a little complicated.
- Let's give it a little context by understanding how and why Node was created.

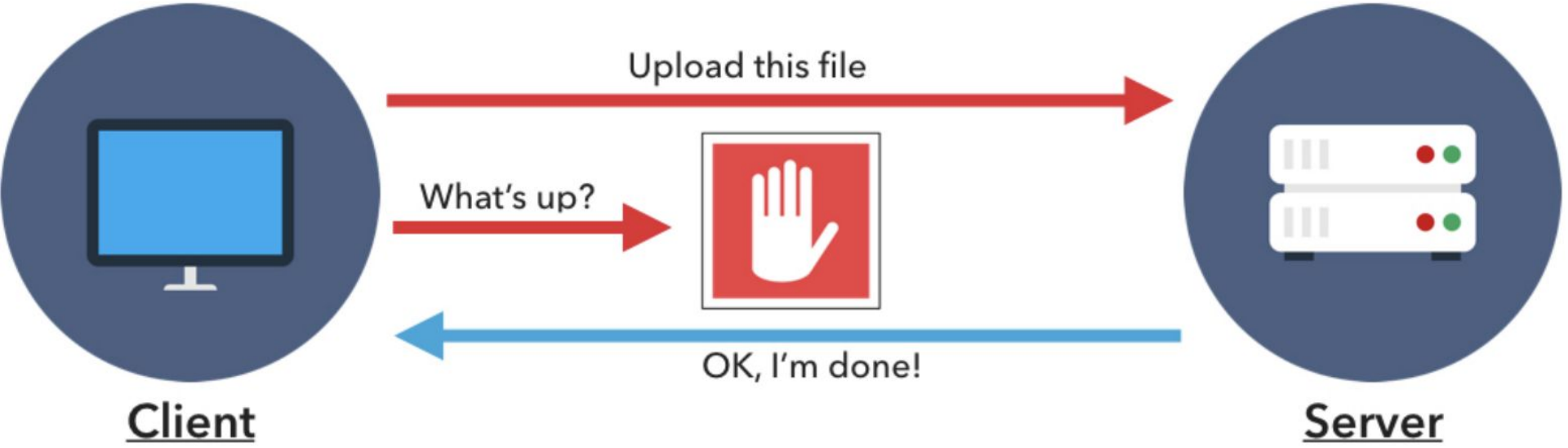
Node.js: The Beginning

Ryan Dahl The Creator

Ryan Dahl, unsatisfied with the constraints of existing web servers, created Node.js in 2009.

He first presented Node.js at a JavaScript Conference in 2009.

BLOCKING



NON- BLOCKING

What is Node.js?

What is Node.js?

Node.js is an open-source, cross-platform **runtime environment** that allows developers to run JavaScript code outside of a web browser. It is built on the **V8 JavaScript engine**, which is the same engine used by Google Chrome. Node.js provides a server-side execution environment for JavaScript, enabling developers to build scalable and high-performance applications. It also uses an event driven, non-blocking I/O model that makes it lightweight and efficient.

What is a Javascript Engine?

A JavaScript engine is a computer program or a part of a web browser that interprets and executes JavaScript code. It is responsible for parsing the JavaScript code, optimizing its execution, and translating it into machine code that the computer can understand and execute.

But then what is a Javascript runtime?

A JavaScript runtime is an environment that provides the necessary infrastructure and resources for executing JavaScript code. It includes components such as a JavaScript engine, a set of libraries, and other supporting functionalities.

Examples:

- Web browser
- Node.js
- Mobile application framework

Ultimately, It abstracts away the underlying hardware and operating system details, allowing developers to write and run JavaScript code across different platforms.

Google Chrome vs Node.js

Chrome

- Uses **client-side** javascript at runtime
- Runtime is part of the browser
- Handles tasks such as
  - Rendering web pages
  - Executing javascript

Node.js

- Uses **server-side** javascript at runtime
- Installed on a computer or server
- Handles tasks such as :
  - Database queries
  - File I/O requests

What is Node.js?

Node.js is an open-source, cross-platform runtime environment that allows developers to run JavaScript code outside of a web browser. It is built on the V8 JavaScript engine, which is the same engine used by Google Chrome. Node.js provides a server-side execution environment for JavaScript, enabling developers to build scalable and high-performance applications. It also uses an **event driven, non-blocking I/O model** that makes it lightweight and efficient.

What is event-driven?

- Event-driven programming is a popular paradigm in Node.js.
- It revolves around events triggered by actions or occurrences.
- Events are defined and handled using the EventEmitter class.
- Example events: "nextSlide" and "showContent".
- Handlers are executed when events are emitted.
- Enables building scalable and efficient applications.
- Promotes decoupling and modularity.
- Provides responsiveness and asynchronous behavior.

Node is Event-Driven

- Node.js recognizes events and sends actions for processing.
- It creates callbacks to handle actions after processing.
- Callbacks are like reminders for Node.js to handle the next event.
  - Example: File open event ↠ Append date ↠ Close file.

Node is Non-Blocking

- Non-blocking operations are also known as asynchronous operations.
- Node.js allows other code to execute while waiting for asynchronous operations to complete.

Blocking

- Blocking operations are sometimes referred to as synchronous operations.
- No other code can execute until the synchronous operation completes.
- If the operation is slow, it can cause delays in the program execution.

Node.js Event Loop and Single Thread

- A thread is a single computer process.
- Node.js's main Event Loop runs in a single thread.
- Events and callbacks are queued in the order they are received.

Node.js Handling a Web Request

1. A web request is received by Node.js.
2. Node.js executes the handler for that request.
3. The handler initiates a database query with a callback.
4. Node.js is free to handle other requests while waiting for the query to complete.
5. When the database query ends, Node.js is notified (event).
6. Node.js adds the callback to the queue.
7. Node.js executes the handler after processing any events before it in the queue.

Why use Node.js?

The Why

1. JavaScript on the server-side: Write both client-side and server-side code in a single language (JavaScript), simplifying development and improving code reusability.

2. Asynchronous and non-blocking: Handles concurrent connections efficiently, allowing for better scalability and responsiveness.

3. Fast and efficient: Built on the V8 JavaScript engine, delivering high performance. Event-driven, non-blocking I/O model ensures efficient execution.

4. Rich ecosystem and package manager: Access thousands of open-source libraries through npm, enabling quick integration of functionality into applications.

5. Community support and active development: Large and vibrant developer community, providing resources and continuous improvement of the platform.

6. Cross-platform compatibility: Runs on Windows, macOS, and Linux, making it versatile for different operating systems and devices.

Installing Node.js

Installation Steps

1. Go to https://nodejs.org
2. Click the "LTS" download button
3. Open the installer
4. Follow the prompts to complete the installation

Basic Node Terminal Commands

- **node**
  - Opens an interactive shell where you can execute Javascript code
- **node file_name.js**
  - Executes Javascript code that is in the file
- **node -v**
  - Displays the version of Node installed on your machine

Exercise 1 : Hello Node!

1. Using the Node interactive shell, output "Hello Node!" in the terminal.

2. Save a "Hello Node!" script in a file and execute that file in the Terminal.

Node Modules

What are Node Modules?

- A module encapsulates related code into a single unit of code.
- Creating a module can be interpreted as moving all related functions into a file.

Benefits of Modules

1. Code organization and reusability: Organize code into separate modules for better organization and reusability.

2. Encapsulation and information hiding: Encapsulate functionality and expose only necessary elements to enhance code modularity.

3. Dependency management: Easily manage project dependencies using tools like npm, ensuring consistent and reliable integration of external libraries.

4. Improved maintainability and testing: Modular code structure simplifies maintenance and allows for focused testing of individual components.

5. Performance optimization: Lazy loading of modules reduces initial loading time, enhancing overall application performance.

6. Vast ecosystem: Access a large collection of pre-built modules through npm, saving development time and effort.

Core Modules

- Node.js comes with built-in modules called Core Modules.
- Core Modules include:
  - fs: Events, classes, and functions for file input/output.
  - http: Events, classes, and functions for creating an HTTP server.
  - util: Functions to assist with debugging, inspecting, and deprecating code.
  - path: Functions for working with file and directory paths.

Third-Party Modules in Node.js

- External modules are sometimes referred to as Third-Party Modules.
- These modules are created by other developers and made publicly available.
- They can save developers a lot of time and effort.

Custom Modules

- Custom modules are usually specific to a developer's application.
- They can be as simple as a greeting module.

Example:

```
// module.js
console.log('Hello Class!');

// app.js
require('./module.js');
```

In this example, the custom module module.js logs the message "Hello Class!" when it is required in the app.js file.

Module Loaders

Require JS

- RequireJS is a JavaScript file and module loader.
- Using a modular script loader like RequireJS improves the speed and quality of your code.

require()

- ‘require()’ is a function that points to the path of a module that will be used.
- If a path is specified, ‘require’ will traverse to it and look for the module there.
- ‘require’ supports relative and absolute paths.

Importing a Module

```
var models = require('./models');
var absolute = require('/some/absolute/path/models');
```

- The first require looks for a file called models.js, located in the same directory as the current file.
- The second require looks for a file on an absolute path.

Exporting a module

```
// module_one.js
exports.foo = 'bar';

// app_one.js
var a = require('./module_one');
console.log(a.foo);

// module_two.js
module.exports = 'hello';

// app_two.js
var b = require('./module_two');
console.log(b);
```

- Whatever module.exports is set to defines what is available when including a module in your app.
- module.exports is returned to the requiring file.
- exports collects properties, ultimately attaching them to module.exports.

Simple Example

```
// make `helloWorld` callable via `require`!
function helloWorld() {
  return "Hello World!";
}

// export the `helloWorld` method directly
module.exports = helloWorld;
```

- exports can be a single function.
- The helloWorld function is defined.
- module.exports is set to the helloWorld function, allowing it to be called when the module is required.

Multiple Functions Example

```
// make `helloWorld` _and_ `helloPerson` callable via `require`
function helloWorld() {
  return "Hello World!";
}

function helloPerson(name) {
  return `Hello ${name}!`;
}

// wrap the methods in an object and export
// note that the functions could be renamed when exporting
module.exports = {
  helloWorld: helloWorld,
  helloPerson: helloPerson
};
```

- exports can also be an object, wrapping local functions.
- The helloWorld and helloPerson functions are defined.
- The functions are wrapped in an object and exported using module.exports.
- Now, both helloWorld and helloPerson functions can be called when the module is required.

Pass Paramets to Module - Example

```
// this is in my-module.js
var extraInformation;

function helloWorld() {
  return "Hello world!";
}

function helloPerson(name) {
  return `Hello ${name}. ${extraInformation}`;
}

module.exports = function(info) {
  extraInformation = info;
  return {
    helloWorld: helloWorld,
    helloPerson: helloPerson
  };
};

// the following is in app.js at the same directory as my-module
var myModule = require('./my-module')("It's a beautiful day.");

// prints "Hello Sally. It's a beautiful day."
console.log(myModule.helloPerson('Sally'));
```

- exports can take a parameter, allowing configuration setup.
- The my-module.js file exports a function that takes an info parameter.
- The extraInformation variable is set based on the provided info.
- An object with the helloWorld and helloPerson functions is returned from the module.
- In app.js, the myModule variable is assigned the result of requiring my-module with a specific parameter.
- Finally, the helloPerson function is called on myModule with the parameter passed during module import.

Install RequireJS

npm install requirejs in the folder you want to use it in

Step 1 - Create Modules

Module 1: set-difference.js

- Objective: Set Difference
- Description: Given two arrays of strings, produce a single array of items that are in one or the other but not both.

Module 2: set-intersection.js

- Objective: Set Intersection
- Description: Given two arrays of strings, produce a single array of unique items that are in both sets.

Step 2 - Create app.js File

```
var setDifference = require('./set-difference');
var setIntersection = require('./set-intersection');
var set1 = ['dogs', 'cats', 'red', 'bananas', 'code', 'movies'];
var set2 = ['blue', 'horses', 'dogs', 'code', 'rain'];
var difference = setDifference(set1, set2);
var intersection = setIntersection(set1, set2);

// should print an array with cats, red, bananas, movies, blue, rain as elements
console.log(difference);

// should print an array with dogs and code as elements
console.log(intersection);
```

- In the app.js file, require the set-difference and set-intersection modules.
- Use the provided sets (set1 and set2) to call the respective functions.
- Print the results using console.log().

Node Packages

What are Node Packages?

- Packages are one or more modules that have been packaged together
- The list of modules in a package is defined in the package.json
  - React is just a module
  - Redux is just a module
  - etc.

Package.json

- package.json is a file that defines important information about your Node.js package.
- It specifies the packages your package depends on.
- It includes the name and version of your package.
- It determines the executed file when the package is required.
- It holds basic licensing information.
- It provides author and repository information.
- Optionally, it can define scripts and tasks using the "scripts" field, like npm run [task]

Node Package Manager - npm

- The NPM registry is a central place to get open-source modules, saving you from reinventing the wheel.
- It allows for installation from various types of locations, including the NPM registry, Git, symbolic links, or tarball archives.
- It helps manage installed modules, making it easy to track dependencies and update packages.

Npm init

- package.json is created for your project.
- It collects important information about your project through interactive prompts.
- The prompts include author information, repository details, package name, and more.

npm install:local

- Packages are installed locally to the node_modules directory.
- Once installed, the package can be pulled into your source code using require by name.
- Running npm install with the --save flag saves the dependency in your package.json.

npm install: global

- Packages can be installed globally, which defaults to the .npm directory under your home directory.
- The -g flag is used to denote a global install.
- Global installation is typically used for command-line tools.

npm uninstall

- Uninstalling a package follows the same rules as installation.
- It removes the package from the node_modules directory.
- Using the -g flag uninstalls global packages.
- The --save flag removes the dependency from package.json.

npm ls

- The tree command shows a tree-like structure of the installed modules, allowing you to see dependencies and their dependencies at a glance.
- It can be run without any flags to show locally installed modules.
- Using the -g flag shows all globally installed modules.
- The --depth [number] flag limits the depth of dependencies shown.

Exercise 1: Create a package

Step 1:

- Create a directory called "my-module" with a file called "my-module.js".
- Inside "my-module.js", make a call to require to pull in Express.

Step 2:

- Run npm init and answer the prompts to initialize your package.
- Install Express by running npm install express --save to save it to your package.json.

Step 3:

- View module dependencies:
  - Run npm ls to see a tree-like structure of installed modules, including Express.
  - Run npm ls --depth 0 to limit the depth and observe the differences in the output.

Exercise # 2: Countdown Timer

1. Create a folder on your Desktop: Create a new folder on your Desktop and give it a name (e.g., "countdown-app").
2. Set up files:
    a. Inside the "countdown-app" folder, create an index.html file. This file will contain the HTML structure of your countdown timer app.
    b. Create a separate JavaScript file (e.g., script.js) inside the "countdown-app" folder. This file will hold your countdown timer code.
3. Install Node HTTP Server:
    a. Open a terminal or command prompt.
    b. Install the Node HTTP server by running the following command: npm install -g http-server.
    c. This command will install the http-server package globally, allowing you to serve your static files.
4. Start the server:
    a. In the terminal or command prompt, navigate to the "countdown-app" folder.
    b. Start the HTTP server by running the command: http-server -p 8080.
    c. This command starts the server on port 8080, but you can choose a different port if needed.
5. Access the app:
    a. Open a web browser and visit http://localhost:8080 (replace 8080 with the chosen port if different).
    b. Your countdown timer app should be accessible now.