Server and Callback Functions

Date	@September 11, 2024
Status	Started
Topic	callback function, importing and exporting js files, lodash, nodejs file system, nodejs os, npm, package.json

learn npm init

```
npm init
```

npm init is cli command which initiate node in our project, this create two files package.json and package.lock.json which contains data about our project like project name, version, dependencies.

Understanding package.json

package.json -

- ensure a list of packages with their version
- Analogy: list of cloth you want to purchase with size and all, it also consists of other metadata like name and all.

package.lock.json-

- ensure detailed of every package installed with version, sub dependencies, store detailed, discount everything...
- it’s like a detailed billed

```
{
  "name": "learning_nodejs",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exi
  },
  "repository": {
    "type": "Learning_Node-Js",
    "url": "https://github.com/itsarman001/Learning_N
  },
  "author": "ARMAN",
  "license": "ISC",
  "dependencies": {
    "express": "^4.20.0",
    "prompt-sync": "^4.2.0"
  }
}
```

Both files work together to ensure smooth development for you and your team.

Callback Function

A callback function is a function passed as an argument to another function, which is then invoked inside the outer function to complete some kind of routine or action. In JavaScript, callbacks are often used to handle asynchronous operations. They allow you to execute code after a specific task has finished, ensuring that certain code doesn't run until needed.

Here's a simple example to illustrate the concept:

```
function greet(name, callback) {
  console.log('Hello ' + name);
  callback();
}

function callMe() {
  console.log('I am callback function');
}

greet('John', callMe);
```

Output

```
Hello John
I am callback function
```

Node Js Files System & OS

Files System

The Node.js File System module allows you to work with the file system on your computer. It provides a way of reading from and writing to files, as well as performing various operations on directories. Some common operations include:

- Reading files (fs.readFile)
- Writing files (fs.writeFile)
- Appending to files (fs.appendFile)
- Deleting files (fs.unlink)

These operations can be performed synchronously or asynchronously, with the asynchronous versions being preferred for better performance in most cases.

OS

The Node.js OS module provides a way to interact with the operating system. It offers methods for retrieving information about the system's CPU, memory, network interfaces, and more. Some common uses include:

- Getting information about the operating system (os.platform(), os.type())
- Retrieving system memory information (os.totalmem(), os.freemem())
- Getting CPU information (os.cpus())

These functions can be useful for system monitoring, performance optimization, and creating cross-platform applications.

Import and Export

Import

In Node.js, importing modules or files is done using the require() function. This function is used to include external modules that exist in separate files. Here's a basic example:

```
// Importing a core module
const fs = require('fs');

// Importing a local file
const myModule = require('./myModule');

// Importing an installed package
const lodash = require('lodash');
```

The require() function returns the exports of the module, which can then be used in your code. It's important to note that Node.js uses the CommonJS module system by default, although ES6 modules are also supported with certain configurations.

Export

Exporting in Node.js allows you to make functions, objects, or variables from one module available for use in another module. There are two main ways to export in Node.js:

- module.exports: This is used to export a single object, function, or value from a module.
- exports: This is used to export multiple named values from a module.

Here's a basic example of exporting:

```
// myModule.js
const myFunction = () => {
  console.log('Hello from myFunction');
};

module.exports = myFunction;

// Or for multiple exports:
exports.func1 = () => console.log('Function 1');
exports.func2 = () => console.log('Function 2');
```

Lodash

Lodash is a popular JavaScript utility library that provides a wide range of helpful functions for working with arrays, objects, strings, and more. It simplifies many common programming tasks and can significantly improve code readability and efficiency. Here are some key points about Lodash:

Usage

After importing Lodash, you can use its functions to manipulate data structures, perform calculations, and handle common programming tasks more easily. Some popular Lodash functions include _.map(), _.filter(), _.reduce(), and _.debounce().

1. Installation

```
npm i lodash
```

2. Importing

There are many methods to import lodash each for different work scenarios use best fit according to your work

```
// Load the full build.
var _ = require('lodash');
// Load the core build.
var _ = require('lodash/core');
```

Note: Using an underscore (_) as the variable name for Lodash is a common convention in many projects. This practice makes the code more recognizable and consistent with other codebases.

3. Commonly use methods

- _.map(): Creates a new array with the results of calling a provided function on every element in the array.
- _.filter(): Creates a new array with all elements that pass the test implemented by the provided function.
- _.reduce(): Reduces an array to a single value by executing a provided function for each value of the array.

Examples:

```
import _ from 'lodash'

// Filter
const numbers = [10, 20, 30, 40, 50];
const filteredNumbers = _.filter(numbers, (num) => num > 30);
console.log(filteredNumbers)

// Map
const names = ["Alice", "Bob", "Charlie"];
const upperCaseNames = _.map(names, (name) => name.toUpperCase());
console.log(upperCaseNames)

// Chunked Array
const myArray = [1, 2, 3, 4, 5, 6];
const chunkedArray = _.chunk(myArray, 2);
console.log(chunkedArray)

// Working with objects

// merge objects
const obj1 = { a: 1, b: 2 };
const obj2 = { b: 3, c: 4 };
const mergedObj = _.merge(obj1, obj2);
```

```
// Get value from nested objects
const user = { info: { name: "John", age: 30 } };
const userName = _.get(user, "info.name");
console.log(`Merged Object: ${mergedObj}, value from nested object: ${userName}`)
```

Summary:

- npm init creates package.json and package-lock.json files, essential for managing project dependencies and metadata
- Callback functions are functions passed as arguments to other functions, often used for handling asynchronous operations in JavaScript
- Node.js File System module enables reading, writing, and manipulating files and directories on the computer
- The OS module in Node.js provides methods to interact with the operating system, retrieving information about CPU, memory, and network interfaces
- Importing in Node.js is done using the require() function, while exporting uses module.exports or exports
- Lodash is a utility library that simplifies common programming tasks and improves code efficiency
- Lodash can be installed via npm and imported in various ways depending on the project requirements
- Common Lodash methods include _.map(), _.filter(), and _.reduce() for array manipulation