

Modules

28 March 2024 12:30

Browser vs Node.js

In the browser, most of the time what you are doing is interacting with the DOM, or other Web Platform APIs like Cookies. You don't have the document, window and all the other objects that are provided by the browser.

In the browser, we don't have all the nice APIs that Node.js provides through its modules. For example the filesystem access functionality.

With Node.js, you control the environment.

With a browser, you are at the mercy of what the users choose

Modules

A module is an encapsulated and reusable chunk of code that has its own context

In Node.js, each file is treated as a separate module

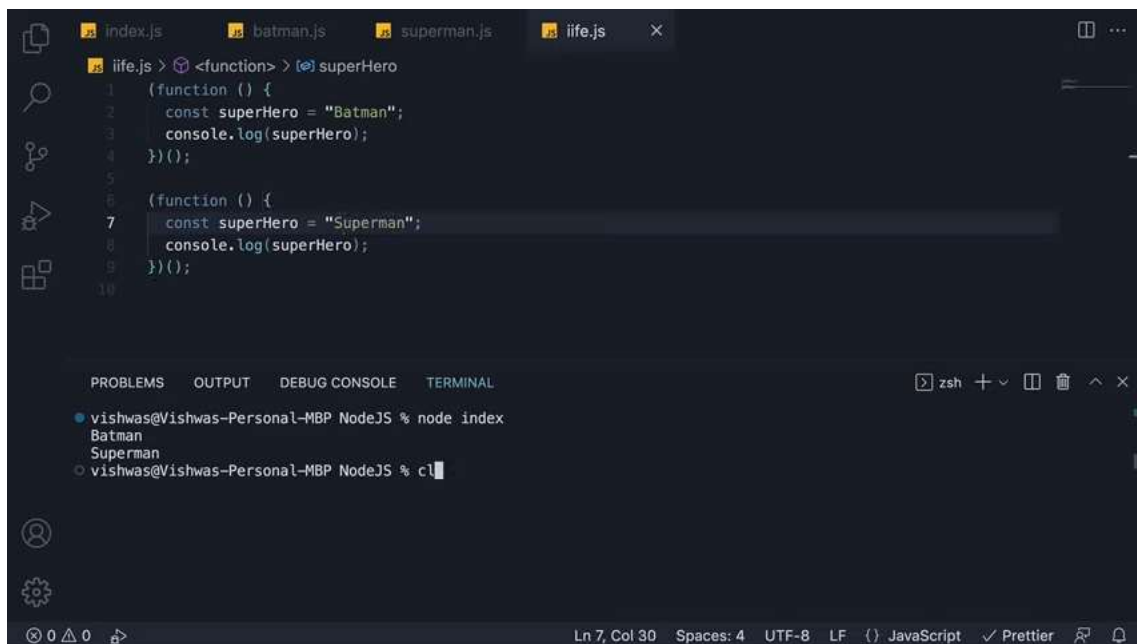
Types of Modules

1. Local modules - Modules that we create in our application
2. Built-in modules - Modules that Node.js ships with out of the box
3. Third party modules - Modules written by other developers that we can use in our application

CommonJS

CommonJS is a standard that states how a module should be structured and shared

Node.js adopted CommonJS when it started out and is what you will see in code bases



The screenshot shows a VS Code editor with two files open: `index.js` and `iife.js`. The `iife.js` file contains the following code:

```
1 (function () {  
2   const superHero = "Batman";  
3   console.log(superHero);  
4 })();  
5  
6 (function () {  
7   const superHero = "Superman";  
8   console.log(superHero);  
9 })();  
10
```

The terminal window at the bottom shows the output of running `node index`:

```
vishwas@Vishwas-Personal-MBP NodeJS % node index  
Batman  
Superman  
vishwas@Vishwas-Personal-MBP NodeJS % cl
```

The status bar at the bottom indicates the current file is `iife.js` at line 7, column 30, using UTF-8 encoding with LF line endings, and is formatted with Prettier.

Immediately Invoked Function Expression (IIFE) in Node.js



```
(function() {  
  // Module code actually lives in here  
})
```

Before a module's code is executed, Node.js will wrap it with a function wrapper that provides module scope

This saves us from having to worry about conflicting variables or functions

There is proper encapsulation and reusability is unaffected

Module Scope Summary

Each loaded module in Node.js is wrapped with an IIFE that provides private scoping of code

IIFE allows you to repeat variable or function names without any conflicts

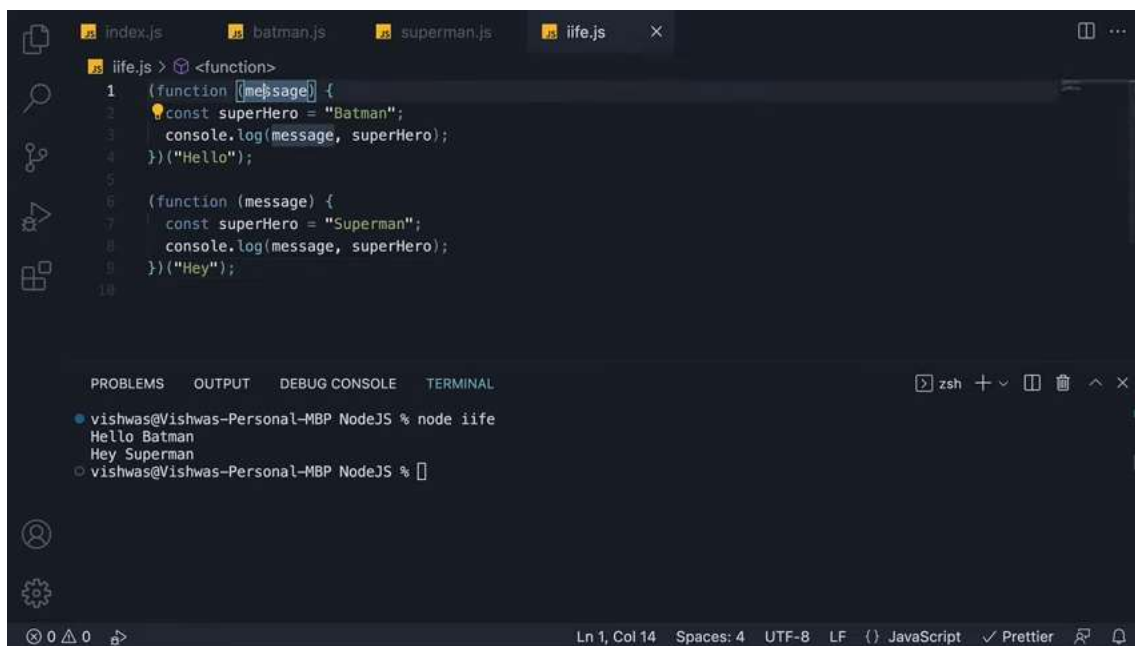
Module Wrapper

Every module in node.js gets wrapped in an IIFE before being loaded

IIFE helps keep top-level variables scoped to the module rather than the global object

The IIFE that wraps every module contains 5 parameters which are pretty important for the functioning of a module

How we pass arguments to iifes.



The screenshot shows a VS Code editor with a file named `iife.js` open. The code in the file is as follows:

```
1 (function (message) {  
2   const superHero = "Batman";  
3   console.log(message, superHero);  
4 })(  
5  
6 (function (message) {  
7   const superHero = "Superman";  
8   console.log(message, superHero);  
9 })(  
10
```

The terminal at the bottom shows the command `node iife` being executed, which results in the output:

```
vishwas@Vishwas-Personal-MBP NodeJS % node iife  
Hello Batman  
Hey Superman  
vishwas@Vishwas-Personal-MBP NodeJS %
```

Module Wrapper contd.



```
const superHero = "Batman";  
console.log(superHero);
```

Same code wrapped in iife.

Module Wrapper contd.



```
(function() {  
    const superHero = "Batman";  
    console.log(superHero);  
})
```

Final code with parameters.

Module Wrapper contd.

```
(function(exports, require, module, __filename, __dirname) {  
  const superHero = "Batman";  
  console.log(superHero);  
})
```

