`02-Node-Module-System/02-module-system.md`

📦 The Module System

🎯 Understanding Modules

Modularity = Small Building Blocks

Functions and objects encapsulated in separate files

📚 What is a Module?

Key Concepts

In a Node.js application:

- **Each separate file** is a module
- **Variables** declared in a file have the scope of **that file only**
- In OOP terms: ≈ **private** (not accessible outside the module)
- Variables/functions can be **exported** to make them public
- Every Node app has at least **one main module**

🔍 The `module` Object

Inspecting the Module Object

The `module` object is **NOT global**! It's specific to each file.

Create `test.js`:

```
console.log(module);
```

Run it:

```
milan@first-app ~\ node test.js
```

Output:

```
Module {
  id: '.',
  path: '/Users/milan/Dev/first-app',
  exports: {},
  filename: '/Users/milan/Dev/first-app/test.js',
  loaded: false,
  children: [],
  paths: [
    '/Users/milan/Dev/first-app/node_modules',
    '/Users/milan/Dev/node_modules',
    '/Users/milan/node_modules',
    '/Users/node_modules',
    '/node_modules'
  ]
}
```

🔑 Module Properties Explained

Property	Description
`id`	Identifier for the module (‘.’ for main module)
`path`	Directory path of the module
`exports`	Object containing exported values (initially empty)
`filename`	Full path to the module file
`loaded`	Whether the module has finished loading
`children`	Array of modules required by this module
`paths`	Paths where Node looks for modules

❌ Module is NOT Global

Try This

```
console.log(global.module);
```

Result:

```
milan@first-app ~\ node test.js
undefined
milan@first-app ~\
```

Why?

`module` is **not a global object**! It's scoped to each module file.

🔄 How Node Keeps Scope Private

The Module Wrapper Function

At runtime, your code is wrapped in a function:

```
(function(exports, require, module, __filename, __dirname) {
    // Your code here
    var message = 'test';
    console.log(message);
})
```

This is called an **IIFE** (Immediately Invoked Function Expression)

What This Means

The following are **NOT global**, but **local to each module**:

- ✅ `exports`
- ✅ `require`
- ✅ `module`
- ✅ `__filename`
- ✅ `__dirname`

📍 Special Module Variables

`__filename` and `__dirname`

These special variables are available in every module:

```
console.log(__filename);
console.log(__dirname);
```

Output:

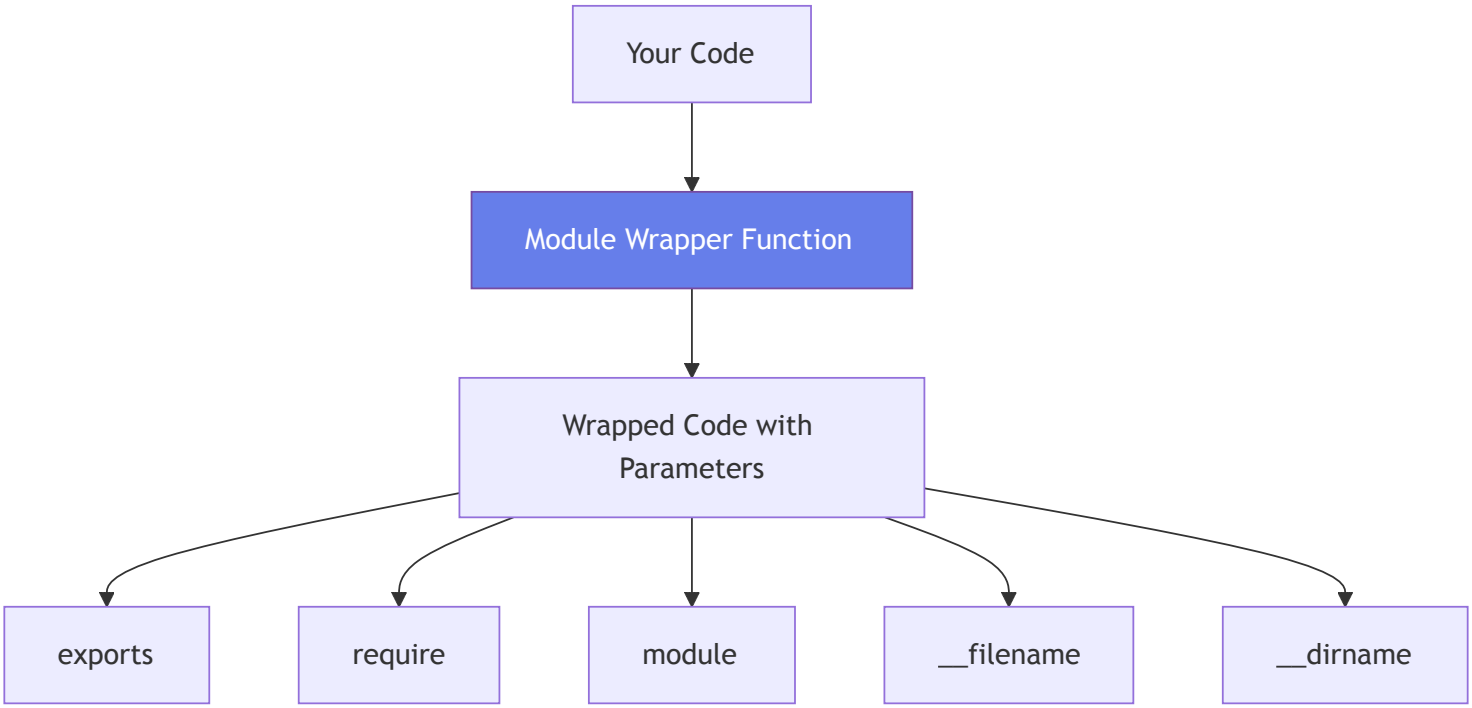
```
/Users/milan/Dev/first-app/test.js
/Users/milan/Dev/first-app
```

Usage

- `__filename` - Full path to the current file
- `__dirname` - Directory path containing the current file

Great for working with paths relative to your module!

🎨 Visualizing the Module Wrapper



🔑 Key Insights

Understanding the Wrapper

1. **Why variables are private**: They're inside a function scope
2. **Why `require` works**: It's passed as a parameter
3. **Why `module.exports` works**: `module` is a parameter
4. **Why `__dirname` exists**: It's passed by Node.js

The Benefits

✅ **Encapsulation** - Variables don't leak to global scope
✅ **No conflicts** - Each module has its own scope
✅ **Clean code** - Explicit imports and exports
✅ **Maintainability** - Easy to understand dependencies

💡 Module Best Practices

DO ✅

- Keep modules focused on a single responsibility
- Use descriptive module names
- Export only what's necessary
- Document what your module does

DON'T ❌

- Create huge monolithic modules
- Export too many things
- Mix unrelated functionality
- Forget to document exports

🧪 Try It Yourself

Exercise

Create a file and experiment with the module object:

```
// test.js
console.log('Module ID:', module.id);
console.log('Module filename:', __filename);
console.log('Module dirname:', __dirname);
console.log('Module exports:', module.exports);
```

Questions to explore:

1. What is the `id` of your main module?
2. Where does Node.js look for modules (check `paths`)?
3. What's in `module.exports` before you add anything?

➡️ What's Next?

Now that you understand **what** modules are and **how** they work internally, let's learn how to **create and use** your own modules!

🏠 Course Home | 📘 Chapter 2 Home

← Previous: Global Objects | Next: Creating Modules →