02-Node-Module-System/03-creating-modules.md

🛠️ Creating and Loading Modules

📝 Creating Your First Module

Example: Logger Module

Let’s create a simple logging module.

Create `logger.js`:

```
var url = "http://mylogger.io/log";

function log(message) {
  // Send HTTP request (simplified)
  console.log(message);
}
```

This is a simple module, but we can’t use it in other files yet!

📤 Exporting from a Module

Making Functions Available

To use the `log` function in other files, we must **export** it:

In `logger.js`:

```
var url = "http://mylogger.io/log";

function log(message) {
  console.log(message);
}

// Export the log function
module.exports.log = log;
```

⚠️ Implementation Details

What Should You Export?

```
// ❌ BAD: Exporting implementation details
module.exports.log = log;
module.exports.logurl = url; // URL is an implementation detail!
```

```
// ✅ GOOD: Only export what's necessary
module.exports.log = log;
```

Best Practice

Don’t export implementation details! Only export the public API.

- ✅ Export: Functions, classes that others need
- ❌ Don’t export: Internal variables, helper functions

📥 Loading a Module with `require()`

Using Your Module

In `app.js`:

```
// Load the module (without assignment)
require("./logger");

// Better: Assign to a variable
var logger = require("./logger");
console.log(logger);
```

Run it:

```
milan@les2~\ node app.js
{ log: [Function: log], logurl: 'http://mylogger.io:log' }
milan@les2~\
```

What Happened?

`require()` returns the `module.exports` object from `logger.js`!

🎯 Using the Exported Function

Calling the Log Function

```
const logger = require("./logger");
logger.log("message");
```

Output:

```
milan@les2 ~\ node app.js
message
milan@les2 ~\
```

🔒 `var` vs `const` for require()

The Problem with `var`

```
var logger = require("./logger");
logger = 1; // ⚠️ Oops! Accidentally reassigned
logger.log("message"); // 💥 TypeError!
```

Result:

```
TypeError: logger.log is not a function
```

The Solution: Use `const`

```
const logger = require("./logger");
logger = 1; // ❌ Error at compile time, not runtime!
logger.log("message");
```

Best Practice

✅ Always use `const` for `require()`

- Prevents accidental reassignment
- Catches errors earlier
- More predictable code

🎁 Exporting a Single Function

Better Export Pattern

Instead of exporting an object with a function, export the function directly:

In `logger.js`:

```
var url = "http://mylogger.io:log";

function log(message) {
  console.log(message);
}

// Export the function directly
module.exports = log;
```

In `app.js`:

```
const log = require("./logger");
log("message"); // Cleaner!
```

Benefits

- ✅ Simpler API
- ✅ More intuitive to use
- ✅ Less typing

📝 Quick Refactoring Tip

VS Code Shortcut

Rename `logger.log` to just `log`:

1. Select `logger` variable
2. Press **F2** (or **Fn + F2**)
3. Type new name: `log`
4. All instances renamed!

Before:

```
const logger = require("./logger");
logger.log("message");
```

After:

```
const log = require("./logger");
log("message");
```

🔄 Understanding `exports` Shorthand

Two Ways to Export

Remember the module wrapper function?

```
(function (exports, require, module, __filename, __dirname) {
```

```
  // Your code
});
```

`exports` is a **reference** to `module.exports`!

This Works ✅

```
module.exports.log = log;
// OR
exports.log = log; // Same thing!
```

This Does NOT Work ❌

```
exports = log; // ❌ Breaks the reference!
```

Why?

`exports` is just a reference to `module.exports`. Reassigning `exports` breaks that reference!

📋 Module Export Patterns

Pattern 1: Export Object with Methods

```
// logger.js
function log(message) {
  console.log(message);
}
function error(message) {
```

```
  console.error(message);
}

module.exports.log = log;
module.exports.error = error;

// OR shorthand
exports.log = log;
exports.error = error;
```

Usage:

```
const logger = require("./logger");
logger.log("Info");
logger.error("Error!");
```

Pattern 2: Export Single Function

```
// logger.js
function log(message) {
  console.log(message);
}

module.exports = log;
```

Usage:

```
const log = require("./logger");
log("Message");
```

Pattern 3: Export Class

```
// logger.js
class Logger {
  log(message) {
    console.log(message);
  }
  error(message) {
    console.error(message);
  }
}

module.exports = Logger;
```

Usage:

```
const Logger = require("./logger");
const logger = new Logger();
logger.log("Message");
```

⚙️ Tool Recommendation: JSHint

Code Quality Tool

Install JSHint for error checking:

```
npm install -g jshint
```

Run it on your files:

```
jshint app.js
```

JSHint will catch common mistakes like:

- Using `var` instead of `const`
- Undefined variables
- Unused variables
- And more!

🎯 Best Practices Summary

✅ DO	❌ DON'T
Use `const` for require()	Use `var` for require()
Export only public API	Export implementation details
Use `module.exports =`	Use `exports =` (breaks reference)
Give modules clear names	Use vague names
Use `./` for local modules	Omit `./` for local modules

🧪 Lab Exercise

Create Your Own Module

1. Create a `calculator.js` module
2. Add functions: `add`, `subtract`, `multiply`, `divide`
3. Export them
4. Use them in `app.js`

Bonus: Try both export patterns and see which you prefer!

🏠 Course Home | 📘 Chapter 2 Home

← Previous: Module System | Next: Built-in Modules →