Chapter 2: Node.js Core Modules 🛠️

Table of Contents

File System Module (fs) 📁

Think of the **fs module** as your **computer's file manager** - just like how you organize files in folders on your phone or laptop!

What is fs? 🤔

The `fs` module is like having a **smart assistant** who can:

- 📖 Read files (like opening a book)
- ✍️ Write files (like writing in a notebook)
- 📁 Create folders (like organizing your room)
- 🗑️ Delete files (like cleaning up)
- 📋 List files (like checking what's in your bag)

Basic fs Operations 🎯

```javascript
// Import the fs module (like getting your file manager app)
const fs = require('fs');

// Reading a file (like opening a book to read)
fs.readFile('data.txt', 'utf8', (err, data) => {
    if (err) {
        console.error('❌ Error reading file:', err);
        return;
    }
    console.log('📖 File content:', data);
});

// Writing a file (like writing in a diary)
const content = 'Hello from Node.js! 🚀';
fs.writeFile('output.txt', content, (err) => {
    if (err) {
        console.error('❌ Error writing file:', err);
        return;
    }
    console.log('✅ File written successfully!');
});
```

Real-life Analogy: Just like how you can read a book (readFile) or write in your diary (writeFile), the fs module lets you read and write computer files!

Path Module (path) 🗺️

Think of the **path module** as a **smart GPS system** for your computer files!

What is path? 🧭

The `path` module helps you:

- 🗺️ Build correct file paths (like giving proper directions)
- 📂 Join folder names (like connecting roads)
- 🏷️ Get file names and extensions (like reading street signs)
- 🔄 Work on different operating systems (Windows, Mac, Linux)

Path Module Examples 🎯

```
const path = require('path');

// Joining paths (like connecting roads)
const fullPath = path.join(__dirname, 'data', 'users.txt');
console.log('🗺️ Full path:', fullPath);

// Getting file information
const filePath = '/home/user/documents/report.pdf';
console.log('📄 File name:', path.basename(filePath));        // report.pdf
console.log('📂 Directory:', path.dirname(filePath));          // /home/user/documents
console.log('🏷️ Extension:', path.extname(filePath));         // .pdf
console.log('📋 Name without extension:', path.basename(filePath, '.pdf')); // report

// Resolving relative paths (like finding shortcuts)
const absolutePath = path.resolve('./data/file.txt');
console.log('📍 Absolute path:', absolutePath);
```

Real-life Analogy: Just like how Google Maps helps you find the best route, the path module helps Node.js find the correct file locations!

OS Module 💻

Think of the **OS module** as your **computer's health checkup report**!

What is OS? 💻

The os module gives you information about:

- 💾 Memory usage (like checking your phone's storage)
- 💻 CPU information (like knowing your phone's processor)
- ⏱ System uptime (like knowing how long your phone has been on)
- 👤 User information (like knowing who's using the device)
- 🌐 Network interfaces (like knowing your WiFi details)

OS Module Examples 🎯

```javascript
const os = require('os');

// System information
console.log('💻 Platform:', os.platform());          // win32, darwin, linux
console.log('🏗 Architecture:', os.arch());            // x64, arm64
console.log('💾 Total Memory:', os.totalmem(), 'bytes');
console.log('📱 Free Memory:', os.freemem(), 'bytes');
console.log('⏰ Uptime:', os.uptime(), 'seconds');

// User information
console.log('👤 Username:', os.userInfo().username);
console.log('🏠 Home Directory:', os.homedir());

// CPU information
console.log('🧠 CPU Cores:', os.cpus().length);
console.log('📊 CPU Model:', os.cpus()[0].model);

// Network interfaces
console.log('🌐 Network Interfaces:', Object.keys(os.networkInterfaces()));
```

Real-life Analogy: Just like how a doctor checks your vital signs, the OS module checks your computer's vital information!

Events Module and EventEmitter 📡

Think of **EventEmitter** as a **smart notification system** - like WhatsApp notifications or school announcements!

What are Events? 🔔

Events are like **signals** that something happened:

- 📱 Phone ringing (incoming call event)
- 🔔 WhatsApp message (new message event)
- 📢 School bell (class change event)
- 🚗 Car horn (warning event)

EventEmitter Basics 🎯

```javascript
const EventEmitter = require('events');

// Create an event emitter (like setting up a notification system)
const myEmitter = new EventEmitter();

// Listen for events (like waiting for notifications)
myEmitter.on('message', (data) => {
    console.log('📨 New message received:', data);
});

myEmitter.on('error', (error) => {
    console.log('❌ Error occurred:', error);
});

// Emit events (like sending notifications)
myEmitter.emit('message', 'Hello from EventEmitter!');
myEmitter.emit('message', 'Another message!');
```

Real-world Example: School Bell System 🏫

```javascript
const EventEmitter = require('events');

class SchoolBell extends EventEmitter {
    constructor() {
        super();
        this.isRecess = false;
    }

    startClass() {
        this.isRecess = false;
        this.emit('classStart', 'Math Class');
        console.log('🔔 Class started!');
    }

    startRecess() {
        this.isRecess = true;
        this.emit('recessStart', 'Lunch Break');
        console.log('🎾 Recess started!');
    }

    emergency() {
        this.emit('emergency', 'Fire Drill');
        console.log('🚨 Emergency!');
    }
}

// Using the school bell
const schoolBell = new SchoolBell();
```

```
// Listen for events
schoolBell.on('classStart', (subject) => {
    console.log(`📚 Time for ${subject}!`);
});

schoolBell.on('recessStart', (breakType) => {
    console.log(`🍎 Time for ${breakType}!`);
});

schoolBell.on('emergency', (type) => {
    console.log(`🚨 ${type} - Everyone evacuate!`);
});

// Trigger events
schoolBell.startClass();
schoolBell.startRecess();
schoolBell.emergency();
```

Real-life Analogy: Just like how your phone can receive different types of notifications (messages, calls, alerts), EventEmitter can handle different types of events!

HTTP Module 🌐

Think of the **HTTP module** as building your own **mini restaurant** where you serve web pages!

What is HTTP? 🍽️

HTTP is like the **language** that web browsers and servers use to communicate:

- 🌐 Browser asks for a webpage (HTTP request)
- 💻 Server sends back the webpage (HTTP response)
- 📱 Just like ordering food at a restaurant!

Basic HTTP Server 🎯

```
const http = require('http');

// Create a server (like opening a restaurant)
const server = http.createServer((req, res) => {
    // Log the request (like noting down customer orders)
    console.log(`${req.method} ${req.url}`);

    // Set response headers (like preparing the dining table)
    res.writeHead(200, { 'Content-Type': 'text/html' });

    // Send different responses based on URL (like serving different dishes)
    if (req.url === '/') {
        res.end(`
            <html>
```

```
                <head><title>My Restaurant</title></head>
                <body>
                    <h1>🍽️ Welcome to My Node.js Restaurant!</h1>
                    <p>What would you like to order?</p>
                    <ul>
                        <li><a href="/menu">📋 Menu</a></li>
                        <li><a href="/about">ℹ️ About Us</a></li>
                        <li><a href="/contact">📞 Contact</a></li>
                    </ul>
                </body>
            </html>
        `);
    } else if (req.url === '/menu') {
        res.end(`
            <html>
                <head><title>Menu</title></head>
                <body>
                    <h1>📋 Our Menu</h1>
                    <ul>
                        <li>🍕 Pizza - $10</li>
                        <li>🍔 Burger - $8</li>
                        <li>🍜 Noodles - $12</li>
                    </ul>
                    <a href="/">🏠 Back to Home</a>
                </body>
            </html>
        `);
    } else {
        // 404 - Page not found (like customer asking for dish we don't have)
        res.writeHead(404, { 'Content-Type': 'text/html' });
        res.end(`
            <html>
                <head><title>404 - Not Found</title></head>
                <body>
                    <h1>❌ 404 - Page Not Found</h1>
                    <p>Sorry, this page doesn't exist!</p>
                    <a href="/">🏠 Back to Home</a>
                </body>
            </html>
        `);
    }
});

// Start the server (like opening the restaurant)
const PORT = 3000;
server.listen(PORT, () => {
    console.log(`🍽️ Restaurant is open! Visit: http://localhost:${PORT}`);
    console.log(`📱 Server running on port ${PORT}`);
});
```

Real-life Analogy: Just like how a restaurant serves different dishes based on customer orders, an HTTP server serves different web pages based on the URL!

Reading/Writing Files Asynchronously vs Synchronously ⏱️

Think of this like **ordering food at a restaurant**:

Synchronous (Blocking) 🚫

```
const fs = require('fs');

// Synchronous reading (like waiting in line at a restaurant)
console.log('🍽️ Ordering food...');
const data = fs.readFileSync('menu.txt', 'utf8');  // Everything stops here!
console.log('📖 Menu:', data);
console.log('✅ Order complete!');  // This waits until file is read
```

What happens: Everything stops and waits for the file to be read completely.

Asynchronous (Non-blocking) ✅

```
const fs = require('fs');

// Asynchronous reading (like ordering online and doing other things)
console.log('🍽️ Ordering food...');
fs.readFile('menu.txt', 'utf8', (err, data) => {
    if (err) {
        console.error('❌ Error:', err);
        return;
    }
    console.log('📖 Menu:', data);
    console.log('✅ Order complete!');
});
console.log('📱 Checking phone while waiting...');  // This runs immediately!
console.log('🎮 Playing games...');               // This also runs immediately!
```

What happens: The file reading happens in the background while other code continues to run.

Comparison Table 📊

Aspect	Synchronous	Asynchronous
Speed	⏰ Slower	⚡ Faster
Blocking	🚫 Blocks everything	✅ Non-blocking
Use Case	📝 Simple scripts	🌐 Web applications
Example	Reading config files	Handling web requests

Real-life Analogy:

- **Synchronous** = Standing in line at a bank (everyone waits)
- **Asynchronous** = Online banking (you can do other things while waiting)

Using Callbacks with fs 📞

Think of **callbacks** as **delivery notifications** - you order food and get notified when it's ready!

What are Callbacks? 📱

Callbacks are functions that run **after** something completes:

- 📞 Like getting a call when your food is ready
- 📱 Like getting a notification when your download is complete
- 🔔 Like getting an alert when your friend messages you

Callback Examples 🎯

```
const fs = require('fs');

// Basic callback example
fs.readFile('data.txt', 'utf8', (err, data) => {
    if (err) {
        console.error('❌ Error reading file:', err);
        return;
    }
    console.log('✅ File read successfully:', data);
});

// Writing file with callback
const content = 'Hello from Node.js! 🚀';
fs.writeFile('output.txt', content, (err) => {
    if (err) {
        console.error('❌ Error writing file:', err);
        return;
    }
    console.log('✅ File written successfully!');
});

// Checking if file exists
fs.access('data.txt', (err) => {
    if (err) {
        console.log('❌ File does not exist');
        return;
    }
    console.log('✅ File exists!');
});
```

Real-world Example: File Manager 📁

```
const fs = require('fs');
const path = require('path');

class FileManager {
    constructor() {
        this.baseDir = __dirname;
    }

    // Create a file
    createFile(filename, content, callback) {
        const filePath = path.join(this.baseDir, filename);

        fs.writeFile(filePath, content, (err) => {
            if (err) {
                callback(`❌ Error creating file: ${err.message}`);
                return;
            }
            callback(`✅ File '${filename}' created successfully!`);
        });
    }

    // Read a file
    readFile(filename, callback) {
        const filePath = path.join(this.baseDir, filename);

        fs.readFile(filePath, 'utf8', (err, data) => {
            if (err) {
                callback(`❌ Error reading file: ${err.message}`);
                return;
            }
            callback(`📖 File content: ${data}`);
        });
    }

    // List all files
    listFiles(callback) {
        fs.readdir(this.baseDir, (err, files) => {
            if (err) {
                callback(`❌ Error listing files: ${err.message}`);
                return;
            }
            callback(`📋 Files in directory: ${files.join(', ')}`);
        });
    }
}

// Using the FileManager
const fileManager = new FileManager();

// Create a file
fileManager.createFile('test.txt', 'Hello from FileManager! 🚀', (result) => {
    console.log(result);
```

```
        // Read the file
        fileManager.readFile('test.txt', (result) => {
            console.log(result);

            // List all files
            fileManager.listFiles((result) => {
                console.log(result);
            });
        });
    });
```

Real-life Analogy: Just like how you get a notification when your food delivery arrives, callbacks notify you when file operations are complete!

Summary 📝

What We Learned:

☑ **fs module** - Your computer's file manager
☑ **path module** - Smart GPS for file locations
☑ **OS module** - Computer's health checkup report
☑ **EventEmitter** - Smart notification system
☑ **HTTP module** - Building your own web restaurant
☑ **Synchronous vs Asynchronous** - Standing in line vs online ordering
☑ **Callbacks** - Delivery notifications for operations

Key Analogies:

- 📁 **fs** = File manager app
- 🗺 **path** = GPS system
- 🏥 **OS** = Health checkup
- 📡 **EventEmitter** = WhatsApp notifications
- 🍽 **HTTP** = Restaurant serving web pages
- ⏱ **Async/Sync** = Online vs in-line ordering
- 📞 **Callbacks** = Delivery notifications

Practice Exercise:

1. Create a simple file manager that can create, read, and list files
2. Build a basic HTTP server that serves different pages
3. Use EventEmitter to create a simple chat system
4. Experiment with both synchronous and asynchronous file operations

Ready for Chapter 3? Let's explore Asynchronous Programming in Node.js! 🚀