FS MODULE

Certainly, let's dive deep into the Node.js fs (File System) module.

1. Introduction

- The fs module provides a set of functions for interacting with the file system in Node.js.
- It allows you to read, write, create, delete, and manipulate files and directories.
- It's essential for many Node.js applications, from simple file handling to complex data storage and processing.

2. Core Concepts

- Asynchronous vs. Synchronous Operations:
  - Asynchronous: Most fs methods have asynchronous counterparts (e.g., readFile, writeFile). These methods accept a callback function that will be executed once the operation completes. This prevents blocking the main thread, making your application more responsive.
  - Synchronous: Synchronous methods (e.g., readFileSync, writeFileSync) block the execution until the operation is finished. Use them with caution, as they can significantly impact performance, especially with large files or multiple file operations.
- File Paths:
  - fs methods typically use file paths to specify the location of files and directories.
  - You can use:
    - Absolute paths: Start with a root directory (e.g., /home/user/, C:\Users\).
    - Relative paths: Start from the current directory (e.g., ./data/file.txt, ../config.json).
- Error Handling:
  - Most fs methods accept a callback function that receives an error object as the first argument.
  - Always include error handling to gracefully handle potential issues like file not found, permission denied, etc.

3. Key Functions

- Reading Files:
    - fs.readFile(path, options, callback): Reads the contents of a file asynchronously.
    - fs.readFileSync(path, options): Reads the contents of a file synchronously.
- Writing[1] Files:
    - fs.writeFile(path, data, options, callback): Writes data to a file asynchronously.
    - fs.writeFileSync(path, data, options): Writes data to a file synchronously.
- Creating/Removing Directories:
    - fs.mkdir(path, options, callback): Creates a directory.
    - fs.rmdir(path, options, callback): Removes a directory.
- Reading Directory Contents:
    - fs.readdir(path, options, callback): Reads the contents of a directory.
- Renaming/Moving Files/Directories:
    - fs.rename(oldPath, newPath, callback): Renames a file or directory.
- Deleting Files:
    - fs.unlink(path, callback): Deletes a file.
- File/Directory Information:
    - fs.stat(path, callback): Gets file or directory statistics (e.g., size, modification time).

4. Example

JavaScript

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

// Read a file asynchronously

```
fs.readFile('data.txt', 'utf-8', (err, data) => {
  if (err) {
    console.error('Error reading file:', err);
  } else {
    console.log(data);
  }
```

```
});
```

// Write to a file synchronously

```
fs.writeFileSync('output.txt', 'Hello from Node.js!');
```

// Create a directory

```
fs.mkdir('newDirectory', { recursive: true }, (err) => {
  if (err) throw err;
  console.log('Directory created successfully.');
});
```

5. Best Practices

- Use Asynchronous Methods Whenever Possible: Avoid blocking the main thread by using asynchronous methods whenever feasible.
- Handle Errors Gracefully: Implement proper error handling to prevent unexpected behavior.
- Use Promises (Optional): For cleaner asynchronous code, consider using Promises with the fs/promises module.
- Security: Be mindful of security considerations, such as validating user input and ensuring proper file permissions.

6. fs/promises Module

- The fs/promises module provides a set of Promise-based versions of the fs module's methods.
- This can lead to more concise and readable code.

JavaScript

```
const { readFile, writeFile } = require('fs/promises');

async function readAndWrite() {
  try {
    const data = await readFile('data.txt', 'utf-8');
    console.log(data);
    await writeFile('output.txt', data);
  } catch (err) {
    console.error('Error:', err);
  }
}
```

```
readAndWrite();
```

This tutorial provides a comprehensive overview of the Node.js `fs` module. By effectively using the `fs` module, you can build robust and efficient Node.js applications that interact with the file system in various ways.