Cours Universitaire : Introduction à Node.js et ses Modules Natifs

Votre Nom

Contents

1 Introduction

Node.js est un environnement d'exécution JavaScript côté serveur qui permet d'exécuter du code JavaScript en dehors d'un navigateur. Ce cours a pour objectif de vous familiariser avec les bases de Node.js, en particulier l'utilisation des modules natifs, la manipulation de fichiers, la méthode `open` et l'utilisation des promesses.

2 Prérequis

- Node.js (version 14.x ou supérieure) installé sur votre machine
- Un éditeur de texte (VS Code recommandé)
- Connaissances intermédiaires en JavaScript (ES6+)

3 Modules en Node.js

Node.js utilise un système de modules pour organiser et réutiliser le code. Il existe deux systèmes de modules principaux : CommonJS et ES Modules.

3.1 CommonJS (require)

C'est le système de modules traditionnel de Node.js.

Exemple - CommonJS

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

3.2 ES Modules (import)

C'est le système de modules standardisé pour JavaScript, maintenant supporté par Node.js.

Exemple - ES Modules

```
import fs from 'fs';
// ou pour les promesses
import fs from 'fs/promises';
```

Pour utiliser ES Modules, vous devez soit utiliser l'extension de fichier `.mjs`, soit définir `"type": "module"` dans votre `package.json`.

4 Modules Natifs de Node.js

4.1 Module fs (File System)

Le module `fs` permet d'interagir avec le système de fichiers.

4.1.1 Utilisation synchrone

Exemple - Lecture synchrone

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

try {
  const data = fs.readFileSync('exemple.txt', 'utf8');
  console.log(data);
} catch (err) {
  console.error('Erreur de lecture :', err);
}
```

4.1.2 Utilisation asynchrone avec callbacks

Exemple - Lecture asynchrone avec callbacks

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

fs.readFile('exemple.txt', 'utf8', (err, data) => {
  if (err) {
    console.error('Erreur de lecture :', err);
    return;
  }
  console.log(data);
});
```

4.1.3 Utilisation avec promesses

Exemple - Lecture avec promesses

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

async function lireFichier() {
  try {
    const data = await fs.readFile('exemple.txt', 'utf8');
    console.log(data);
  } catch (err) {
    console.error('Erreur de lecture :', err);
  }
}

lireFichier();
```

4.1.4 Utilisation avec `open`

La méthode `open` est utilisée pour ouvrir un fichier et retourner un descripteur de fichier. Cela permet de lire ou écrire dans un fichier tout en gardant le fichier ouvert, ce qui est utile pour travailler avec de gros fichiers ou pour un contrôle plus fin de la gestion des fichiers.

Exemple - Lecture avec `open`

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

async function manipulationAvecOpen() {
  try {
    // Ouverture du fichier en mode lecture
    const file = await fs.open('exemple.txt', 'r');
    const contenu = await file.readFile({ encoding: 'utf8' });
    console.log('Contenu du fichier :', contenu);
    await file.close();
  } catch (err) {
    console.error('Erreur :', err);
  }
}

manipulationAvecOpen();
```

Dans cet exemple, nous avons ouvert le fichier avec la méthode `open`, lu son contenu, puis fermé le fichier à la fin. Cela permet un contrôle plus précis sur la gestion du fichier.

5 Manipulation de Fichiers

5.1 Écriture dans un fichier

Exemple - Écriture dans un fichier

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

async function ecrireFichier(chemin, contenu) {
  try {
    await fs.writeFile(chemin, contenu);
    console.log('Fichier  crit  avec succ s ');
  } catch (err) {
    console.error('Erreur d\'  criture  :', err);
  }
}

ecrireFichier('output.txt', 'Contenu du fichier');
```

5.2 Lecture d'un répertoire

Exemple - Lecture d'un répertoire

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

async function lireRepertoire(chemin) {
  try {
    const fichiers = await fs.readdir(chemin);
    console.log('Contenu du r pertoire :', fichiers);
  } catch (err) {
    console.error('Erreur de lecture du r pertoire :', err);
  }
}

lireRepertoire('.');
```

6 Gestion des Erreurs et Promesses

6.1 Try/Catch avec Async/Await

Exemple - Gestion des erreurs avec try/catch

```
async function operationFichier() {
  try {
    const data = await fs.readFile('fichier.txt', 'utf8');
    const newData = data.toUpperCase();
    await fs.writeFile('nouveauFichier.txt', newData);
    console.log('Op ration r ussie ');
  } catch (err) {
    console.error('Une erreur est survenue :', err);
  }
}
```

7 Exercice 1

Créez une application Node.js qui effectue les tâches suivantes :

1. Lit le contenu d'un répertoire spécifié.
2. Filtre les fichiers pour ne garder que ceux avec une extension '.txt'.

3. Lit le contenu de chaque fichier '.txt' en utilisant la méthode `open`.
4. Concatène le contenu de tous les fichiers dans un nouveau fichier `resultat.txt`.
5. Affiche des statistiques : nombre de fichiers traités, taille totale des données.

8 Exercice 2

Créez une application Node.js qui affiche l'arborescence d'un répertoire de manière récursive, en indiquant les sous-répertoires et fichiers.

9 Conclusion

Ce cours vous a introduit aux concepts fondamentaux de Node.js, notamment l'utilisation des modules natifs, la manipulation de fichiers et l'utilisation des promesses. Vous avez appris à :

- Importer des modules avec `require` et `import`
- Utiliser le module `fs` pour les opérations de fichiers
- Travailler avec les chemins de fichiers à l'aide du module `path`
- Gérer les opérations asynchrones avec les callbacks et les promesses
- Utiliser async/await pour une syntaxe plus claire

Ces compétences vous permettront de créer des applications Node.js robustes et efficaces pour diverses tâches côté serveur.

10 Ressources Supplémentaires

- Documentation officielle de Node.js
- Guide MDN sur les modules JavaScript
- javascript.info - Async/await