NodeJs

1. **Getting started**
2. **Introduction**

Node.js est une plateforme logicielle open-source basée sur le moteur JavaScript V8 de Google(**V8 aussi nous aide a compiler le code Js sur le navigateur**). Elle permet l'exécution de code JavaScript côté serveur, permettant ainsi de développer des applications web et des services réseau performants et évolutifs.

Une image contenant texte, capture d’écran, Police, logo

Description générée automatiquement

1. **Example nodeJs Application**

Pour exécuter ce code, sauvegardez-le en tant que fichier server.js et exécutez node server.js dans votre terminal.

const http = require('http');

//

const hostname = '127.0.0.1';

const port = 3001;

// createServer() de http crée un nouveau serveur HTTP et le renvoie.

const server = http.createServer((req, res) => {

  res.statusCode = 200;

  res.setHeader('Content-Type', 'text/plain');

  res.end('Hello World\n');

});

// Le serveur est configuré pour écouter sur le port et le nom d'hôte spécifiés.

server.listen(port, hostname, () => {

  console.log(`Server running at http://${hostname}:${port}/`);

});

1. **Combien de Js devez nous connaitre pour utiliser NodeJs**

Une image contenant texte

Description générée automatiquementUne image contenant texte

Description générée automatiquement

1. **Npm**

NPM (Node Package Manager) est un gestionnaire de paquets pour le développement JavaScript

**Une image contenant texte, capture d’écran, écran, noir

Description générée automatiquement**

Une image contenant texte, capture d’écran, écran, sombre

Description générée automatiquement

**Une image contenant texte

Description générée automatiquement**

**Une image contenant texte, capture d’écran, écran, intérieur

Description générée automatiquementUne image contenant texte

Description générée automatiquement**

1. **La différence entre le développement et la production**

**Une image contenant texte

Description générée automatiquement**Une image contenant texte

Description générée automatiquement

1. **NodeJs Avec Typescript**

Node.js est souvent utilisé en combinaison avec TypeScript pour améliorer la robustesse et la sécurité des applications back-end. L'utilisation de TypeScript avec Node.js peut aider à prévenir les erreurs de type courantes qui peuvent survenir lors du développement d'applications JavaScript de grande envergure.

**Une image contenant texte, capture d’écran, noir, écran

Description générée automatiquement**

**Une image contenant texte

Description générée automatiquement**

1. **NodeJS et WebAssembly**

[**https://nodejs.dev/fr/learn/nodejs-with-webassembly/**](https://nodejs.dev/fr/learn/nodejs-with-webassembly/)

1. **Manipulation Files**
2. **Introduction**

En Node.js, le module intégré fs (file system) permet de travailler avec le système de fichiers sur l'ordinateur exécutant l'application Node.js.

Quelques utilisations courantes du module fs dans Node.js incluent :

1. Lecture et écriture de fichiers : Le module fs fournit des fonctions pour lire et écrire des fichiers, y compris la création, la suppression, le déplacement et la copie de fichiers.
2. Travail avec les répertoires : Le module fs inclut des fonctions pour travailler avec les répertoires, comme la création et la suppression de répertoires, ainsi que la liste du contenu d'un répertoire.
3. Suivi des modifications de fichiers : Le module fs comprend des fonctions pour surveiller les fichiers et répertoires afin de réagir aux modifications en temps réel.
4. Modification des permissions des fichiers : Le module fs fournit des fonctions pour modifier les permissions des fichiers, telles que les permissions de lecture, d'écriture et d'exécution.

En Node.js, le module fs (file system) fournit de nombreuses fonctions utiles pour travailler avec des fichiers et des répertoires. Voici quelques-unes des fonctions les plus couramment utilisées :

1. fs.readFile() - Lit de manière asynchrone l'intégralité du contenu d'un fichier.
2. fs.writeFile() - Écrit de manière asynchrone des données dans un fichier, en remplaçant le fichier s'il existe déjà.
3. fs.appendFile() - Ajoute de manière asynchrone des données à un fichier, en créant le fichier s'il n'existe pas.
4. fs.existsSync() - Teste de manière synchrone si un fichier ou un répertoire existe.
5. fs.mkdir() - Crée de manière asynchrone un nouveau répertoire avec le nom spécifié.
6. fs.readdir() - Lit de manière asynchrone le contenu d'un répertoire.
7. fs.stat() - Récupère de manière asynchrone des informations sur un fichier ou un répertoire.
8. fs.watch() - Surveille un fichier ou un répertoire pour détecter les changements.
9. fs.createReadStream() - Crée un flux lisible pour un fichier.
10. fs.createWriteStream() - Crée un flux inscriptible pour un fichier.
11. fs.rmdir() - Supprime de manière asynchrone un répertoire.
12. fs.unlink() - Supprime de manière asynchrone un fichier.

Ce ne sont là que quelques-unes des nombreuses fonctions fournies par le module fs. La fonction la plus utile dépendra des exigences spécifiques de votre application. Cependant, fs.readFile() et fs.writeFile() sont deux des fonctions les plus couramment utilisées dans Node.js, car elles offrent un moyen simple de lire et d'écrire de manière asynchrone des fichiers.

1. **Statistique des fichiers NodeJS**

Chaque fichier est livré avec un ensemble de détails que nous pouvons inspecter en utilisant Node.js. En particulier, en utilisant la méthode stat() fournie par le module fs.

Vous l'appelez en passant un chemin de fichier, et une fois que Node.js obtient les détails du fichier, il appellera la fonction de rappel que vous passez, avec 2 paramètres : un message d'erreur, et les statistiques du fichier :

* **dev**: le numéro de périphérique du fichier ou du répertoire
* **ino**: le numéro d'i-noeud (inode) du fichier ou du répertoire
* **mode**: les permissions de fichier (dans un format octal)
* isFile: true ou false
* **isDirectory**: true ou false
* isSymbolicLink: les permissions de fichier (dans un format octal)
* **size**: la taille du fichier en octets
* **mtime**: la date de modification du fichier (un objet **Date**)

Une image contenant texte

Description générée automatiquement

**Une image contenant texte, capture d’écran, logiciel, Police

Description générée automatiquement**

1. **Travailler avec des chemins**

Une image contenant texte

Description générée automatiquement

1. **Travailler avec des descripteur de fichiers dans NodeJs**

En Node.js, les descripteurs de fichiers sont des identificateurs numériques qui représentent des fichiers ouverts par le programme. Ils permettent de manipuler un fichier ouvert en effectuant des opérations telles que la lecture, l'écriture, la fermeture ou la modification des options de fichier.

Pour ouvrir un fichier en mode lecture, écriture ou lecture/écriture, vous pouvez utiliser la méthode **fs.open()** qui retourne un descripteur de fichier. Par exemple, pour ouvrir un fichier en mode écriture, vous pouvez utiliser le code suivant :

fs.open('monfichier.txt', 'w', (err, fd) => {

  if (err) throw err;

  console.log('Fichier ouvert avec succès');

  // Manipulation du fichier avec le descripteur de fichier fd

});

Dans cet exemple, la méthode **fs.open()** prend en premier paramètre le chemin d'accès au fichier et en deuxième paramètre le mode d'ouverture (**'r'** pour la lecture, **'w'** pour l'écriture ou **'r+'** pour la lecture/écriture). La méthode prend également un callback qui sera appelé avec deux arguments : une erreur en cas d'échec de l'opération et un descripteur de fichier (**fd**) en cas de succès.

Une fois que vous avez un descripteur de fichier, vous pouvez utiliser d'autres méthodes **fs** telles que **fs.read()**, **fs.write()** ou **fs.close()** pour manipuler le fichier. Par exemple, pour écrire dans un fichier à l'aide d'un descripteur de fichier, vous pouvez utiliser le code suivant :

const fs = require('fs');

fs.open('monfichier.txt', 'w', (err, fd) => {

  if (err) throw err;

  const buffer = Buffer.from('Contenu à écrire dans le fichier');

  const offset = 0;

  const length = buffer.length;

  const position = null; // Écrit à partir de la fin du fichier

  fs.write(fd, buffer, offset, length, position, (err, bytesWritten, buffer) => {

    if (err) throw err;

    console.log(`${bytesWritten} octets écrits`);

  });

});

Dans cet exemple, la méthode **fs.write()** est utilisée pour écrire dans le fichier à l'aide du descripteur de fichier (**fd**). Le premier paramètre est le descripteur de fichier, le deuxième paramètre est le contenu à écrire sous forme de **Buffer**, le troisième paramètre est l'offset dans le **Buffer** à partir duquel écrire, le quatrième paramètre est la longueur des données à écrire et le cinquième paramètre est la position de départ pour écrire les données dans le fichier. Dans cet exemple, la position est définie sur **null** pour écrire à partir de la fin du fichier.

Une image contenant texte

Description générée automatiquement

1. **Lire des Fichiers avec NodeJS**

Une image contenant texte, capture d’écran, écran, noir

Description générée automatiquement

1. **Ecrire des fichiers dans NodeJS**

Une image contenant texte

Description générée automatiquement

Une image contenant texte, écran, capture d’écran, noir

Description générée automatiquement

1. **La difference entre fs.open et fs.readFile et fs.writeFile**

La principale différence entre **fs.readFile()** et **fs.open()** en Node.js est la façon dont ils manipulent les fichiers.

**fs.readFile()** est utilisé pour lire le contenu d'un fichier en entier de manière asynchrone, et renvoie le contenu sous forme de chaîne de caractères ou de tampon (**Buffer**). Cette méthode est pratique pour lire rapidement le contenu d'un fichier, mais elle n'offre pas autant de contrôle sur la manière dont le fichier est lu.

**fs.open()** est utilisé pour ouvrir un fichier en mode lecture, écriture ou lecture/écriture, et renvoie un descripteur de fichier qui peut ensuite être utilisé pour manipuler le fichier en effectuant des opérations telles que la lecture, l'écriture ou la fermeture du fichier. Cette méthode offre un contrôle plus fin sur la manière dont le fichier est manipulé, mais elle nécessite plus de code pour être utilisée.

En résumé, **fs.readFile()** est pratique pour lire rapidement le contenu d'un fichier, tandis que **fs.open()** offre un contrôle plus fin sur la manipulation du fichier. Le choix entre ces deux méthodes dépendra des besoins spécifiques de votre programme.

1. **Travailler avec les dossiers dans NodeJs**

Une image contenant texte

Description générée automatiquement

Une image contenant texte, écran, capture d’écran, téléphone

Description générée automatiquement

Une image contenant texte, écran, capture d’écran, téléphone cellulaire

Description générée automatiquement

Une image contenant texte, écran, capture d’écran, téléphone

Description générée automatiquement