Introdução ao Node.js

- Node.js é um ambiente de execução de JavaScript no lado do servidor.
- Ele permite executar JavaScript fora do navegador, usando o motor **V8** do Google Chrome.
- Com Node.js, podemos criar aplicativos de rede escaláveis e de alto desempenho.

Principais Características

- **Event-Driven:** Node.js usa um modelo orientado a eventos, onde ações são tratadas por eventos, permitindo alta eficiência.
- **Non-blocking I/O:** As operações de entrada/saída não bloqueiam a execução do programa, permitindo que ele lide com múltiplas operações simultaneamente.
- **Single-threaded:** Node.js usa um único thread, mas pode lidar com muitas conexões simultâneas graças ao seu modelo de I/O não bloqueante.

Por que usar Node.js?

- **Desempenho:** É ideal para aplicativos que requerem um grande número de conexões simultâneas, como servidores Web em tempo real.
- **Unificação de Linguagem:** Permite usar JavaScript tanto no Front-End quanto no Back-End.
- **Grande Ecossistema:** Possui um vasto repositório de pacotes e módulos através do npm (Node Package Manager).

Instalação do Node.js

- Acesse o Site Oficial: nodejs.org.
- Faça o download da versão LTS.
- Siga as instruções do instalador.

Verificação da Instalação

- Node: no cmd digite **`node -v`**
- NPM: **`npm -v`**

Criando o primeiro App utilizando Node.js

- Crie um arquivo: **`app.js`**
- Digite: **`console.log('Hello world, Node.js!');`**
- Rode o arquivo no terminal: **`node app.js`**

Criando um Servidor HTTP com Node.js

- Crie um arquivo: **`server.js`**
- Digite o código abaixo:

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

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello world, Node.js!\n');
});

server.listen(port, hostname, () => {
  console.log(`Servidor rodando em http://${hostname}:${port}/`);
});
```

- Rode o arquivo no terminal: **`node server.js`**
- Abra o navegador e digite: **`http://127.0.0.1:3000`**

O que fizemos:

1. Importamos o módulo HTTP do Node.js, que permite criar servidores Web.

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

2. Definimos o **Hostname** e a **Porta**, configurando o servidor para rodar no endereço 127.0.0.1 (localhost) na porta 3000.

   `const hostname = '127.0.0.1';`
   `const port = 3000;`

3. Usamos http.createServer para criar um novo servidor. A função callback define como o servidor deve responder a uma requisição.

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

4. Definimos o código de status HTTP para 200, indicando sucesso.
   `res.statusCode = 200;`

5. Definimos o cabeçalho de resposta para text/plain.
   `res.setHeader('Content-Type', 'text/plain');`

6. Enviamos a resposta "Hello, World!" e encerramos a resposta.
   `res.end(Hello world, Node.js!\n');`

7. Iniciamos o Servidor para escutar requisições no hostname e porta definidos.
   `server.listen(port, hostname, ( ) => { ... });`

Métodos Principais do Módulo FS

- O módulo FS (File System) é um módulo nativo do Node.js que permite interagir com o sistema de arquivos.
- Ele fornece uma API para ler, escrever, modificar e excluir arquivos e diretórios, além de outras operações relacionadas ao sistema de arquivos.

Por que usar o Módulo FS?

- **Gerenciamento de Arquivos:** Facilita a leitura e escrita de arquivos, importante para muitas aplicações que precisam armazenar e recuperar dados.
- **Automatização:** Permite criar scripts para automatizar tarefas relacionadas ao sistema de arquivos, como backup, manipulação de logs, etc.
- **Desenvolvimento de Aplicações:** É essencial para aplicações que necessitam lidar com uploads de arquivos, geração de relatórios, processamento de dados armazenados em arquivos, entre outros.

Uso do Módulo FS:

- O módulo FS pode ser usado em modo **síncrono** ou **assíncrono**, oferecendo flexibilidade dependendo das necessidades da aplicação.
- **Modo Síncrono:** Executa operações de forma bloqueante, ideal para scripts simples ou operações que não afetam o desempenho global da aplicação.
- **Modo Assíncrono:** Executa operações de forma não bloqueante, permitindo que outras partes do código sejam executadas simultaneamente, ideal para aplicações de maior escala ou que requerem alta performance.

Principais Métodos do Módulo FS

1. **fs.readFile**
    - Lê o conteúdo de um arquivo e retorna o resultado em um callback.
    - Usado para recuperar dados armazenados em arquivos.

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

fs.readFile('example.txt', 'utf8', (err, data) => {
  if (err) {
    console.error('Erro ao ler o arquivo:', err);
    return;
  }
  console.log('Conteúdo do arquivo:', data);
});
```

2. **fs.writeFile**
    - Escreve dados em um arquivo, criando o arquivo se ele não existir.
    - Usado para salvar informações ou gerar arquivos de saída.

```
const content = 'Escrevendo em um arquivo com Node.js';

fs.writeFile('example.txt', content, err => {
  if (err) {
    console.error('Erro ao escrever no arquivo:', err);
    return;
  }
  console.log('Arquivo escrito com sucesso');
});
```

3. **fs.appendFile**
    - Adiciona dados ao final de um arquivo existente.
    - Usado para atualizar arquivos de log ou adicionar entradas a um arquivo.

```
const content = '\nConteúdo adicional';

fs.appendFile('example.txt', content, err => {
  if (err) {
    console.error('Erro ao adicionar conteúdo no arquivo:', err);
    return;
  }
  console.log('Conteúdo adicionado com sucesso');
});
```

4. **fs.unlink**
    - Remove um arquivo do sistema de arquivos.
    - Usado para excluir arquivos desnecessários ou liberar espaço.

```
fs.unlink('example.txt', err => {
  if (err) {
    console.error('Erro ao deletar o arquivo:', err);
    return;
  }
  console.log('Arquivo deletado com sucesso');
});
```

5. **fs.readdir**
    - Lista todos os arquivos e diretórios dentro de um diretório especificado.
    - Usado para navegar e gerenciar diretórios (pastas).

```
fs.readdir('.', (err, files) => {
  if (err) {
    console.error('Erro ao ler o diretório:', err);
    return;
  }
  console.log('Conteúdo do diretório:', files);
});
```

6. **fs.mkdir**
    - Cria um novo diretório.
    - Necessário para organizar arquivos em estruturas de pastas.

```
fs.mkdir('novo_diretorio', { recursive: true }, err => {
  if (err) {
    console.error('Erro ao criar o diretório:', err);
    return;
  }
  console.log('Diretório criado com sucesso');
});
```

7. **fs.rm**
    - Remove um diretório.
    - Usado para excluir diretórios vazios ou limpar estruturas de diretórios.

```
fs.rm('novo_diretorio', { recursive: true }, err => {
  if (err) {
    console.error('Erro ao remover o diretório:', err);
    return;
  }
  console.log('Diretório removido com sucesso');
});
```

Síncrono vs Assíncrono, diferença prática

- **Síncrono:** As operações são executadas uma de cada vez, bloqueando o fluxo do programa até que a operação atual seja concluída.
- **Assíncrono:** As operações são executadas em paralelo, permitindo que o programa continue executando outras tarefas enquanto aguarda a conclusão da operação.

Quando usar Síncrono ou Assíncrono

- **Síncrono:** Use quando a simplicidade é mais importante que a performance, como em scripts de automação ou tarefas de inicialização.
- **Assíncrono:** Use quando a performance é crítica, como em servidores web ou aplicações que precisam manipular muitas requisições simultâneas.

Exemplo Comparativo

- Criamos um arquivo grande

```
const filePath = 'largefile.txt';
const lines = 1000000;

const writeStream = fs.createWriteStream(filePath);

writeStream.write('Linha 1\n');
for (let i = 2; i <= lines; i++) {
  writeStream.write(`Linha ${i}\n`);
}

writeStream.end(() => {
  console.log(`${lines} linhas escritas em ${filePath}`);
});
```

- Ler o arquivo de forma **Síncrona** (fs.readFileSync)

```
console.log('\n1. Iniciando leitura do arquivo síncrono...');

try {
  const data = fs.readFileSync('largefile.txt', 'utf8');
  // Mostra os primeiros 1.000 caracteres
  console.log(`\n2. Conteúdo do arquivo (síncrono):\n${data.slice(0, 1000)}`);

} catch (erro) {
  console.error(`\nErro ao ler o arquivo: ${erro}`);
}

console.log('\n3. Continua executando após a leitura síncrona...\n');
```

- Ler o arquivo de forma **Assíncrona** (fs.readFile)

```
console.log('\n1. Iniciando leitura do arquivo assíncrono...');

fs.readFile('largefile.txt', 'utf8', (erro, data) => {
  if (erro) {
    console.error(`\nErro ao ler o arquivo: ${erro}`);
    return;
  }

  // Mostra os primeiros 1.000 caracteres
  console.log(`\n2. Conteúdo do arquivo (assíncrono):\n${data.slice(0, 1000)}`);
});

console.log('\n3. Continua executando outras operações...');
```