07

Faça como eu fiz: Percorrendo diretórios

Durante esta aula, aprendemos como utilizar o método `fs.readfile()` que, como o próprio nome diz, é usado para ler um arquivo (*file*). Mas como podemos fazer para que o JavaScript percorra todo um diretório, vendo todos os arquivos que ele contém?

Para isso, vamos utilizar outro método, o `fs.promises.readdir()` (ler diretório). Ele funciona de uma forma parecida com o `readfile()`, com a diferença que, uma vez que o programa deve percorrer todos os arquivos de um diretório, as instruções que escrevemos para serem executadas apenas em um arquivo deverão ser executadas em todos. Como fazemos isso?

Acessando um diretório

Vamos refatorar a função `pegarArquivo()`, que é onde está acontecendo a lógica de leitura e serialização do arquivo de texto. Porém, agora a função será responsável por ler um diretório e **percorrer os arquivos deste diretório**, então já temos uma dica do que fazer: algum tipo de laço de repetição que passe por cada arquivo e execute a extração dos links:

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

async function pegarArquivo(caminho) {
 const caminhoAbsoluto = path.join(__dirname, '..', caminho);
 const encoding = 'utf-8';
 const arquivos = await fs.promises.readdir(caminhoAbsoluto, {
```

```
  console.log('arquivos', arquivos);
}
```

COPIAR CÓDIGO

O que está sendo feito na nova função `pegarArquivo()`:

1) Ela continua recebendo como parâmetro `caminho`; lembrando que este parâmetro se refere ao caminho do arquivo que informamos no terminal através da função `processaTexto()` no arquivo `cli.js`.

2) Veja que adicionamos um módulo importado no topo do arquivo, o `path`. Esse módulo é nativo do NodeJS e não precisamos instalar, apenas importar. Usamos o método `path.join()` para montar o **caminho absoluto** do diretório que queremos pesquisar, passando como parâmetro de `path.join(__dirname, '..', caminho)`. Se quiser saber mais sobre este módulo, confira a documentação do NodeJS (https://nodejs.org/api/path.html#path_path_join_paths).

- O primeiro parâmetro `__dirname` vai capturar o caminho absoluto do arquivo até a pasta atual;
- O segundo `'..'` utilizamos para voltar um nível na hierarquia de pastas (lembrando que a pasta `arquivos` onde está o arquivo de texto não está dentro de `src`);
- O terceiro, `caminho` é o **caminho relativo** que estamos passando via linha de comando.

3) No lugar de `fs.promises.readfile()` vamos trabalhar com `fs.promises.readdir()` passando dois parâmetros: o **caminho absoluto** que montamos no item 2 com o módulo `path`; o segundo parâmetro é um objeto que aceita algumas opções (https://nodejs.org/docs/latest/api/fs.html#fs_fspromises_readdir_path_options),

entre elas o `encoding` do texto. Não esqueça que estamos trabalhando com **promessas**, então usamos `await` antes do método.

> É possível suprimir o valor de uma chave de objeto caso seja exatamente o mesmo termo. Ou seja, `{ encoding: encoding }` (o nome da chave é `encoding` e estamos passando o valor da variável `encoding`) pode ser declarada na forma `{ encoding }` . Este é um padrão de sintaxe do JavaScript e o uso deste padrão é encorajado.

4) Para melhorar este teste, fiz uma cópia do arquivo `texto1.md` dentro da mesma pasta. Agora, ao executarmos o código acima com o comando `npm run cli ./arquivos/` (sem informar o arquivo), o `console.log('arquivos', arquivos)` deve retornar um array com os arquivos de texto existentes no diretório `arquivos : arquivos [ 'texto1 copy.md', 'texto1.md' ]` .

Agora já temos nosso array de arquivos. É importante frisar aqui que o método `readdir()` vai somente **listar os arquivos de um diretório** e retornar seus nomes; então ainda precisamos do método `readfile()` que usamos anteriormente para, aí sim, ler o **conteúdo do texto** de cada arquivo do array.

```
async function pegarArquivo(caminho) {
 const caminhoAbsoluto = path.join(__dirname, '..', caminho);
 const encoding = 'utf-8';
 const arquivos = await fs.promises.readdir(caminhoAbsoluto, { er
 const result = arquivos.map(async (arquivo) => {
   const localArquivo = `${caminhoAbsoluto}/${arquivo}`;
   const texto = await fs.promises.readFile(localArquivo, encodir
   return extraiLinks(texto);
 });
 console.log(result);
}
```

COPIAR CÓDIGO

Revisando o código acima:

1) Já que estamos trabalhando com arrays, vamos utilizar o método `.map()` para iterar o array de nomes de arquivos e nos devolver outro array, com os resultados.

2) Ainda precisamos ler cada um dos arquivos e extrair os links. Para isso, podemos reconstruir o caminho absoluto de cada arquivo a partir do caminho que já temos `/` o nome de cada um dos arquivos (que estamos recebendo a cada iteração do `map()` através do parâmetro `arquivo`. Concatenamos tudo isso em uma string usando *template strings* e o caminho absoluto até o arquivo (por exemplo `/home/juliana/Documents/alura/2299-lib-regex/arquivos//texto1.md`) está pronto para ser passado como parâmetro de `readFile()` e seu resultado guardado na constante `texto`.

3) Lembrando que ainda estamos dentro do *loop* do `.map()`, então precisamos retornar algum dado para o array de resultados. A função `extraiLinks()` que desenvolvemos pode ser executada agora, recebendo `texto` - da mesma forma que fizemos durante a aula.

4) Por enquanto vamos só checar se o array que esperamos receber ao final da iteração do `.map()` está correto, com `console.log(result)`.

Se você executar este código com o mesmo comando `npm run cli ./arquivos/`, deve receber o seguinte resultado:

```
[ Promise { <pending> }, Promise { <pending> } ]
(node:364138) UnhandledPromiseRejectionWarning: TypeError: Can
```

```
(... restante do erro suprimido)
```

COPIAR CÓDIGO

Recebemos um erro, mas também duas dicas: A primeira é que nosso array de resultados está recebendo dois `Promise { <pending> }`, o que significa que existem "promessas não resolvidas" em alguma parte do código. A linha seguinte dá a outra dica: `Cannot read property 'map' of undefined`, apesar de já termos passado o `async` para a função *callback* dentro do `.map()` e o `await` para o método `readFile()` que executa neste bloco.

> Aqui há uma distinção importante a fazer: enquanto a **função callback** dentro do `.map()` é uma função assíncrona (fizemos isso com o `async/await`), **a iteração em si, ou seja, o próprio .map(), não respeita esta assincronicidade**.

Partindo do princípio que o `.map()` vai receber sempre um array com um número X de nomes de arquivo, podemos concluir que temos aqui **uma ou várias promessas para serem concluídas**. Então envolvemos o `.map()` com o método do JavaScript feito justamente para trabalhar com arrays de promessas, o `Promise.all()`:

```
async function pegarArquivo(caminho) {
 const caminhoAbsoluto = path.join(__dirname, '..', caminho);
 const encoding = 'utf-8';
 const arquivos = await fs.promises.readdir(caminhoAbsoluto, { er
 const result = await Promise.all(arquivos.map(async (arquivo) =>
   const localArquivo = `${caminhoAbsoluto}/${arquivo}`;
   const texto = await fs.promises.readFile(localArquivo, encodir
   return extraiLinks(texto);
 }));
```

```
  console.log(result);
}
```

COPIAR CÓDIGO

Executando novamente o código com o mesmo comando `npm run cli ./arquivos/` , ainda recebemos um erro! Mas, em compensação, antes do erro já podemos ver o resultado do `console.log(result)` , um array de arrays com os links dos dois arquivos `.md` da pasta `arquivos` .

Quanto ao erro, podemos ver que falta alguma coisa no final da função: o **retorno do resultado**. Então podemos corrigir apenas trocando `console.log(result)` por `return result` :

```
async function pegarArquivo(caminho) {
  const caminhoAbsoluto = path.join(__dirname, '..', caminho);
  const encoding = 'utf-8';
  const arquivos = await fs.promises.readdir(caminhoAbsoluto, { er
  const result = await Promise.all(arquivos.map(async (arquivo) =>
    const localArquivo = `${caminhoAbsoluto}/${arquivo}`;
    const texto = await fs.promises.readFile(localArquivo, encodir
    return extraiLinks(texto);
  }));
  return result;
}
```

COPIAR CÓDIGO

Agora o resultado é o esperado: um array de arrays, contendo os links dos dois arquivos.

Finalizamos a refatoração com um bloco `try/catch` para capturar possíveis erros, utilizando a função `trataErro()` que já tínhamos desenvolvido:

```
async function pegarArquivo(caminho) {
 const caminhoAbsoluto = path.join(__dirname, '..', caminho);
 const encoding = 'utf-8';
 try {
   const arquivos = await fs.promises.readdir(caminhoAbsoluto, {
   const result = await Promise.all(arquivos.map(async (arquivo)
     const localArquivo = `${caminhoAbsoluto}/${arquivo}`;
     const texto = await fs.promises.readFile(localArquivo, encoc
     return extraiLinks(texto);
   }));
   return result;
 } catch (erro) {
   return trataErro(erro);
 }
}
```

COPIAR CÓDIGO

Opinião do instrutor

Leia todo o código acima com atenção e refaça os passos! Caso tenha alguma dúvida sobre algum dos métodos ou quais parâmetros eles recebem, não deixe de checar a documentação do NodeJS.