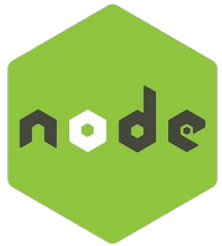
NodeJS



Prof. Celso Henrique Masotti

Você verá nesta aula:

Objetivo: diferenças entre processamento síncrono e assíncrono.

Programação
Programação Síncrona
Programação Assíncrona

Programação

Quando programamos com o nodeJS podemos solicitar o processamento de forma síncrona ou assíncrona. Por se tratar de um dos paradigmas principais desta plataforma, vejamos o que isto significa com muita atenção.

Programação

Computadores são assíncronos por padrão (default). Isto significa que coisas podem acontecer de maneira independente do fluxo do programa principal. Nos computadores atuais, cada programa roda por um tempo específico e então pára sua execução para permitir que outro programa execute um pouco. Isto acontece tão rápido que é quase impossível perceber e temos geralmente a falsa impressão de que o computador está de fato fazendo várias coisas ao MESMO tempo, quando na verdade ele está fazendo UMA coisa de cada vez, um pouco de cada.

Claro, isso tendo uma CPU em mente. A realidade para UMA CPU é essa.

Programação

Programas internamente usam interrupções, um sinal que é emitido para o processador quando o programa quer a atenção do sistema. Logo, é normal para os programas serem assíncronos: eles começam, são deixados processando sozinhos e quando algo novo requer a atenção do sistema, eles emitem esse aviso.

No entanto, normalmente as linguagens de programação são síncronas (executam o código em série, uma linha depois da outra) e algumas fornecem mecanismos de assincronicidade através de bibliotecas. C, Java, C#, PHP, Go, Ruby, Swift, Python e JavaScript. Todas elas são síncronas por padrão e várias delas permitem assincronicidade usando threads, criando novos fluxos do processo.

Programação

Em Node.js, por outro lado, as coisas são diferentes, graças ao Event Loop. Resumidamente, dependendo do tipo de processamento necessário, a atividade sai da thread principal e é enviada para um pool de threads em background automaticamente e só retorna ao event loop quando terminar.

Programação Síncrona

O código a seguir faz parte da programação síncrona. Basicamente, requereremos o módulo “fs” para criaremos um loop de cinco iterações, sendo que a cada iteração será criado um arquivo texto com o mesmo conteúdo "Hello Node.js!".

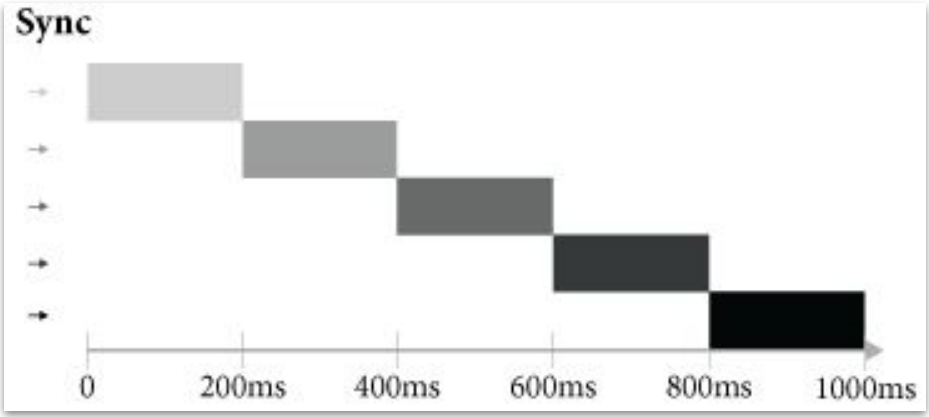
Neste exemplo fazemos uso da função “writeFileSync” que especifica qual método de processamento desejamos. Neste caso: síncrono.

```
1 var fs = require('fs');
2 for(var i = 1; i <= 5; i++) {
3     var file = "sync-txt" + i + ".txt";
4     fs.writeFileSync(file, "Hello Node.js!");
5     console.log("Criando arquivo: " + file);
6 }
```

Programação Síncrona

Se fôssemos criar um gráfico hipotético do processamento do algoritmo teríamos algo como a ilustração ao lado.

Os retângulos representam o período de processamento para a criação dos arquivos e um processamento só ocorre quando o anterior foi finalizado.



Programação Assíncrona

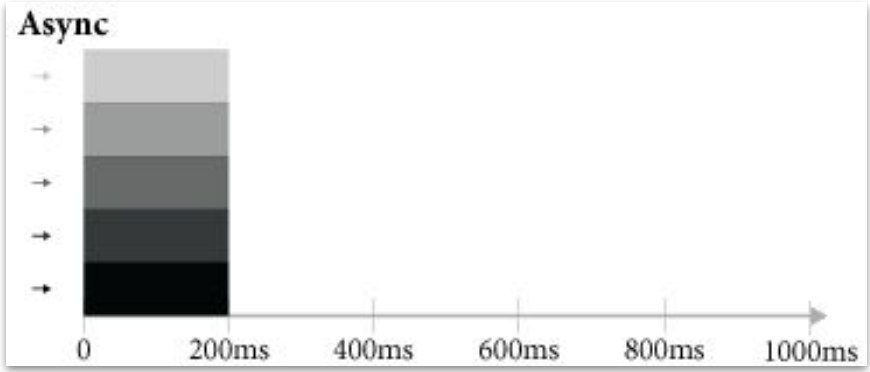
Agora “rodaremos” o mesmo programa, contudo, mudaremos para o processamento assíncrono, fazendo uso da função “writeFile”.

```
1 var fs = require('fs');
2 for(var i = 1; i <= 5; i++) {
3     var file = "async-txt" + i + ".txt";
4     fs.writeFile(file, "Hello Node.js!", function(err, out) {
5         console.log("Criando arquivo: " + file);
6     });
7 }
```

Programação Assíncrona

Se fôssemos observar o gráfico hipotético do processamento do algoritmo veríamos algo como a imagem ao lado.

Os blocos ainda representam a criação dos arquivos. Perceba que todos os arquivos, ao contrário do algoritmo anterior, foram criados simultaneamente.



Programação Assíncrona

A declaração “async function” define uma função assíncrona, que retorna um objeto “AsyncFunction”.

Síntaxe:

```
async function nome([param[, param[, ... param]]]) {

   instruções

}
```

sendo: "nome" o nome da função; "param" o nome de um parâmetro a ser passado para a função; "instruções" as instruções que compõem o corpo da função.

Obrigado

Profº. Celso Henrique Masotti