05

Criação do servidor web

Transcrição

Vamos nos aprofundar um pouco mais na plataforma Node de modo a criar um servidor web. Primeiramente, precisaremos de um responsável por receber as requisições do navegador, tratá-las e devolver uma resposta ao cliente. A boa notícia é que o Node, por padrão, disponibiliza um módulo específico para isso.

Um **módulo Node** é como uma biblioteca, ou seja, um conjunto de funcionalidades que ajuda a resolver determinadas tarefas. Sendo assim, no mundo Node, um arquivo qualquer com a extensão `.js` já é um módulo.

No nosso caso, teremos dois arquivos no projeto "casadocodigo":

- `olaMundoNode.js` , que criamos anteriormente
- `server.js` , que utilizaremos para criar nosso servidor

Durante o curso, iremos nos aprofundar ainda mais sobre os módulos e suas funcionalidades, portanto guarde essa informação!

Para utilizarmos o módulo HTTP disponibilizado pelo Node, precisaremos importá-lo. Isso é feito com a função `require()` , recebendo como parâmetro uma *string*, que é simplesmente o nome do módulo a ser importado. Atribuiremos a referência dessa função a uma constante (`const`) chamada `http` .

De posse dessa referência, executaremos o método `createServer()` , que retorna um objeto do tipo `Server` . Esse retorno será armazenado dentro de outra constante, chamada `servidor` .

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

const servidor  = http.createServer();
```

Quem trabalha com Java tem o costume de utilizar o **Tomcat**, o **JBoss** ou outros servidores. Normalmente, as aplicações desses servidores rodam na porta `8080` . Já o **Apache**, um servidor para o mundo PHP, normalmente roda na porta `80` . Ou seja, nesse momento precisamos definir em qual porta nosso servidor irá funcionar.

Com o método `servidor.listen()` , configuraremos a porta `3000` como sendo responsável pela execução do nosso servidor. Essa é a porta mais comumente utilizada no mundo Node, portanto seguiremos essa convenção.

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

const servidor  = http.createServer();
servidor.listen(3000);
```

Ainda precisamos definir o que esse servidor deverá fazer ao receber uma requisição. O método `createServer()` pode receber como parâmetro opcional uma função que será executada toda vez que o servidor receber uma requisição do cliente (`requestListener`). Essa função recebe dois parâmetros, `request` e `response` .

Portanto, criaremos uma função anônima (`function ()` recebendo esses dois parâmetros. De posse desses objetos, definiremos o que o Node precisará fazer na requisição e na resposta. Nosso objetivo é simplesmente devolver para o usuário uma *string* representando o HTML. Faremos isso com o método `end()` , recebendo tal conteúdo:

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

const servidor  = http.createServer(function (req, resp) {
    resp.end(`
        <html>
            <head>
                <meta charset="utf-8">
            </head>
            <body>
                <h1> Casa do Código </h1>
            </body>
        </html>
    `);
});
servidor.listen(3000);
```

No Prompt de Comando, usaremos `node server.js` para executar esse arquivo. No navegador, acessaremos http://localhost:3000 (http://localhost:3000). Como resposta, teremos a mensagem:

> Casa do Código

Ou seja, um HTML cujo corpo é somente o texto exibido. Isso significa que a requisição foi atendida satisfatoriamente pelo nosso servidor!

Lembre-se que a função que passamos para o método `createServer()` só será executada quando o servidor receber uma requisição do cliente. No mundo Java, e também em outras linguagens, funções que só são executadas dada a ocorrência de um evento são chamadas de "funções *callback*", e serão muito utilizadas na plataforma Node.

No navegador, atendemos a requisição para o endereço http://localhost:3000 (http://localhost:3000). E se quiséssemos criar uma URL http://localhost:3000/livros (http://localhost:3000/livros), por exemplo, para listar os livros da casa do código? Atualmente, acessando essa URL ou qualquer outra precedida por http://localhost:3000 (http://localhost:3000), teríamos sempre a mesma resposta "Casa do Código".

A seguir, aprenderemos como devolver informações diferentes em cada URL da nossa aplicação.