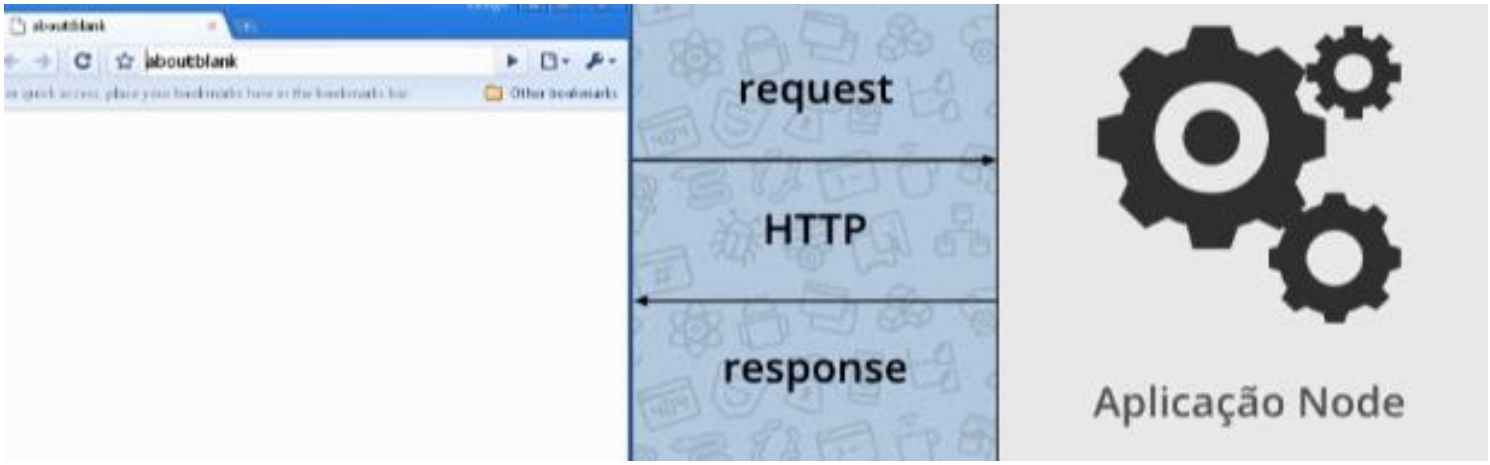
Node.js

O que é Node.js?



O que é Node.js?

- É uma poderosa plataforma de desenvolvimento de aplicações que utiliza JavaScript. Qualquer arquivo com sintaxe Javascript o Node.js consegue "executar".
- O Node.js trabalha de forma assíncrona, ou seja, ele não espera terminar uma tarefa para iniciar outra.
- **https://nodejs.org/**
- **https://www.w3schools.com/nodejs/**
- Diferente do PHP, onde temos PHP como linguagem server-side e o Apache como servidor web, com o Node.js escrevemos nosso código server-side e também o client-side, tudo em javascript.
- O Node.js usa uma arquitetura orientada a eventos fazendo com que as aplicações sejam leves e eficientes.

O que é Node.js?

- É possível desenvolver aplicações web e aplicativos nativos para Windows, Linux, OS X e mobile (usando o Ionic Framework).
- O Node.js tem uma forma de acessarmos as propriedades e funções utilizando o terminal do SO sem que seja necessário escrever e salvar códigos em um arquivo.
- **https://nodejs.org/api/repl.html**

O que é Node.js?

- O Node Inspector (**https://github.com/node-inspector/node-inspector**) permite debugar o código NodeJs de forma similiar ao que vemos em outras linguagens de programação.
- Sobre os editores de código, temos VIM, Sublime Text, VisualCode, WebStorm, EditorConfig.org.

O que é possível fazer com Node.js?

- Gerar páginas dinâmicas.
- Criar, abrir, ler, gravar excluir e fechar arquivos no servidor.
- Coletar dados de formulários.
- Adicionar, excluir, modificar dados em uma base de dados.

Estrutura do Node.js

- Arquivos Node.js contêm tarefas que serão executadas em determinados eventos.
- Um exemplo típico de evento seria alguém tentando **acessar uma porta** no servidor.
- Os arquivos Node.js devem ser iniciados no servidor antes de ter qualquer acesso.
- Arquivos Node.js têm extensão ".js“.

Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.

Download for Windows (x64)

8.12.0 LTS
Recommended For Most Users

Other Downloads | Changelog | API Docs

10.11.0 Current
Latest Features

Other Downloads | Changelog | API Docs

Or have a look at the Long Term Support (LTS) schedule.

Sign up for Node.js Everywhere, the official Node.js Monthly Newsletter.

1º exemplo com Node.js

- Depois de instalar o Node.js crie um arquivo com o nome exemplo1.js com o seguinte código:

(somente será executada qdo o servidor receber uma requisição do cliente – função callback)

```
var http = require('http');
http.createServer(function (req, res) {
   res.writeHead(200, {'Content-Type': 'text/html'});
   res.end('Alo GENTE!!!');
}).listen(3000);
```

Linha de comando do Node.js

- O Node.js deve ser inicializado pela linha de comando (abrir o Command Prompt do Windows digitando cmd no campo executar)
- Dentro do CMD navegue até a pasta onde está o arquivo .js e digite:

node exemplo1.js <*enter*>

- O prompt de comando vai ficar “congelado” o que significa que o programa está em execução

Testando o 1º exemplo

- Abra o navegador e digite o endereço:
- http://localhost:3000

Módulos

- Módulos são como bibliotecas JavaScript.
- São conjuntos de funções/funcionalidades que podem ser incluídas em sua aplicação.
- Alguns módulos já são incorporados no Node.js e estão disponíveis para uso na aplicação.
- Existem módulos que são incorporados manualmente, através de um processo de instalação.
- É possível também você criar seus próprios módulos e incorporá-los na aplicação.

Incluir Módulos

- Para incluir um módulo, usamos o comando require(), como no exemplo abaixo:

var http = require(‘http’);

- Agora a aplicação tem acesso ao módulo HTTP e podemos criar um servidor:

```
http.createServer(function (req, res) {
   res.writeHead(200, {'Content-Type': 'text/html'});
   res.end(Alo GENTE!');
}).listen(3000);
```

Alguns módulos existentes no Node.Js

const http = require('http');

var express = require('express');

É o framework web mais popular, e é a biblioteca subjacente para uma série de outros frameworks populares de Nodes. Por exemplo: Definir as configurações comuns da aplicação web, como a porta a ser usada para conexão e a localização dos modelos que são usados para renderizar a resposta.

var morgan = require('morgan');

Usado para geração de logs.

var compress = require('compression');

Compactar a resposta num gzip.

var bodyParser = require('body-parser');

É um módulo capaz de converter o *body* da requisição para vários formatos. Um desses formatos é *json.*

Módulo **HTTP** do Node.js

- O Node.js possui um módulo interno (nativo) chamado **HTTP** que permite transferir dados através do protocolo HTTP.
- Para incluir o módulo HTTP, usamos a função require():

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

Node.js como Servidor Web

- O módulo HTTP pode criar um servidor HTTP que ouve as portas do servidor e retorna uma resposta ao cliente.
- Usamos o método **createServer ()** para criar um servidor HTTP.
- A função passada para o método **http.createServer ()** será executada quando alguém tentar acessar o computador na porta 3000.

Cabeçalhos HTTP

- Se a resposta do servidor HTTP deverá ser exibida como HTML, é necessário incluir um cabeçalho HTTP com o tipo de conteúdo correto:

```
var http = require('http');
http.createServer( function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write(Alo Gente!');
  res.end();
}).listen(3000);
```

Tratando solicitações do cliente

- A função passada para o **http.createServer ()** possui um argumento **req** que representa a solicitação do cliente.
- Este objeto tem uma propriedade chamada "url" que contém a parte da URL que vem depois do nome do domínio:

```
var http = require('http');
http.createServer(function (req, res) {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write(req.url);
    res.end();
}).listen(3000);
```

Conceitos de ROTAS em Node.Js

- Aqui se faz necessário vermos o conceito de ROTAS.
- ROTAS refere-se à definição de terminais do aplicativo (URIs) e como eles respondem às solicitações do cliente.
- Por exemplo:

http://localhost:3000/produtos

http://localhost:3000/produtos/999

Módulo **File System** (FS)

- Com o módulo file system é possível transformar o Node em um servidor de arquivos.
- Para isso é necessário adicionar o módulo FS:

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

Módulo File System (FS)

Com o módulo FS é possível realizar as seguintes operações com arquivos:

- Ler
- Criar
- Atualizar
- Excluir
- Renomear

Lendo arquivos com o Node.js

- O método **fs.readFile ()** é usado para ler arquivos no seu computador.
- Suponha que tenhamos o arquivo HTML exemplo.html (localizado na mesma pasta que o Node.js):

```
<html>
<head>
  <meta charset="utf-8">
  <title>Exemplo</title>
</head>
<body>
  <h1> Este é um exemplo! </h1>
</body>
</html>
```

Lendo arquivos com o Node.js

```
var http = require('http');
var fs   = require('fs');
http.createServer(function (req, res) {
  fs.readFile(‘exemplo4.html', function(err, data) {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write(data);
    res.end();
  });
}).listen(3000);
```

Criando arquivos com o Node.js

- Temos três métodos para criar aquivos:
  - fs.appendFile()
  - fs.open()
  - fs.writeFile()
- O método fs.appendFile () acrescenta conteúdo especificado a um arquivo.
- Se o arquivo não existir, o arquivo será criado:

Criando arquivos com o Node.js

```
var fs = require('fs');
fs.appendFile('exemplo5.txt', ‘Campinas, 26 de março de 2019!', function (err) {
  if (err) throw err;
  console.log('Feito!');
});
```

Criando arquivos com o Node.js

- O método **fs.open ()** recebe uma "flag" como segundo argumento.
- Se a flag for "**w**" para "**writing**", o arquivo especificado é aberto para gravação.
- Se o arquivo não existir, um arquivo vazio será criado:

```
var fs = require('fs');
fs.open(‘exemplo.txt', 'w', function (err, file) {
  if (err) throw err;
  console.log(‘Feito!');
});
```

Criando arquivos com o Node.js

- O método **fs.writeFile ()** substitui o arquivo e o conteúdo especificados, se existirem.
- Se o arquivo não existir, um novo arquivo, contendo o conteúdo especificado, será criado:

```
var fs = require('fs');
fs.writeFile('exemplo6.txt',  'CURSO
JAVASCRIPT',  function (err) {
  if (err) throw err;
  console.log(‘Feito!');
});
```

Atualizando Arquivos com Node.js

- O módulo do FS tem métodos para atualizar arquivos:
  - fs.appendFile()
  - fs.writeFile()
- O método **fs.appendFile ()** acrescenta o conteúdo especificado no arquivo especificado:

Atualizando Arquivos com Node.js

```
var fs = require('fs');
fs.appendFile(‘exemplo7.txt', ' NODE E
ANGULAR', function (err) {
  if (err) throw err;
  console.log('Atualizado!');
});
```

Atualizando Arquivos com Node.js

- O método **fs.writeFile ()** substitui o arquivo e o conteúdo especificados:

```
var fs = require('fs');
fs.writeFile(‘exemplo5.txt', ‘Exemplo
5', function (err) {
  if (err) throw err;
  console.log(‘Atualizado!');
});
```

Excluindo Arquivos com Node.js

- O módulo FS possui apenas um método.
- O método **fs.unlink ()** exclui o arquivo especificado:

```
var fs = require('fs');
fs.unlink(‘exemplo.txt', function (err) {
  if (err) throw err;
  console.log(‘Arquivo excluido!');
});
```

Renomear Arquivos com Node.js

- Para renomear um arquivo com o módulo FS, usamos o método **fs.rename ():**

```
var fs = require('fs');
fs.rename('exemploA.txt', 'exemploB.txt', functi
on (err) {
  if (err) throw err;
  console.log('Arquivo renomeado!');
});
```

Módulo **URL**

- O módulo de URL divide um endereço da web em partes legíveis.
- Para incluir o módulo de URL, use o método require ():

```
var url = require('url');
```

Módulo URL

Analise um endereço com o método **url.parse ()** e ele retornará um objeto URL com cada parte do endereço como propriedades:

```
var url = require('url');
var adr = 'http://localhost:3000/default.htm?id=5&curso=59';
var q = url.parse(adr, true);

console.log(q.host);
console.log(q.pathname);
console.log(q.search);

var qdados = q.query;
console.log(qdados.curso);
```

Criando seus próprios módulos

- Podemos criar seus próprios módulos e importá-los facilmente em seus aplicativos.
- Os módulos permitem incluir bibliotecas externas, como bibliotecas de acesso ao banco de dados e ajudam a organizar seu código em partes separadas com responsabilidades limitadas.
- Você deve tentar identificar partes reusáveis do seu código e transformá-las em módulos separados para reduzir a quantidade de código por arquivo e para ficar mais fácil de ler e manter seu código.

- O exemplo a seguir cria um módulo que retorna um objeto de data e hora:

```
exports.myDateTime = function () {
    return Date();
};
```

- O comando exports faz com que propriedades e métodos fiquem acessíveis para fora do arquivo
- Podemos salvar o módulo em um arquivo com o nome **incluiModulo.js**

Incluindo o módulo criado

- Considere o exemplo abaixo (exemplo2.js):

```
var http = require('http');
var dt = require('./incluiModulo');

http.createServer(function (req, res) {
    res.writeHead(200, {'Content-
Type': 'text/html'});
    res.write("The date and time are currently:
" + dt.myDateTime());
    res.end();
}).listen(3000);
```