



Backend com Node.js

O que é Node.js?

Node.js é uma plataforma construída sobre o motor JavaScript do Google Chrome para facilmente construir aplicações de rede rápidas e escaláveis, utilizando JavaScript. Node.js usa um modelo de I/O direcionado a eventos, não bloqueante, que o torna leve e eficiente. Ele é ideal para aplicações que precisam funcionar em tempo real com troca intensa de dados através de dispositivos distribuídos.

Breve História

Na JSConf 2009 Europeia, um programador jovem chamado Ryan Dahl, apresentou um projeto em que estava trabalhando. Este projeto era uma plataforma que combinava a máquina virtual JavaScript V8 da Google e um laço de eventos. O projeto apontava para uma direção diferente das outras plataformas em JavaScript que rodam no servidor: todos I/O primitivos são orientados a eventos. Aproveitando o poder e a simplicidade do JavaScript, isso tornou tarefas difíceis de escrever aplicações assíncronas em tarefas fáceis. Desde quando foi aplaudido de pé no final do seu discurso, o projeto de Dahl tem recebido uma popularidade e uma aprovação sem precedentes.

Em janeiro de 2010, o NPM foi adicionado ao projeto, sendo um poderoso gerenciador de pacotes que possibilitou o compartilhamento e publicação de códigos e bibliotecas escritos utilizando Node.js. O NPM também simplificou o processo de instalação, atualização e desinstalação de módulos, aumentando ainda mais a popularidade do Node.js.

A Microsoft auxiliou na popularização do projeto, lançando em 2011 a versão Windows do Node.

Atualmente, o Node.js é mantido pela Node.js Foundation – uma organização independente e sem fins lucrativos, que tem como objetivo a manutenção da tecnologia, tendo apoio da comunidade de desenvolvedores.

Como o Node funciona

O Node roda em uma JavaScript V8 VM. Mas espere, JavaScript no servidor? Isso, você leu certo. JavaScript no lado do servidor pode ser um conceito novo para todos que trabalharam exclusivamente com o JavaScript no lado do cliente, mas a ideia em si não é tão absurda – porque não usar a mesma linguagem de programação no cliente que você usa no servidor?

O que é V8? O motor JavaScript V8 é o motor que a Google usa com seu navegador Chrome. Poucas pessoas pensam sobre o que realmente acontece com o JavaScript no lado do cliente. Bem, a engine JavaScript realmente interpreta o código e o executa. Com o V8, a Google criou um ultra-rápido interpretador escrito em C++, com um outro aspecto único: você pode baixar a engine e incorporá-la em qualquer aplicação desejada. Isso não está restrito em rodar em um navegador. O Node atualmente usa o motor JavaScript V8 escrito pela Google e propõe que seja usado no servidor.

Que problema o Node pode resolver?

Node estabeleceu o objetivo número um que é “fornecer uma maneira fácil para construir programas de rede escaláveis”. Qual é o problema com os programas servidores atuais? Vamos fazer os cálculos. Em linguagens como Java™ e PHP, cada conexão cria uma nova thread que potencialmente tem anexado 2 MB de memória com ela. Em um sistema que tenha 8 GB de RAM, isso põe o número máximo teórico de conexões concorrentes a cerca de 4.000 usuários. E quando o número de usuários aumenta, se você quer que sua aplicação web suporte mais usuários, você tem que adicionar mais e mais servidores. Somado a estes custos também podem haver possíveis problemas técnicos: um usuário pode usar diferentes servidores para cada requisição, então cada recurso compartilhado deve ser compartilhado para todos os servidores. Por todas estas razões, o gargalo em toda a arquitetura de aplicações web (incluindo velocidade de tráfego, velocidade do processador e velocidade da memória) é o número de conexões concorrentes que o servidor pode manipular.

Node resolve esta questão trocando a maneira como a conexão é tratada no servidor. Ao invés de criar uma nova thread do sistema a cada conexão (e alocar a memória anexa a ela), cada conexão dispara um evento executado dentro da engine de processos do Node. Node afirma que nunca vai dar deadlock, já que não há bloqueios permitidos, e ele não bloqueia diretamente para chamadas de I/O. Node também alega que um servidor rodando ele pode suportar dezenas de milhares de conexões simultâneas.

Instalando o Node.js

Instalação no Windows

Para realizar a instalação no Windows, você deve entrar na página oficial do Node.js (<https://nodejs.org>) e realizar o download do instalador. Recomendamos fortemente que seja utilizada a versão “Recommended For Most Users”, pois é a versão mais estável da runtime.

Para concluir a instalação, siga os passos do instalador. Recomendamos que não seja alterada a pasta padrão de instalação do Node.js.

Após efetuar a instalação, você pode verificar se está tudo correto utilizando o comando:

```
node -v
```

Instalação no Linux

Em distribuições do Ubuntu, a instalação pode ser feita através do seguinte comando:

```
sudo apt-get install -y nodejs
```

Após efetuar a instalação, você pode verificar se está tudo correto utilizando o comando:

```
node -v
```

Hello Word com Node.js

Crie uma pasta para seu projeto, em um lugar de sua escolha, e dentro dela crie um arquivo chamado `hello.js`, e dentro dele digite o seguinte código:

```
console.log('Olá Mundo!');
```

Para executar o programa volte ao terminal, acesse a pasta do seu projeto:

```
cd pasta/projeto
```

e execute o seguinte comando:

```
$ node hello.js
```

```
Olá Mundo!
```

Através do Node, será possível executar qualquer arquivo JavaScript (extensão `js`), utilizando o comando “node”.

Porém, o que acontece se alterarmos nosso arquivo `hello.js` incluindo uma chamada ao método “`alert`”, no lugar do “`console.log`”?

```
alert("Olá Mundo!");
```

Ao executarmos novamente o arquivo `hello.js`, veremos que um erro é apresentado no terminal:

```
(function (exports, require, module, __filename, __dirname) { alert("Olá Mundo
Node!");
```

^

```
ReferenceError: alert is not defined
    at Object.<anonymous> (C:\Users\edson.alves\Google
Drive\Residência\NodeJS\hello-word\hello.js:1:63)
    at Module._compile (internal/modules/cjs/loader.js:688:30)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:699:10)
    at Module.load (internal/modules/cjs/loader.js:598:32)
    at tryModuleLoad (internal/modules/cjs/loader.js:537:12)
    at Function.Module._load (internal/modules/cjs/loader.js:529:3)
    at Function.Module.runMain (internal/modules/cjs/loader.js:741:12)
    at startup (internal/bootstrap/node.js:285:19)
    at bootstrapNodeJSCore (internal/bootstrap/node.js:739:3)
```

Isto ocorre porque, apesar da função `alert` ser extremamente comum no mundo do JavaScript web, ela só é definida no navegador que está no lado cliente. Logo, quando tentamos acessá-la no lado servidor, encontramos este erro que nos indica que a função não está definida.

Utilizando o NPM

Com a popularização do Node.js, diversos módulos, frameworks e bibliotecas surgiram para o Node, facilitando muito o fluxo de desenvolvimento. Para utilizar estes módulos, precisamos utilizar uma ferramenta para realizar o gerenciamento dos pacotes. Esta ferramenta é o NPM (Node Package Manager).

Como o NPM funciona

O NPM é o Gerenciador de Pacotes do Node, que vem junto com ele e que é muito útil no desenvolvimento com o Node. Ele funciona baseado em dois ofícios:

- Ele é um repositório amplamente usado para a publicação de projetos Node.js de código aberto (open-source). Isso significa que ele é uma plataforma online onde qualquer pessoa pode publicar e compartilhar ferramentas escritas em JavaScript.
- O NPM é uma ferramenta de linha de comando que ajuda a interagir com plataformas online, como navegadores e servidores. Essa utilidade auxilia na instalação e desinstalação de pacotes, gerenciamento de versões e gerenciamento de dependências necessárias para executar um projeto.

Começar um Projeto Com NPM

Se você já tiver o Node e o NPM e deseja começar a construir seu projeto, execute o comando `NPM init`. Com isso, você dará procedimento à inicialização do seu projeto.

Por exemplo, vamos criar um diretório chamado `test-npm` e entrar nele através do comando `"cd test-npm"`. Agora, vamos executar nosso primeiro comando NPM:

```
$ npm init
```

Ao executar o comando `npm init`, algumas informações serão requeridas a respeito do seu projeto. Isto ocorre pois o comando funciona como uma ferramenta para criar o arquivo `package.json`, que é responsável por conter metadados específicos relacionados ao seu projeto. Os metadados mostram alguns aspectos do projeto, por exemplo:

- O nome do projeto;
- A versão inicial;
- A descrição;
- O ponto de entrada;
- Os comandos de teste;
- O repositório git;

- As palavras-chave;
- A licença;
- As dependências;
- As dependências do desenvolvedor (devDependencies).

Os metadados ajudam a identificar o projeto e agem como uma base para que os usuários obtenham as informações sobre ele. Veja um exemplo de arquivo package.json.

```
{
  "name": "test-npm",
  "version": "1.0.0",
  "description": "Um teste do npm",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

Como Instalar Módulos npm e Começar Projetos?

Um pacote em node.js contém todos os arquivos que você precisa para um módulo. Os módulos são bibliotecas JavaScript que você pode incluir no seu projeto.

Instalar módulos é uma das coisas mais básicas que você deve aprender a fazer quando começar a usar o gerenciador de pacotes do Node. Segue um comando para instalar um módulo no diretório atual:

```
$ npm install <module>
```

Ou então:

```
$ npm i <module>
```

Por exemplo, se você quer instalar o módulo Express – um conhecido framework web Node.js – você pode executar o seguinte comando:

```
$ npm i express
```

O comando acima irá instalar o módulo express em /node_modules, no diretório atual. Sempre que você instalar um módulo do npm, ele será instalado na pasta node_modules.

Após executar o comando, você também observará que o express foi adicionado ao arquivo package.json.

```
{
  "name": "test-npm",
  "version": "1.0.0",
  "description": "Um teste do npm",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.17.1"
  }
}
```

Isto significa que o `express` agora é uma dependência do nosso projeto. Caso futuramente um outro desenvolvedor precise trabalhar em nosso projeto, ele não precisará que disponibilizemos o diretório `node_modules`. O desenvolvedor, tendo os arquivos fontes do projeto e o arquivo `package.json`, só precisará entrar no diretório e executar o comando:

```
$ npm install
```

Este comando lerá o arquivo `package.json`, e fará download de todas as dependências necessárias para execução do projeto, criando um diretório `node_modules` no diretório do projeto.

O que é uma API?

A sigla API corresponde às palavras em inglês “Application Programming Interface”. No português “Interface de Programação de Aplicações”. Elas são uma forma de integrar sistemas, possibilitando benefícios como a segurança dos dados, facilidade no intercâmbio entre informações com diferentes linguagens de programação e a monetização de acessos.

As APIs são um tipo de “ponte” que conectam aplicações, podendo ser utilizadas para os mais variados tipos de negócio, por empresas de diversos nichos de mercado ou tamanho, conforme veremos melhor ao longo de nosso post.

A maioria das pessoas não sabe o que é uma API. Isso porque elas são invisíveis ao usuário comum, que enxerga apenas a interface dos softwares e aplicativos. No entanto, os profissionais de programação conhecem por dentro essa tecnologia que é resultado da evolução de diversos sistemas e ferramentas. Aplicativos e softwares de diversos tipos são apenas passíveis de construção por meio dos padrões e especificações disponibilizados pelas APIs.

As APIs proporcionam a integração entre sistemas que possuem linguagem totalmente distintas de maneira ágil e segura. Em outras formas de integração de sistemas, o profissional que realiza o trabalho precisa, muitas vezes, instalar recursos compatíveis com o sistema no qual

se busca efetuar a integração, gerando um grande trabalho e, conseqüentemente, atraso na geração de negócios e processos produtivos de uma companhia.

As possibilidades disponibilizadas pelo uso das APIs proporcionam para os desenvolvedores de softwares e aplicativos a possibilidade de conectar tecnologias heterogêneas, como diferentes bancos de dados, por exemplo. Além disso, é possível fazer com que funcionalidades e ferramentas específicas de determinados aplicativos sejam utilizadas em outros, sem que isso cause qualquer dificuldade, conforme veremos no tópico a seguir.

Utilizando o Express

Para que serve o Express?

Nos capítulos anteriores, utilizamos o express como exemplo para instalação de um módulo. O Express é o framework web mais popular, e a biblioteca subjacente para uma série de outros frameworks populares de Node.js. Fornece mecanismos para:

- Gerenciar as requisições de diferentes requisições e rotas e URLs.
- Combinar mecanismos de renderização de "view" para gerar respostas inserindo dados em modelos.
- 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.
- Adicionar em qualquer ponto da requisição um "middleware" para interceptar, processar ou realizar tratamentos em relação à mesma.

Enquanto o Express é bastante minimalista, os desenvolvedores criam pacotes de middleware para resolver quase todos os problemas no desenvolvimento web. Há bibliotecas para trabalhar com cookies, sessões, login de usuários, parâmetros de URL, dados em requisições POST, cabeçalho de segurança e entre tantos outros.

Você pode achar uma lista de pacotes de middleware mantidos pela equipe Express em <http://expressjs.com/en/resources/middleware.html> (juntamente com uma lista de alguns pacotes populares de terceiros).

Utilizaremos o Express neste material para criar uma API, utilizando Node.js!

Express é opinativo?

Os frameworks web costumam se referir a si mesmas como "opinativas" ou "não opinativas".

Os frameworks opinativos são aqueles com opiniões sobre o "caminho certo" para lidar com qualquer tarefa específica. Muitas vezes, eles apoiam o desenvolvimento rápido em um domínio particular (resolvendo problemas de um tipo específico) porque a maneira correta de fazer qualquer coisa geralmente é bem compreendida e bem documentada. No entanto, eles podem ser menos flexíveis na resolução de problemas fora de seu domínio principal e tendem a oferecer menos opções para quais componentes e abordagens eles podem usar.

Frameworks não opinativos, ao contrário, têm muito menos restrições sobre a melhor maneira de utilizar componentes juntos para atingir um objetivo, ou mesmo quais componentes devem ser usados. Eles tornam mais fácil para os desenvolvedores usar as ferramentas mais adequadas para completar uma tarefa específica, embora você precise encontrar esses componentes por si próprio.

Express é não opinativo. Você pode inserir qualquer middleware que você goste no manuseio das solicitações em quase qualquer ordem que desejar. Pode estruturar o aplicativo em um arquivo ou em vários, usar qualquer estrutura de pastas dentro do diretório principal. Às vezes você sente que você tem diversas opções.

Hello World com Express

Vamos evoluir nosso Hello World, dos capítulos passados, para agora utilizar o Express, retornando uma resposta ao navegador. Para isso, vamos criar um novo diretório chamado teste-express, e iniciar um projeto Node a partir do comando “npm init”. Também precisaremos instalar o express em nosso projeto, utilizando o comando:

```
$ npm i express
```

Dentro do diretório do projeto, vamos criar um arquivo hello.js, onde vamos escrever o código necessário.

O primeiro passo é importar a dependência do express, que está instalada em nosso projeto. Para isso vamos utilizar a função require, disponibilizada pelo Node.js para importar módulos:

```
var express = require("express");
```

Em nossa variável express, temos nada mais do que uma função capaz de criar nosso app express. Vamos utilizá-la, incluindo o código abaixo:

```
var express = require("express");  
var app = express();
```

Agora, precisamos cadastrar a rota que o usuário acessará no navegador, para receber nosso “Olá Mundo!”. O código abaixo (as três linhas que começam com app.get) mostra uma definição de rota. O método app.get() especifica uma função de retorno de chamada que será invocada sempre que exista uma solicitação HTTP GET com um caminho ('/') relativo à raiz do site. A função de retorno de chamada requer uma solicitação e um objeto de resposta como argumentos, e simplesmente chama send() na resposta para retornar a string "Olá Mundo!".

```
var express = require("express");
var app = express();

app.get("/", function(req, res) {
  res.send("Olá Mundo!");
});
```

O bloco final será responsável por iniciar o servidor na porta '3000' e imprimir um comentário de log no console.

```
var express = require("express");
var app = express();

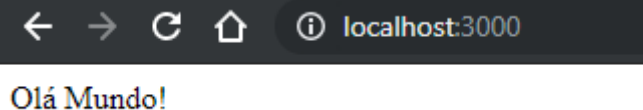
app.get("/", function(req, res) {
  res.send("Olá Mundo!");
});

app.listen(3000, function() {
  console.log("App de Exemplo escutando na porta 3000!");
});
```

Para executar o projeto, dentro do diretório do mesmo, execute o comando:

```
$ node hello.js
```

Com o servidor em execução, você pode acessar o localhost:3000 em seu navegador para ver o exemplo de resposta retornado.



← → ↻ 🏠 ⓘ localhost:3000

Olá Mundo!

Bibliografia

- <http://nodebr.com/o-que-e-node-js/>
- <http://clubedosgeeks.com.br/programacao/node-js/node-js-iniciando-criando-sua-primeira-aplicacao>
- <https://www.sitepoint.com/beginners-guide-node-package-manager/>
- <https://www.hostinger.com.br/tutoriais/o-que-e-npm>
- https://developer.mozilla.org/pt-BR/docs/Learn/Server-side/Express_Nodejs/Introdu%C3%A7%C3%A3o
- <https://vertigo.com.br/o-que-e-api-entenda-de-uma-maneira-simples/>