Introdução ao Node.js

Conhecimentos	
Data da aula	@19 de agosto de 2024
Tipo	Atividade em Classe · Aula Expositiva

Objetivos de Aprendizagem

1. Funcionamento do node.js
2. Instalação do ambiente
3. Criando e executando projetos node

Funcionamento do node.js

A história do node.js

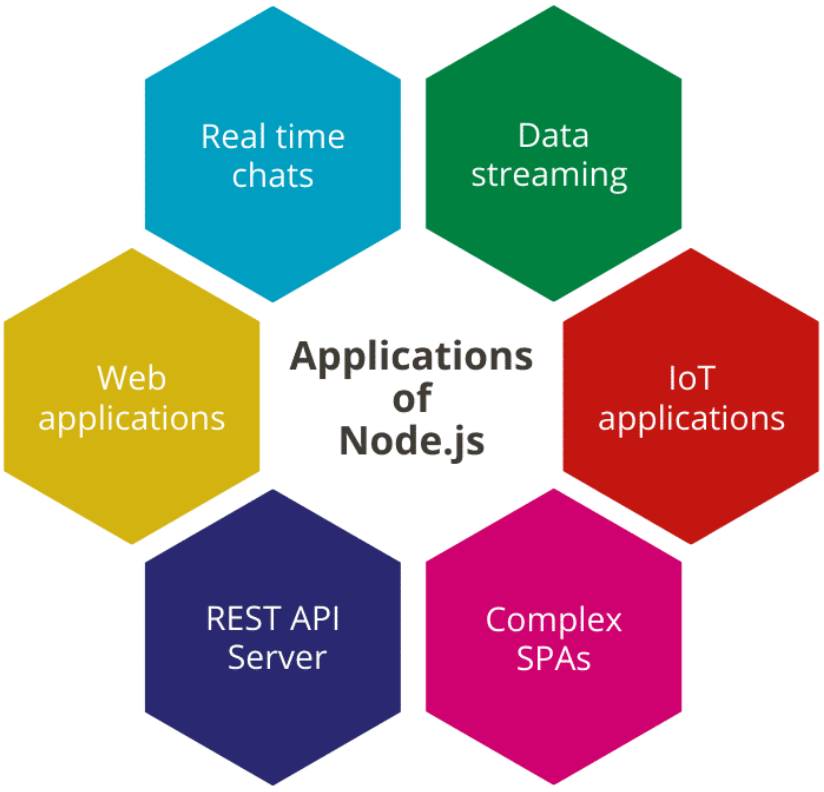
- **2009**: Node.js foi criado por Ryan Dahl e lançado como um ambiente de execução de JavaScript do lado do servidor.
- **2010**: A versão 0.1.14 do node.js foi lançada, marcando o início do desenvolvimento da comunidade e o crescente interesse na plataforma.

- **2011**: O node.js foi adotado por grandes empresas como Microsoft, Yahoo! e LinkedIn, aumentando ainda mais a sua popularidade.
- **2012**: A formação da Fundação node.js como uma organização sem fins lucrativos para gerenciar e evoluir o projeto. Mais tarde conhecida como OpenJS Foundation.
- **2013**: Lançamento da versão 0.10 do node.js, trazendo melhorias de desempenho e estabilidade.
- **2015**: A versão 0.12 introduziu o suporte oficial para o ECMAScript 6 (ES6), trazendo recursos modernos ao JavaScript.
- **2016**: O lançamento da versão 6 do node.js, marcando a transição para um ciclo de lançamento de versões Long Term Support (LTS).
- **2020**: Ryan Dahl cria o Deno, o maior concorrente do node.js, dessa vez fazendo tudo do jeito como ele queria.

O que o node não é

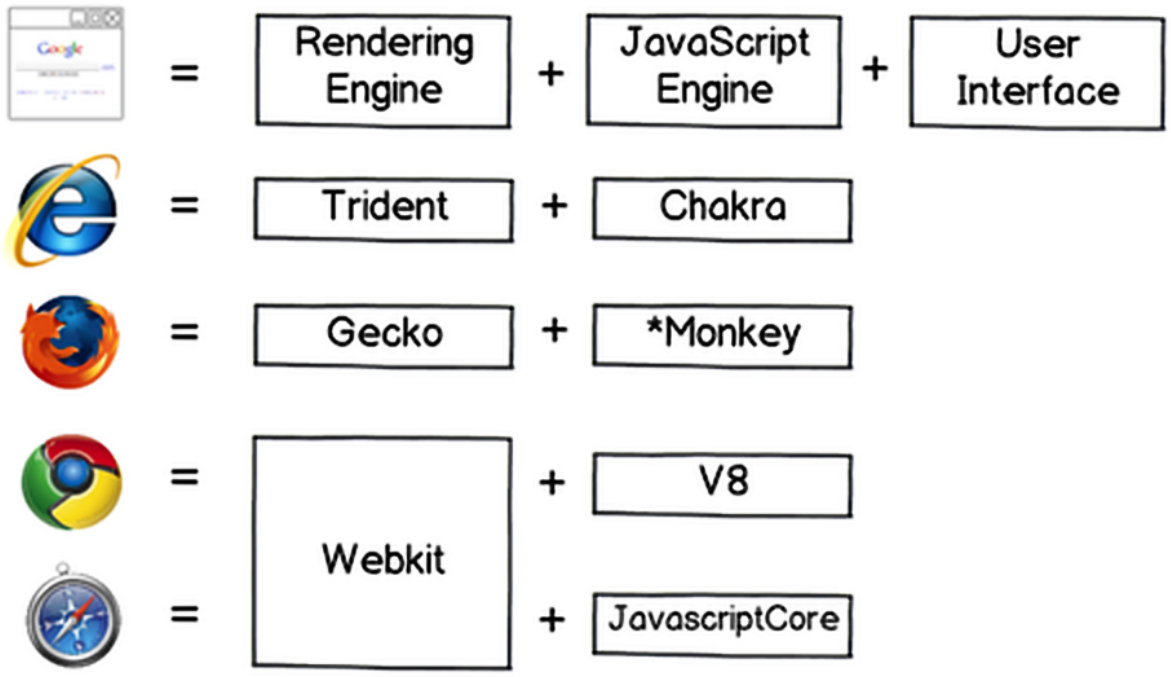
- node.js não é uma linguagem de programação, a linguagem é JavaScript
- Não é um framework, é um *runtime* (ambiente de execução) de JavaScript
- node.js não é limitado, ele faz tudo que outras tecnologias de *backend* fazem

O que node faz



Aplicações com node.js

Como o node.js funciona

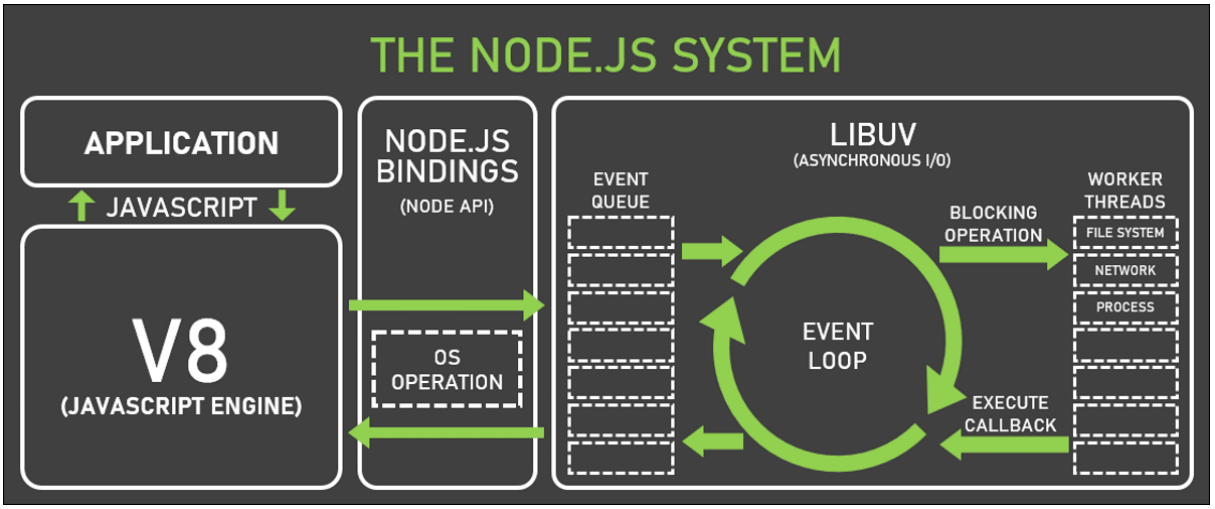


Funcionamento dos navegadores

1. **Aplicação**: O código JavaScript escrito pelo usuário que roda no Node.js.
2. **V8 (Motor JavaScript)**: O motor que executa o código JavaScript. Ele é responsável por converter JavaScript em código de máquina.
3. **Bindings do Node.js (API do Node)**: A interface entre o motor V8 e o sistema operacional subjacente. Ele lida com as operações do sistema operacional.
4. **LIBUV (I/O Assíncrono)**: Uma biblioteca que fornece o loop de eventos e operações de I/O assíncronas.
    - **Loop de Eventos**: O mecanismo central que lida com operações assíncronas. Ele processa tarefas da fila de eventos.
    - **Operação de Bloqueio**: Quando uma tarefa requer I/O, ela é delegada a threads de trabalho.
    - **Threads de Trabalho**: Realizam tarefas relacionadas ao sistema de arquivos, rede ou processos.
    - **Executar Callback**: Uma vez que uma tarefa é concluída pelos threads de trabalho, o loop de eventos executa o callback associado.

O fluxo envolve o código da aplicação acionando eventos, que são tratados pelo loop de eventos. As tarefas são delegadas aos threads de trabalho

conforme necessário e os callbacks são executados após a conclusão, facilitando operações de I/O não bloqueantes.



Funcionamento do node

Instalando o ambiente node.js

Checklist de itens básicos:

- [ ] Editor de código de sua preferência: No curso utilizamos **VSCode**
  - [ ] Instalação: seguir o passo a passo no site https://code.visualstudio.com/
- [ ] Tecnologia responsável principal: **node.js**
  - [ ] Instalação: seguir o passo a passo no site https://nodejs.org/
- [ ] Ferramenta auxiliar: **NPM**

Criando projetos node

Boas práticas ao criar o seu projeto node:

- Criar uma pasta para salvar seu projeto, o seu *workspace*
- O node trabalha com arquivos JavaScript conectados, através do arquivo empacotador, o `package.json`. Para iniciar, utilize o comando `npm init`. Com isso irá vim uma série de perguntas pelo próprio terminal. Responda-as com calma para criar o arquivo.

- Se você quiser criar um projeto com as respostas padrões, pode-se utilizar o `npm init -y`.

Informações sobre o `package.json`

Vamos olhar o seguinte arquivo exemplo:

```
{
  "name": "@projeto-teste",     // apelido do projeto
  "version": "1.0.0.",          // versão do seu projeto
  "description": "descricao",   // breve descrição do projeto
  "main": "teste.js",           // arquivo principal
  "scripts": {
      "test": "echo \"Error: no teste specified\" && exit 1"
        /*
            -- scripts de automação --
            os scripts podem ser chamados por npm run <nome-d
        */

    }
    "keywords": [],              // palavras chaves para orga
    "author": "Thiago Nogueira", // autor do projeto
    "license": "ISC"             // licença do projeto
}
```

Executando projetos node

A pasta *source*

como boa prática de programação em node.js, é interessante organizar os arquivos .js de uma forma que faça sentido. Uma das primeiras categorias é criar uma pasta scr (*source*). Basicamente, **tudo que ficar dentro da pasta src são arquivos JavaScript que você vai codificar dentro do seu projeto**. Dessa forma, **todos os arquivos fora da pasta src são de configuração do projeto**.

Execução simples

Para executar a aplicação basta iniciar o comando (no terminal) `node <caminho-do-arquivo.js>`. Caso o arquivo, por exemplo um arquivo chamado index.js, esteja dentro da pasta src, basta utilizar o comando `node src/index.js`.

O comando `node --watch`

Da forma anterior, sempre que você alterar algo do arquivo, precisará executar novamente o comando node. Em aplicações mais complexas esse processo pode ser bem desgastante. Pensando nisso, desde a versão **18.11.0** o comando node -watch foi introduzido. Dessa forma, uma vez iniciada a instância, qualquer alteração salva feita no arquivo é executado automaticamente. O comando completo no terminal é `node --watch <caminho-do-arquivo.js>`.

Atividades de Aprendizagem

Exercício 1 - Array de Números aleatórios

Crie um projeto em node, da qual crie uma função que gera um array de números aleatórios. A função recebe como argumento: tamanho do array, valor mínimo e valor máximo dos números.

Exercício 2 - Elemento mais frequente

No mesmo workspace, crie uma função que recebe um array como argumento e retorna o elemento que mais aparece no array. Caso não haja, retorne um texto no terminal com o seguinte texto: `“Não há elemento mais frequente”`.

Exercício 3 - Remoção de elementos repetidos

Ainda no mesmo workspace, Crie uma função que recebe um array, remove os elementos que estiverem repetidos e retorna um array sem essas repetições. Caso não haja elementos repetidos, retorne o seguinte texto: `“Não há elementos repetidos”`.

Exercício 4 - Concatenar arrays

Crie outra função que receba dois arrays e retorne um novo array que é a união de todos os elementos dos arrays anteriores.

Exercício 5 - Junção das funções

Crie uma função `main()` que chame todas as outras funções (você pode criar funções aninhadas e chamar a última nessa função também). Ao passar por cada função, precisa imprimir o seguinte trecho:

```
"-----------------------------------"
"Passando pela função do Exercício X"
"Entrada: array [x,x,x]"
"Saída:   array [y,y,y]"
"-----------------------------------"
```