NODE.JS

Webscrapping com Node.js

Escrito por **Luiz Duarte**
em 30/01/2024

Eu crio webcrawlers e mecanismos de busca tem mais de 10 anos na data que escrevo este post, sendo alguns dos empreendimentos digitais mais bem sucedidos da minha carreira como o Busca Acelerada, o Só Famosos, o BuildIn, entre outros, a maioria feitos para terceiros. Já falei de como você pode criar buscadores em outras oportunidades que listo abaixo:

Como criar um mecanismo de busca

Como criar um mecanismo de busca com Node.js e MongoDB

Como criar um mecanismo de busca com PHP e MongoDB

Como criar um mecanismo de busca em ASP.NET Core e MongoDB

No entanto, eu nunca havia escrito antes sobre como criar um webcrawler antes, ou seja, um script que automaticamente percorre páginas web em busca de informações a serem indexadas para seu buscador. Na verdade eu não vou mostrar hoje como criar um webcrawler completo (que é uma tarefa que une muitos conceitos para um post só), mas sim como fazer um algoritmo que coleta informações de páginas HTML e armazena em um banco de dados, uma técnica chamada webscrapping.

Para que você consiga acompanhar este tutorial é importante que já esteja familiarizado com Node.js e MongoDB, que trato bastante aqui no blog em posts como esse e em meu livro sobre o assunto.

Veremos neste post:

1. Utilidade do webscrapping
2. Criando o projeto

3. Programando o webscrapper
4. Trabalhando com os dados

Vamos lá!

#1 – Utilidade do Webscrapping

Basicamente webscrapping consiste em ler o conteúdo HTML de páginas web, extrair as informações que você deseja se baseando em padrões de elementos (tags), armazenar no seu banco de dados e ignorar o resto, avançando para a próxima página.

Webscrapping é útil em diversos cenários e largamente utilizado por grandes empresas de tecnologia e inteligência competitiva.

você pode fazer scrapping de redes sociais para descobrir tópicos que estão na moda, como o Sentimonitor faz

você pode fazer scrapping de endereços de email disponíveis em websites para vender como o Hunter.io faz

você pode fazer scrapping de informações de outros sites para usar no seu, como o Google faz

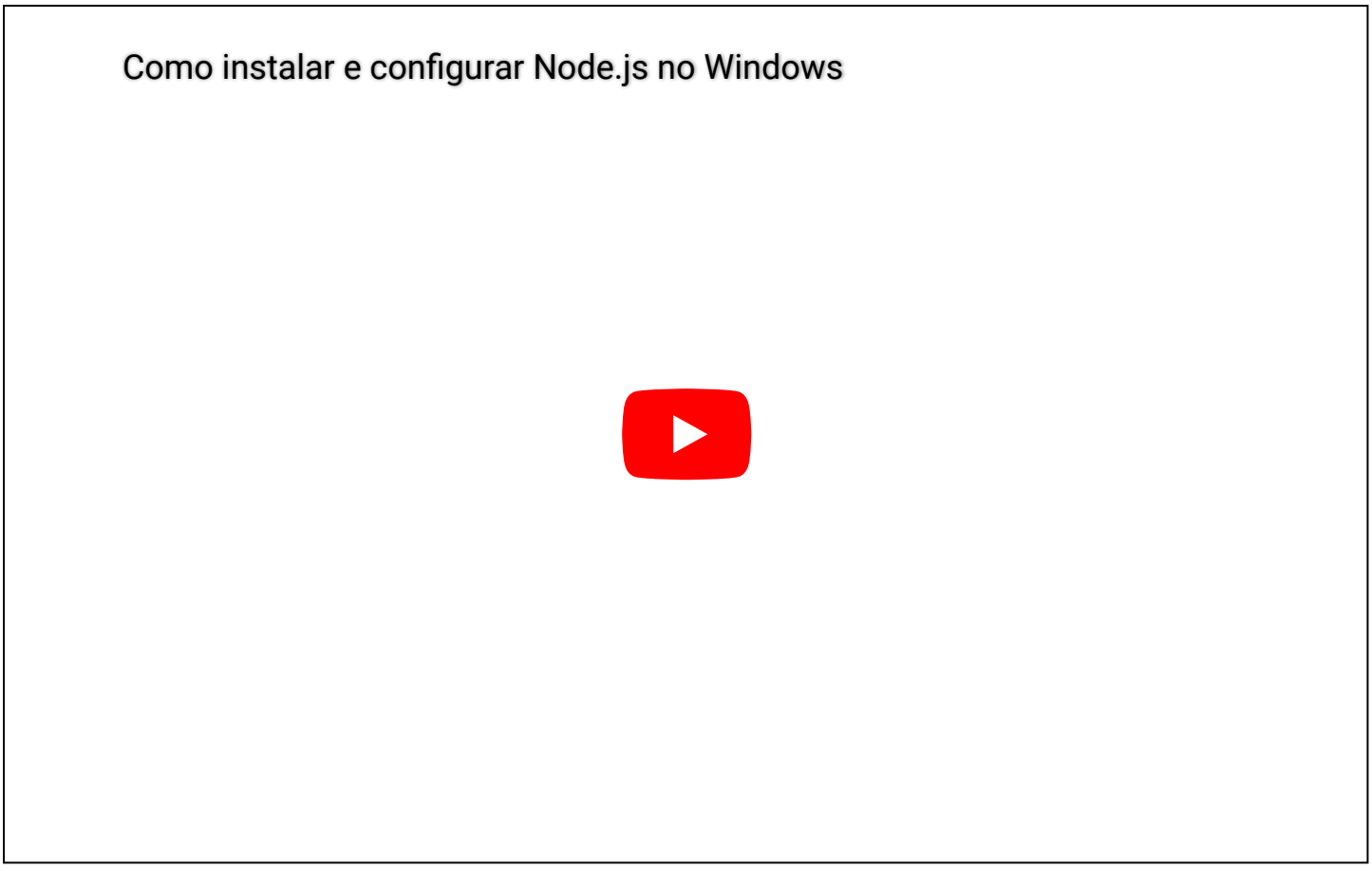
você pode fazer scrapping de preços de produtos em ecommerces para criar comparadores, como o Buscapé faz

Note, no entanto, que muitos websites consideram webscrapping errado, violando seus termos de uso. Sendo assim, utilize as técnicas apresentadas neste artigo com cuidado pois caso o seu webcrawler fique visitando demais o mesmo site é bem possível que seu IP seja bloqueado ou até mesmo que você receba emails com ameaças.

Faça webscrapping por sua própria conta e risco.

#2 – Criando o projeto

Agora que sabe o que é e para que serve webscrapping, vamos criar este projeto. Usaremos Node.js neste tutorial, que você precisa ter na sua máquina antes de continuar. Eu costumo usar também Visual Studio Code e Git, ensino a instalar os três nos vídeo abaixo.



Crie uma pasta para salvar os fontes deste projeto na sua máquina. Eu chamei a minha de webscrapper. Entre na pasta webscrapper via terminal e rode o seguinte comando:

```
1 npm init -y
```

Depois disso, vamos instalar as dependências que precisaremos neste projeto, usando o comando abaixo:

```
1 npm install axios cheerio
```

A saber:

Axios: biblioteca para fazer requisições HTTP

cheerio: oferece as funcionalidades de seletores do JQuery em Node.js

Agora crie um arquivo index.js na raiz do seu projeto e vamos seguir em frente pois o setup está pronto!

#3 – Programando o Webscrapper

Comece o seu index.js adicionando as referências às bibliotecas que instalamos anteriormente:

```
1 const axios = require('axios');
2 const cheerio = require('cheerio');
```

A biblioteca Axios possui uma função get que aceita uma string com a URL da página que queremos "raspar" e retorna uma resposta com os dados da página (HTML cru mesmo). O código abaixo demonstra uma request para uma página com a lista de países existentes no mundo (esse site serve justamente para testes de scrapping, então não se preocupe):

```
async function scrap(){
    const response = await axios.get("https://www.scrapethissite.com/pages/simple/");
    console.log(response.data);
}

scrap();
```

A ideia aqui é pegar a listagem dos nomes dos países existentes no mundo a partir deste site. O que vou fazer com esses dados? Eu não tenho nem ideia, mas achei que seria um bom exemplo...Rode o projeto com o comando abaixo e verá o HTML da página sendo impresso.

```
node index
```

Agora para fazer o carregamento no Cheerio é bem simples, basta usarmos o objeto response como parâmetro da função load dele, como abaixo:

```
async function scrap(){
    const response = await axios.get("https://www.scrapethissite.com/pages/simple/");
    const $ = cheerio.load(response.data);
    console.log($.html());
}
```

Rode novamente o projeto e como resultado, você deve ver no terminal um monte de HTML novamente, mas com a diferença que agora usando aquele objeto $ você pode “navegar” por esse HTML e pegar apenas as informações que lhe interessam.

#4 – Trabalhando com os dados

Agora que você tem um webscrapper simples porém funcional, é hora de trabalhar com aquele objeto gerado no load do Cheerio. Se você nunca usou JQuery antes, o mais importante aqui é bem simples de aprender: seletores. Basicamente todo seletor começa com o cifrão ($) seguido de parênteses e o identificador do seletor que pode ser:

#idDoElemento

.classeDoElemento

tagDoElemento

tag[atributo=valor]

E muito mais, mas esses aí são os principais. Ou seja, se quiser carregar em memória um elemento HTML cujo id seja divCadastro, em JQuery você faria o seguinte:

```
$('#divCadastro')
```

E depois na sequência poderia chamar funções para pegar informações deste elemento, incluindo seu texto, seu HTML, seus atributos, seus nós-filhos, etc.

E cheerio trabalha exatamente desta forma!

Para entender a informação que vamos pegar, é sempre útil ver o código-fonte da página que estamos fazendo scrapping ou melhor ainda: usar o F12 do Google Chrome para inspecionar elementos específicos, como fiz na imagem abaixo para entender a estrutura HTML da listagem de países.

Note que todas as linhas com nomes de países (h3) possuem a classe 'country-name', sendo assim, fica fácil de fazer um seletor por essa classe e com um laço descobrir todos os países. Dentro de cada h3 obtido, pegamos o texto interno do elemento e removemos os espaços desnecessários:

```
async function scrap(){
    const response = await axios.get("https://www.scrapethissite.com/pages/simple/");
    const $ = cheerio.load(response.data);
    $('.country-name').each((i, item) => console.log($(item).text().trim()));
}
```

Se você executar agora, verá no console o nome de todos os países.

Para encerrar, vou modificar o código uma última vez, para pegar além do nome do país, a sua capital, guardando estas informações em um array JSON que facilmente você poderia depois salvar em um banco de dados como o MongoDB, como já mostrei em outros posts.

```
const countries = [];

async function scrap() {
    const response = await axios.get("https://www.scrapethissite.com/pages/simple/");
    const $ = cheerio.load(response.data);
    $('.country-name').each((i, item) => countries.push({ name: $(item).text().trim() }));
    $('.country-capital').each((i, item) => countries[i].capital = $(item).text().trim());

    console.log(countries);
}
```

O resultado esperado deste simples webscrapper, na data que escrevo este post é o array JSON abaixo impresso no console:

```
[
  { name: 'Andorra', capital: 'Andorra la Vella' },
  { name: 'United Arab Emirates', capital: 'Abu Dhabi' },
 //...
]
```

Atente ao fato de que muitos sites alteram o DOM da página usando JavaScript e que algumas vezes você não conseguirá ter acesso aos elementos HTML via cheerio. Nestes casos a técnica correta envolve usar headless browsers como o Phantom. Mas essa é uma técnica muito mais elaborada que explico neste outro post.

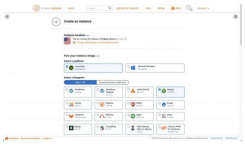
Também aproveito para te convidar a acessar o vídeo abaixo, também sobre bots:

Os 3 Bots para Devs ganharem dinheiro

Espero que tenha gostado!

TAGS: nodejs

Artigos Relacionados



NODE.JS

Como criar mocks de BD em NestJS com Jest e Prisma

NODE.JS

Deploy de aplicação Node.js com WebSocket na AWS (LightSail)

CRIPTO

Como criar bot trader para PancakeSwap (V2) em Node.js

CRIPTO

Como criar bot trader para PancakeSwap (V2) em Node.js - Parte 2

Ver mais

Olá, tudo bem?

O que você achou deste conteúdo? Conte nos comentários.

Escreva seu comentário...

Seu nome? Seu email? Postar

Br Sempre Br , 03/10/2017 às 07:06

Procurei por isso desde maio deste ano. E finalmente consegui fazer o que eu queria.
Você salvou a minha vida! Muito obrigado

Responder

Luiz Fernando Jr , 03/10/2017 às 21:57

Fico feliz de ter salvado sua vida, hehehe

José Maria Reganhan , 24/11/2022 às 12:58

Prezado Luiz:
Bom seu artigo, parabéns! Luiz eu preciso saber se você pode me prestar um serviço à distância (um curso de webscraping em R ou em python para dados estatísticos de bancos de dados públicos). E qual será o preço? Caso possa fazer isto, poderá me enviar uma mensagem privada no messenger, no meu perfil do Facebook (https://www.facebook.com/josemaria.reganhan). Grato.

Responder

Luiz Duarte , 09/12/2022 às 17:58

Infelizmente não possuo disponibilidade para consultoria ou aulas particulares. Deixo aqui seu contato caso alguém leia no futuro e possa lhe ajudar.

Categorias

.NET (20)

Agile (106)

Android SDK (45)

Carreira (74)

Corona SDK (2)

Cripto (103)

Empreendedorismo (67)

Livros (52)

Mobile (71)

Node.js (217)

React Native (14)

Web (180)

Últimas Novidades

Como atualizar Smart Contracts em Solidity (Proxy)

Como configurar a MetaMask para desenvolvimento blockchain

Como criar robô/bot trader para PancakeSwap (V3) em Node.js – Parte 2

Deploy de Smart Contract com HardHat Ignition

Como criar robô/bot trader para PancakeSwap (V3) em Node.js

Tags

agile android aws blo business c# camunda carreira corona criptomoedas digital ocean emp firebase heroku ios java jira mobile mongodb mssql mysql nestjs nextjs nodejs oauth performance phonegap postgresql prisma react redis regex resenha seguranca solidity sqlite tdd typescript web web3