

# JAVASCRIPT ASSÍNCRONO (AJAX E JSON)

FAPESC – DESENVOLVEDORES PARA TECNOLOGIA DA INFORMAÇÃO

HERCULANO DE BIASI

[herculano.debiasi@unoesc.edu.br](mailto:herculano.debiasi@unoesc.edu.br)

# TÓPICOS

- Requisições
- Tecnologia AJAX
- Objeto XMLHttpRequest
- Prática
- JSON
- JSON com JavaScript
- BSON



# REQUISIÇÕES

## ■ Requisições síncronas

- Navegador que fez a requisição fica aguardando (bloqueado) até que o servidor envie uma resposta
- Uma requisição precisa ser terminada para que uma nova seja feita

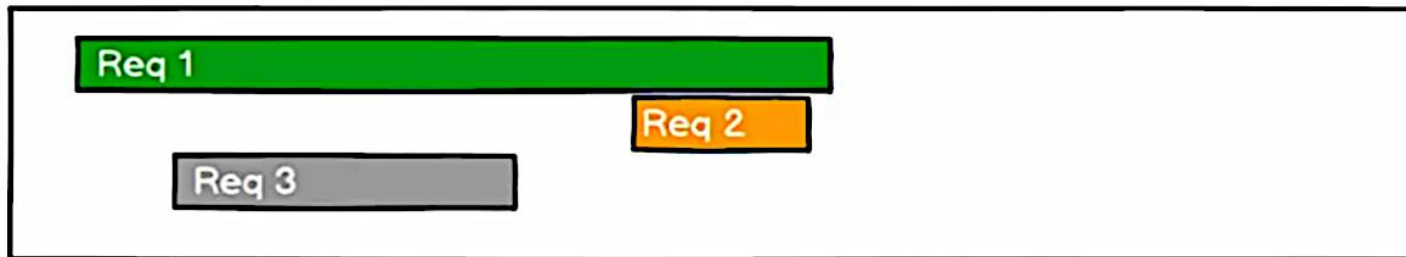
Requisições **síncronas**



## ■ Requisições assíncronas

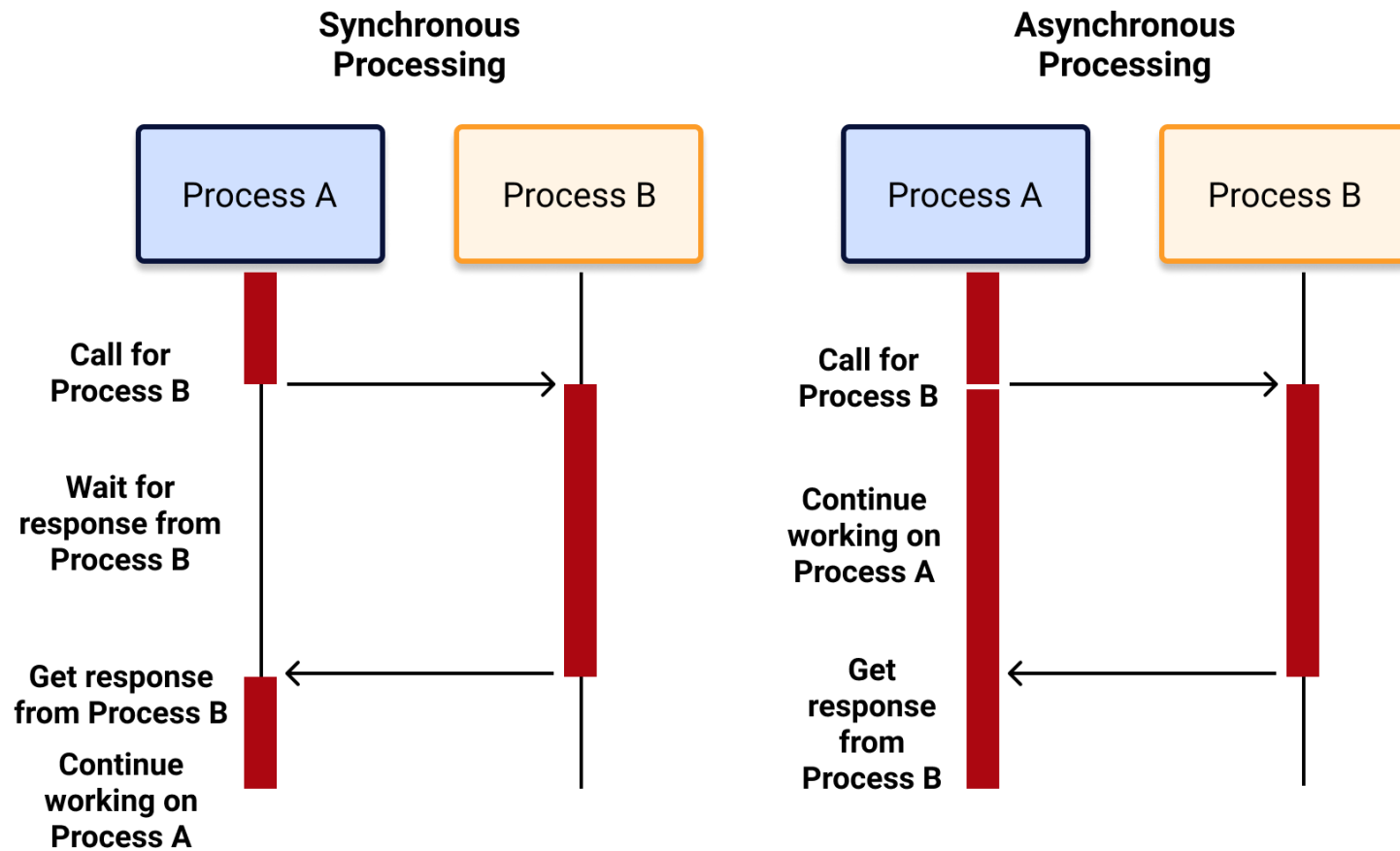
- Várias requisições podem ser disparadas em paralelo ao mesmo tempo
- Sistema baseado em 'eventos'

Requisições **assíncronas**



# REQUISIÇÕES

## ■ Requisições síncronas vs. Requisições assíncronas



# TECNOLOGIA AJAX

- **AJAX** ou Ajax (*Asynchronous JavaScript and XML* – JavaScript Assíncrono e XML) é uma forma de enviar e receber dados do servidor de forma assíncrona (não bloqueia o navegador, ou seja, não o espera para atualizar os dados), sem a necessidade de recarregar a página para que todos os dados sejam atualizados
- Ajax permite tornar páginas web mais interativas com o usuário, utilizando-se de solicitações assíncronas de informações
- Ajax é a chamada de um recurso no servidor a partir de um código JavaScript no navegador web, de forma que o resultado atualize apenas uma parte da página sem precisar fazer uma atualização dela inteira
- Essa chamada é assíncrona, ou seja, o *script* que a chamou continua sua execução sem esperar pela resposta
- Quando o servidor responde, uma função JavaScript especificada trata corretamente os dados retornados, fazendo a atualização de parte da tela apenas
- Em geral, isso significa que páginas web com recursos Ajax permitem maior interatividade, velocidade de processamento e usabilidade

# TECNOLOGIA

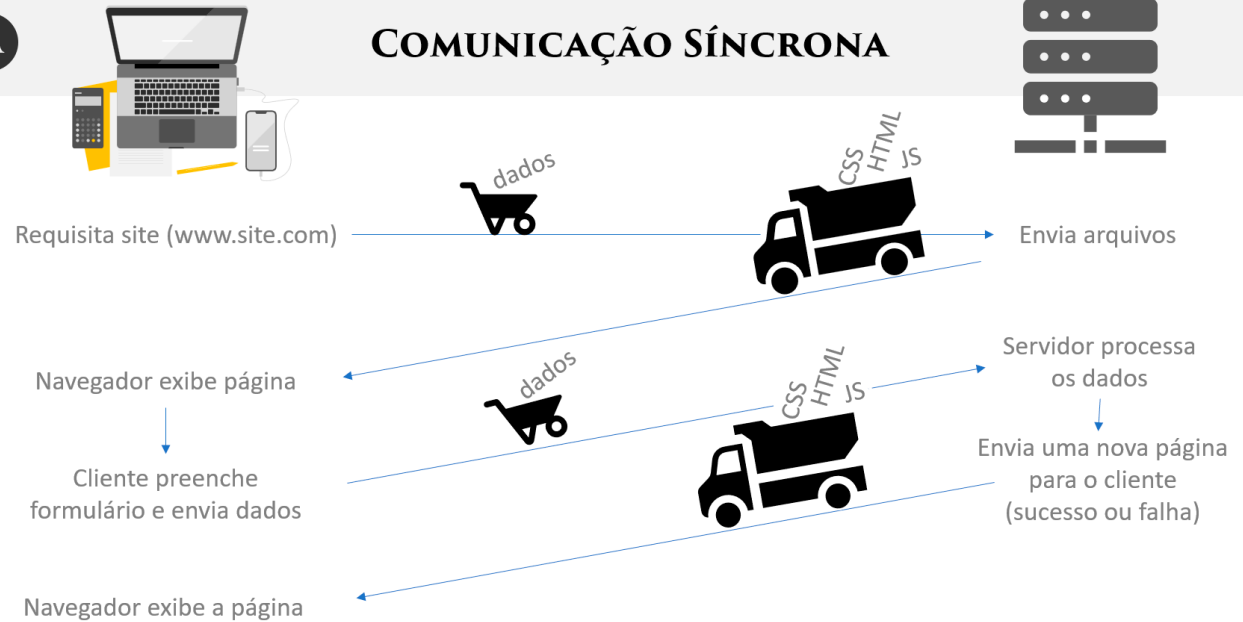
## ■ Comunicação síncrona vs. assíncrona

CLIENTE

SERVIDOR

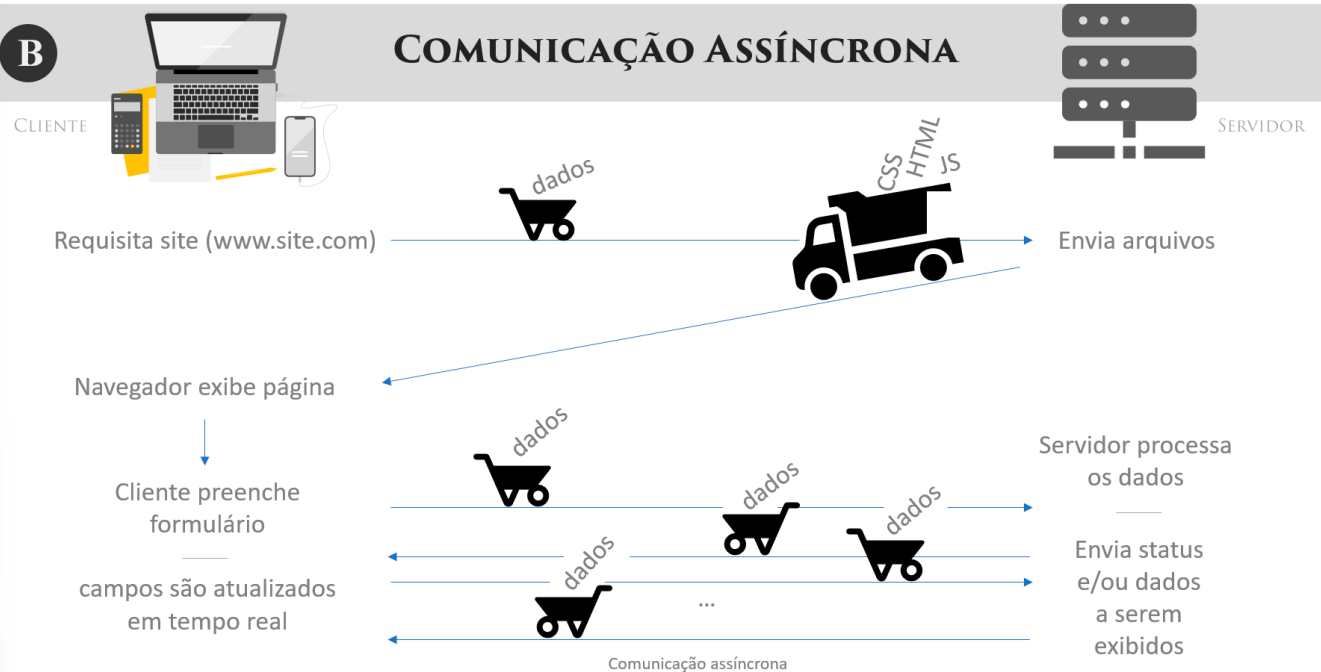
A

### COMUNICAÇÃO SÍNCRONA



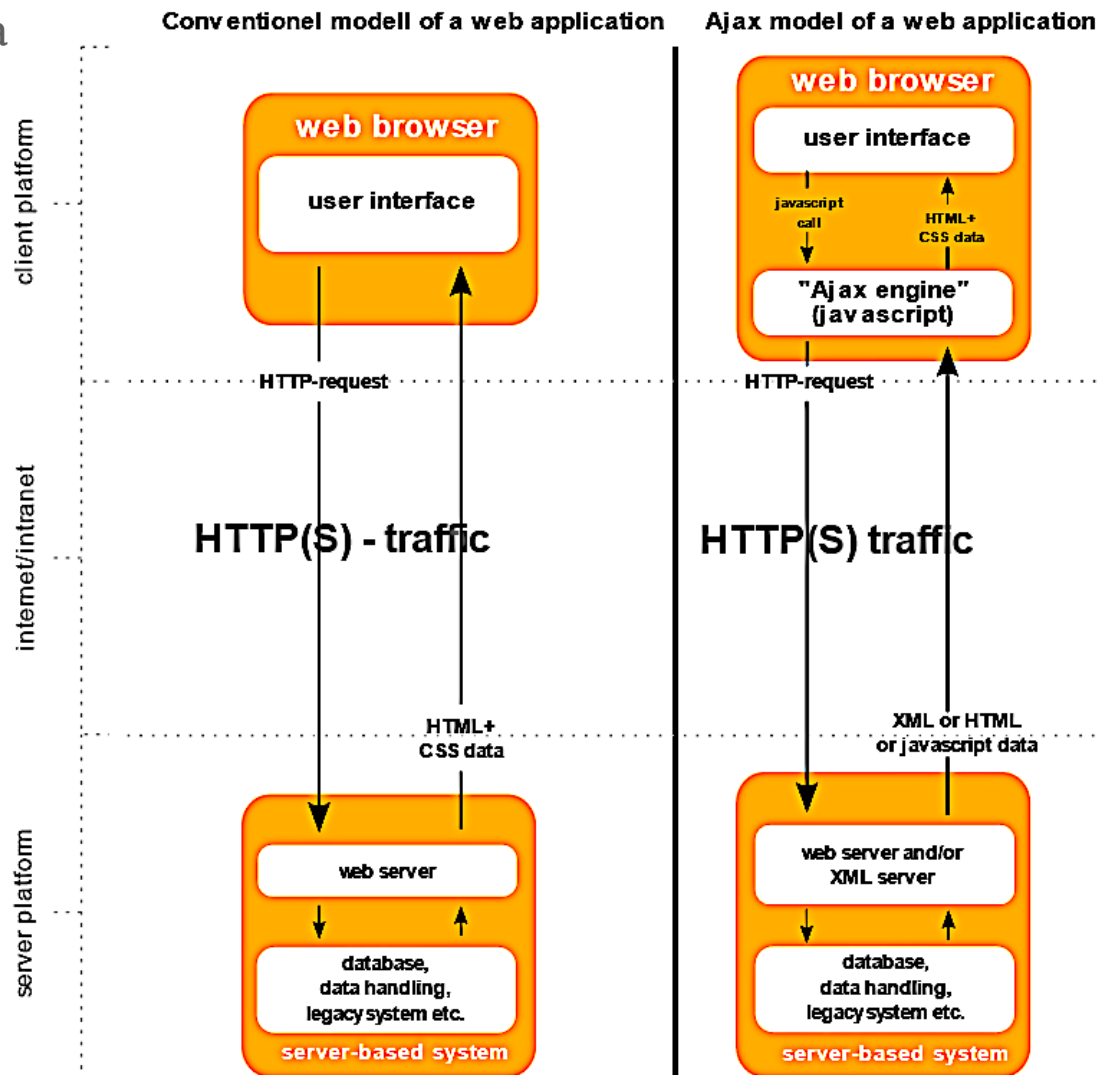
B

### COMUNICAÇÃO ASSÍNCRONA



# TECNOLOGIA AJAX

- Modo convencional de uma aplicação web vs. Aplicação web usando Ajax



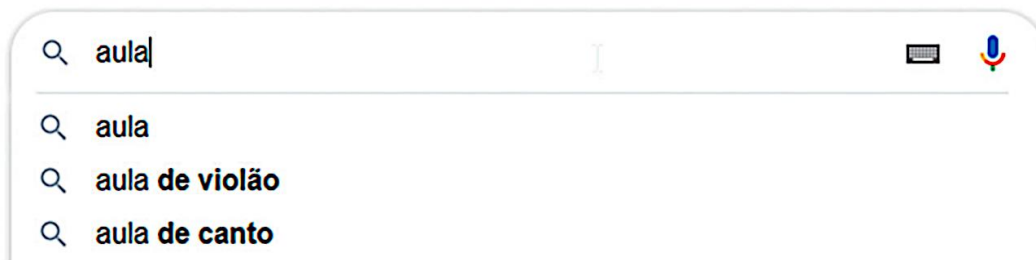
# TECNOLOGIA AJAX

- O termo Ajax foi utilizado inicialmente por [Jessé James Garret](#) em 2005 no artigo '[Ajax: A New Approach to Web Applications](#)' ('Ajax: uma nova abordagem para aplicativos da web'), incorporando as seguintes tecnologias
  - HTML/XHTML e CSS
  - DOM para *display* dinâmico e interação com dados
  - JSON/XML para intercâmbio de dados e [XSLT](#) para manipulação do XML
  - Objeto XMLHttpRequest para comunicação assíncrona
  - JavaScript para unir todas essas tecnologias
- Apesar do nome, a utilização de XML não é obrigatória (JSON é frequentemente utilizado) e as solicitações também não necessitam ser assíncronas



# TECNOLOGIA AJAX

- O Google foi uma das primeiras empresas a usar o Ajax com sua ferramenta de sugestões de pesquisa e de preenchimento automático (*autocomplete*)
- Ao digitar na barra de pesquisa do Google, o *site* usa Ajax para obter resultados comuns do banco de dados, a cada pressionamento de tecla
- O preenchimento automático facilita o uso de formulários em que se têm muitas entradas possíveis, visto que uma lista suspensa seria demorada e complicada



Q aula|

Q aula

Q aula de violão

Q aula de canto

# TECNOLOGIA AJAX

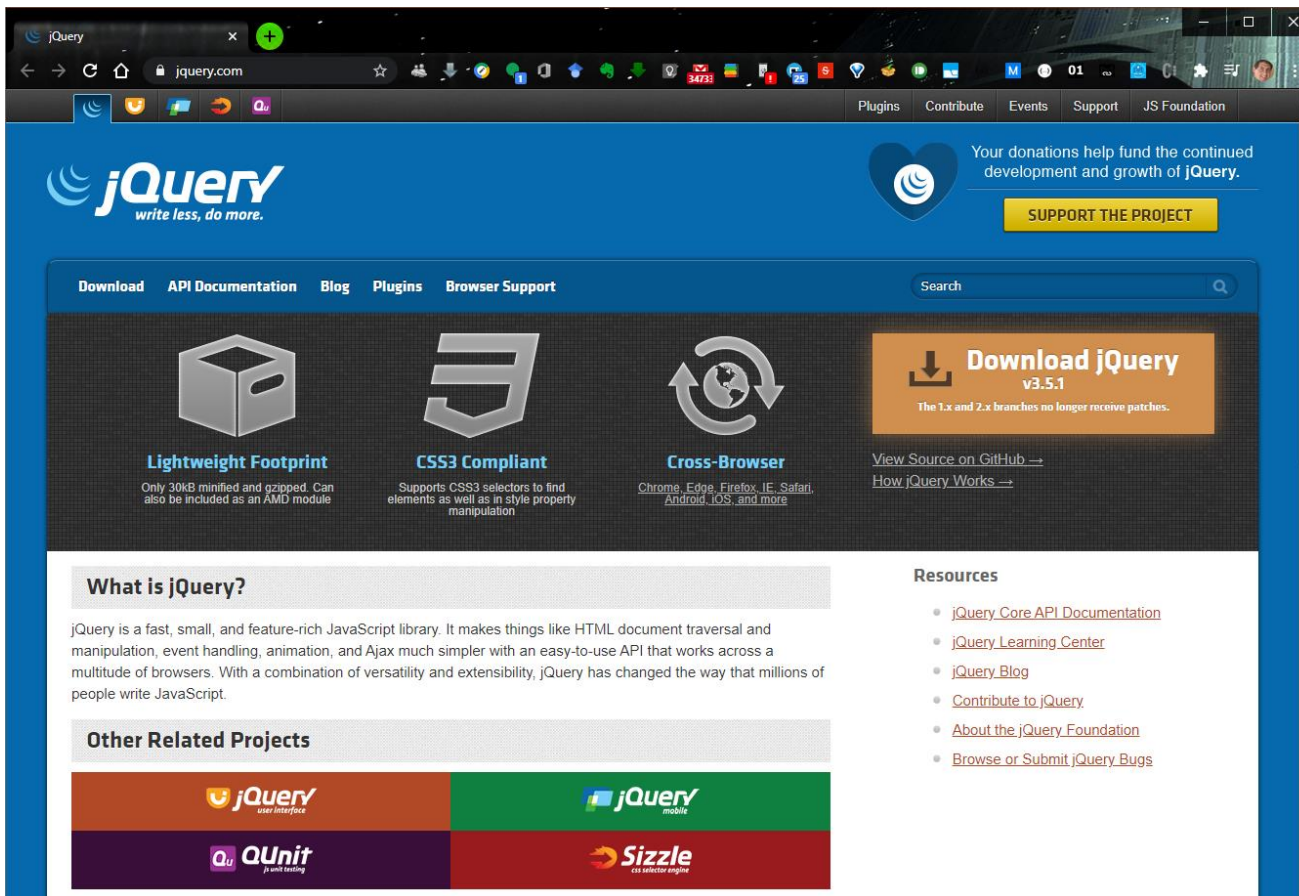
- Mais recentemente foi lançado o [Fetch API](#), que possui funcionalidade semelhante ao XHR com linguagem em mais alto nível
- A Fetch API tem seu funcionamento baseado em [promessas](#)

```
fetch('send-ajax-data.php')  
  .then(data => console.log(data))  
  .catch (error => console.log('Error:' + error));
```

# TECNOLOGIA AJAX

## ■ Frameworks para Ajax

- **jQuery**: Biblioteca rápida, pequena e rica em recursos, que simplifica o processo de manipulação de documentos HTML, efeitos e eventos – é *open source* e crossbrowser



The screenshot shows the jQuery website homepage. At the top, there's a navigation bar with links for Plugins, Contribute, Events, Support, and JS Foundation. Below this is the jQuery logo with the tagline "write less, do more." and a heart icon with the text "Your donations help fund the continued development and growth of jQuery." and a "SUPPORT THE PROJECT" button. The main content area features four large icons: a box for "Lightweight Footprint", a stylized 'E' for "CSS3 Compliant", a circular arrow for "Cross-Browser", and a download icon for "Download jQuery v3.5.1". Below these icons are sections for "What is jQuery?" and "Resources". The "What is jQuery?" section explains that jQuery is a fast, small, and feature-rich JavaScript library. The "Resources" section lists links to jQuery Core API Documentation, jQuery Learning Center, jQuery Blog, Contribute to jQuery, About the jQuery Foundation, and Browse or Submit jQuery Bugs. At the bottom, there's a section for "Other Related Projects" featuring logos for jQuery user interface, jQuery mobile, QUnit, and Sizzle.



# TECNOLOGIA AJAX

## ■ Frameworks para Ajax

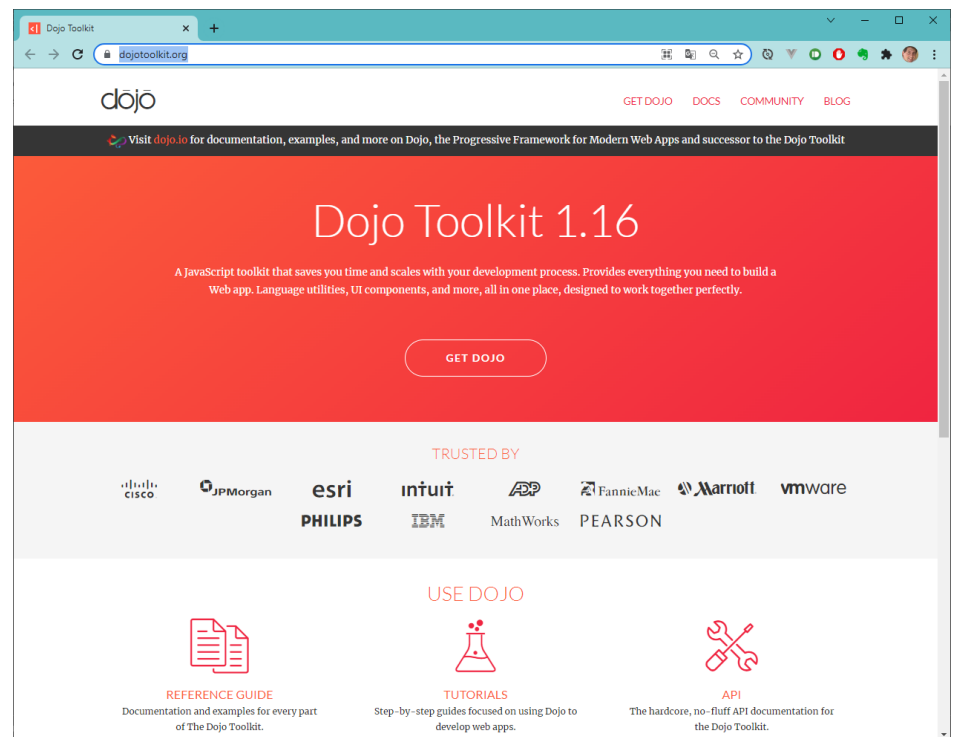
- **Dojo:** Biblioteca (*toolkit*) em JavaScript de código fonte aberto, projetado para facilitar o rápido desenvolvimento de *interfaces* ricas, independente de plataforma servidora

- Para utilizar use o DSN

```
<script src="//ajax.googleapis.com/ajax/libs/dojo/1.14.1/dojo/dojo.js"></script>
```

- Fornece recursos como

- **Menus, abas** e **tooltips** (dicas de contexto)
- Tabelas com ordenação
- Gráficos dinâmicos
- Desenho 2D vetorial
- Efeitos de animação

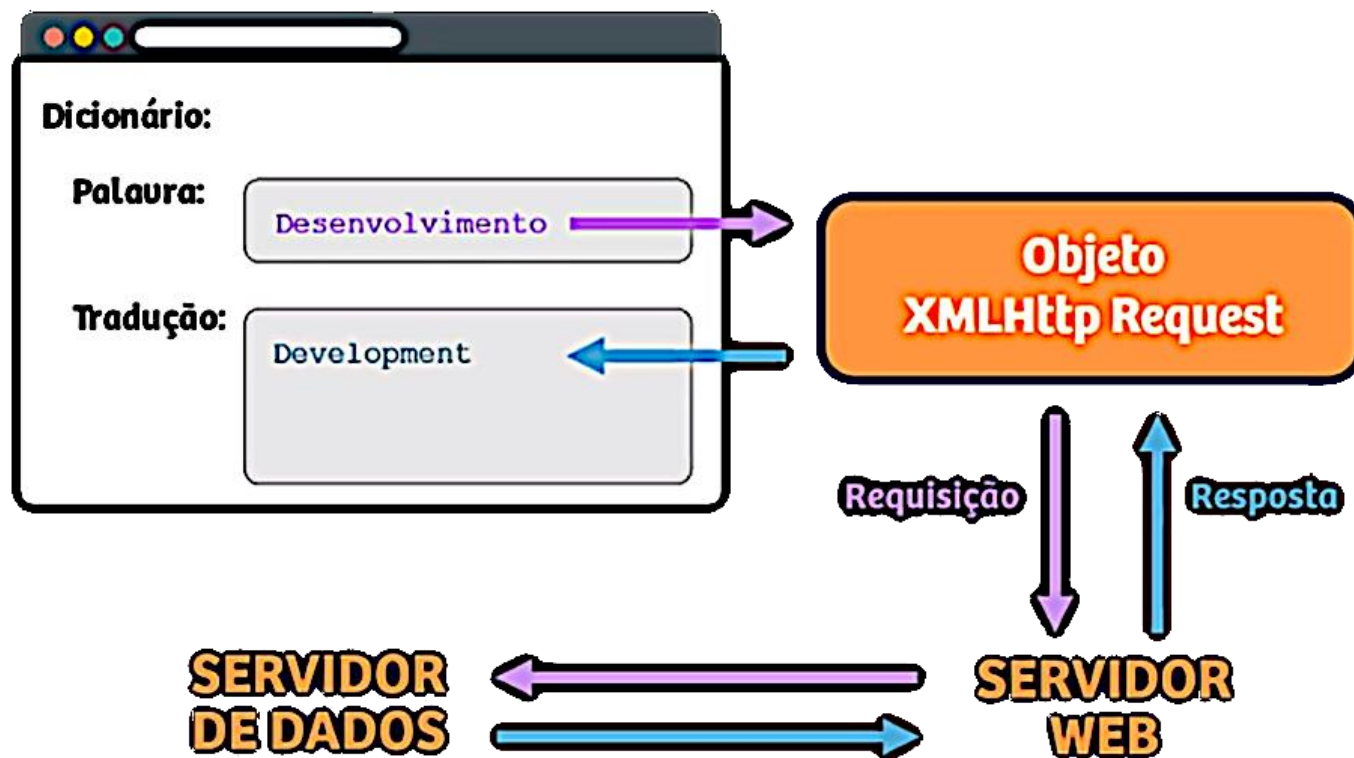


# OBJETO XMLHttpRequest

- [XMLHttpRequest](#) (também chamado de XHR, cuja especificação pode ser lida [aqui](#)) é uma [API](#) em forma de objeto que tem um papel importante na técnica de desenvolvimento web Ajax para se comunicar com os *scripts* do lado do servidor, permitindo trocar dados entre o navegador e o servidor
- O conceito por trás desse objeto foi criado originalmente pelos desenvolvedores do [Outlook Web Access](#) (Microsoft) para o [Microsoft Exchange Server 2000](#)
- Uma [interface](#) chamada `IXMLHttpRequest` foi implementada na segunda versão da biblioteca [MSXML](#) usando este conceito
- A segunda versão dessa biblioteca foi embarcada em 1999 no [Internet Explorer 5.0](#), permitindo acesso, via [ActiveX](#), à interface `IXMLHttpRequest` utilizando um [wrapper](#) `XMLHTTP` da biblioteca `MSXML`
- O objeto `XMLHttpRequest` só foi incluído em 2006 no [Internet Explorer 7.0](#)
- O projeto [Mozilla](#) implementou uma [interface](#) chamada `nsIXMLHttpRequest` no [engine Gecko](#) e um [wrapper](#) para JavaScript chamado `XMLHttpRequest` em 2000, mas que só ficou completamente funcionando em 2002 na versão 1.0 do Gecko
- O objeto `XMLHttpRequest` está atualmente sendo padronizado pela [WHATWG](#)

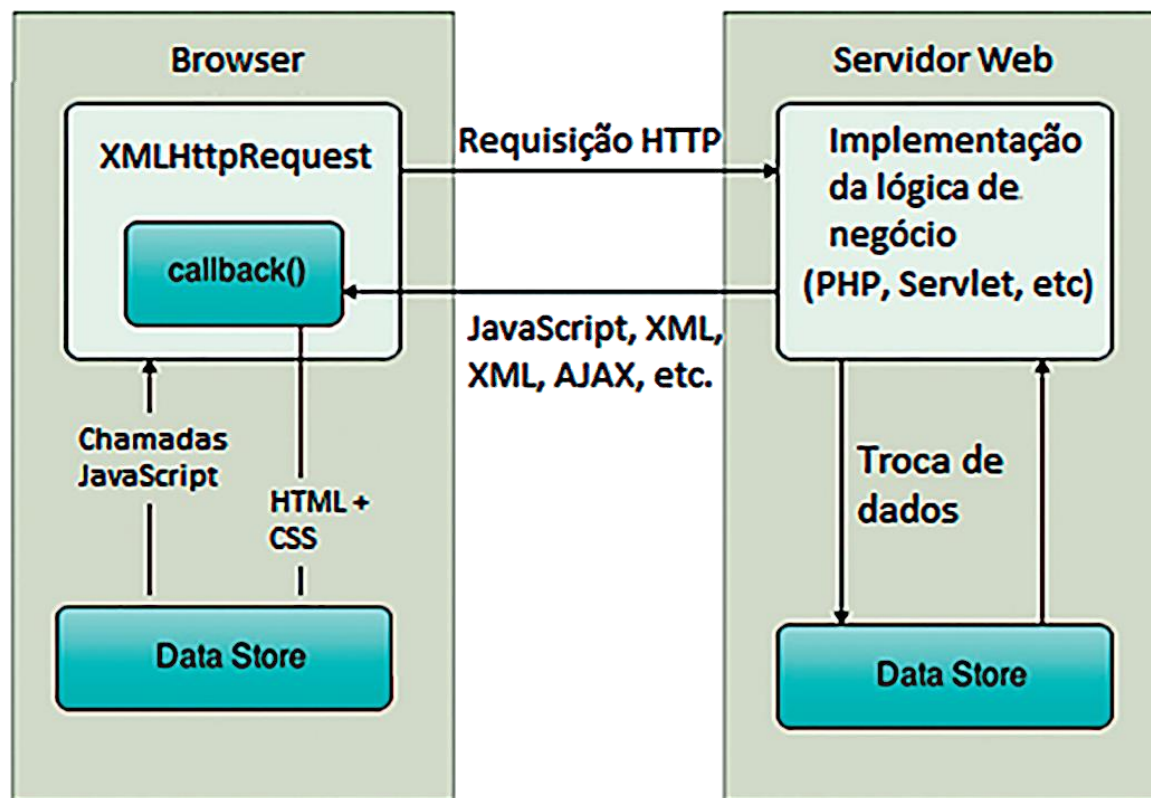
# OBJETO XMLHttpRequest

## ■ Modo de funcionamento



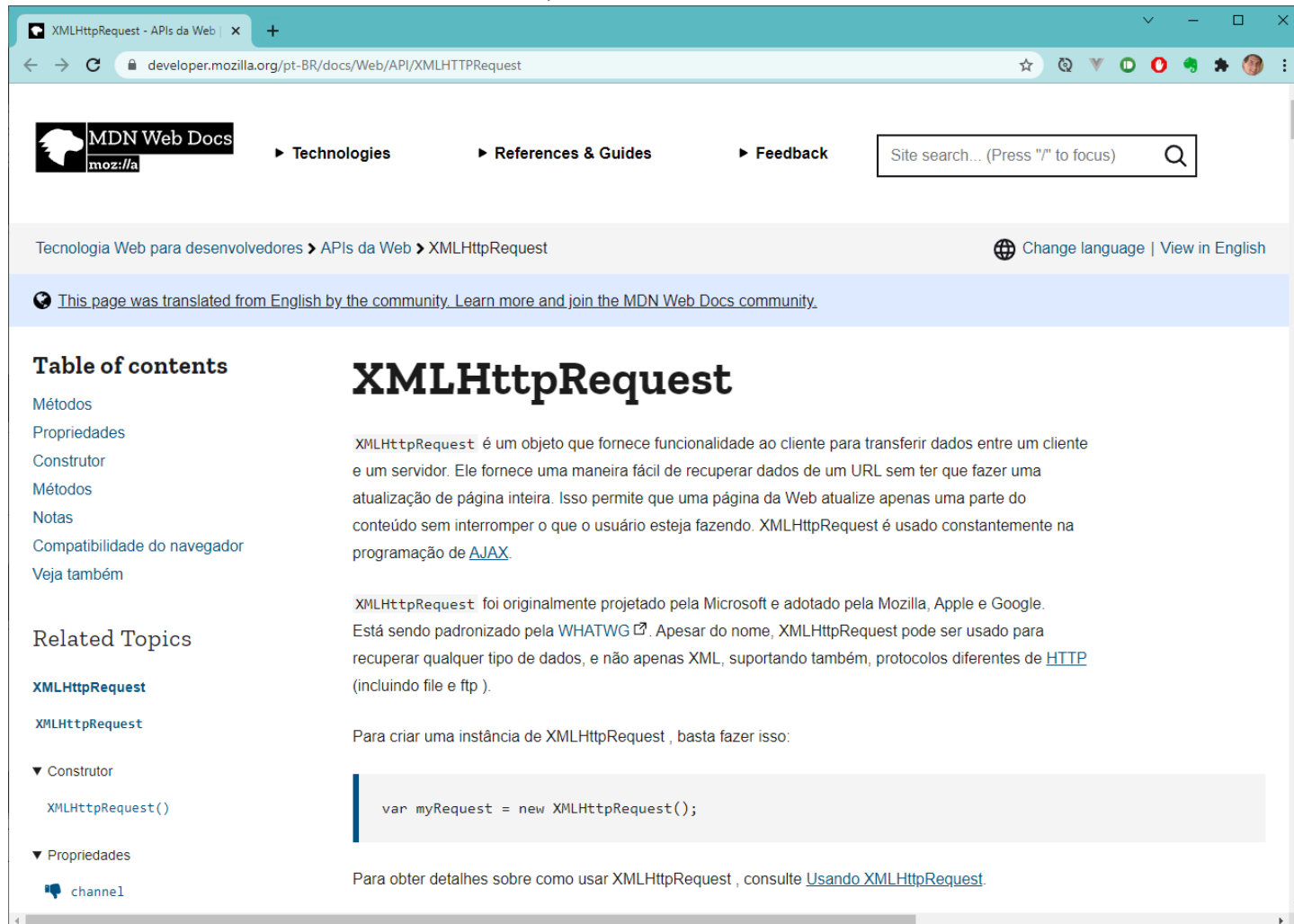
# OBJETO XMLHttpRequest

- Fluxo de funcionamento da troca de dados entre cliente e servidor web pelo objeto XMLHttpRequest



# OBJETO XMLHttpRequest


## ■ Documentação da [Mozilla](#) do objeto XMLHttpRequest





The screenshot shows the MDN Web Docs page for the XMLHttpRequest object. The browser address bar shows the URL: `developer.mozilla.org/pt-BR/docs/Web/API/XMLHttpRequest`. The page header includes the MDN Web Docs logo, navigation links for Technologies, References & Guides, and Feedback, and a search bar. The breadcrumb trail indicates the path: Tecnologia Web para desenvolvedores > APIs da Web > XMLHttpRequest. A notice states that the page was translated from English by the community. The main content area features a 'Table of contents' on the left with links to Métodos, Propriedades, Construtor, Métodos, Notas, Compatibilidade do navegador, and Veja também. The main heading is 'XMLHttpRequest'. The text explains that XMLHttpRequest is an object that provides functionality for transferring data between a client and a server. It mentions that it was originally designed by Microsoft and adopted by Mozilla, Apple, and Google, and is being standardized by WHATWG. A code block shows the syntax for creating an XMLHttpRequest object: `var myRequest = new XMLHttpRequest();`. The page also includes a 'Related Topics' section with a link to 'Usando XMLHttpRequest'.

XMLHttpRequest - APIs da Web | x +

← → ↺ `developer.mozilla.org/pt-BR/docs/Web/API/XMLHttpRequest` ☆ 🔍 🌐 📱 🗖

 **MDN Web Docs** `moz://a` ▶ Technologies ▶ References & Guides ▶ Feedback Site search... (Press "/" to focus) 🔍

Tecnologia Web para desenvolvedores > APIs da Web > XMLHttpRequest  Change language | View in English

 This page was translated from English by the community. [Learn more and join the MDN Web Docs community.](#)

### Table of contents

- Métodos
- Propriedades
- Construtor
- Métodos
- Notas
- Compatibilidade do navegador
- Veja também

## XMLHttpRequest

`XMLHttpRequest` é um objeto que fornece funcionalidade ao cliente para transferir dados entre um cliente e um servidor. Ele fornece uma maneira fácil de recuperar dados de um URL sem ter que fazer uma atualização de página inteira. Isso permite que uma página da Web atualize apenas uma parte do conteúdo sem interromper o que o usuário esteja fazendo. XMLHttpRequest é usado constantemente na programação de [AJAX](#).

`XMLHttpRequest` foi originalmente projetado pela Microsoft e adotado pela Mozilla, Apple e Google. Está sendo padronizado pela [WHATWG](#). Apesar do nome, XMLHttpRequest pode ser usado para recuperar qualquer tipo de dados, e não apenas XML, suportando também, protocolos diferentes de [HTTP](#) (incluindo file e ftp).

Para criar uma instância de XMLHttpRequest, basta fazer isso:

```
var myRequest = new XMLHttpRequest();
```

Para obter detalhes sobre como usar XMLHttpRequest, consulte [Usando XMLHttpRequest](#).

### Related Topics


**XMLHttpRequest**


XMLHttpRequest

▼ Construtor

`XMLHttpRequest()`

▼ Propriedades

 channel





# OBJETO XMLHttpRequest

## ■ Principais métodos do objeto XMLHttpRequest

- `new XMLHttpRequest()`: **Cria um novo objeto** XMLHttpRequest
- `open( method, url, asynchronous, user, password )`: **Requisição específica**
  - `method`: Tipo de requisição GET ou POST
  - `url`: O local do recurso
  - `asynchronous`: `true` (modo assíncrono) ou `false` (modo síncrono)
  - `user`: Nome de usuário opcional
  - `password`: Senha opcional
- `abort()`: **Cancela uma requisição**
- `setRequestHeader( HeaderName, Value )`: **Adiciona um par chave/valor ao cabeçalho sendo enviado**
- `getResponseHeader()`: **Retorna informações específicas do cabeçalho**
- `getAllResponseHeaders()`: **Retorna informações do cabeçalho**
- `send( Data )`: **Envia uma solicitação ao servidor**

# OBJETO XMLHttpRequest

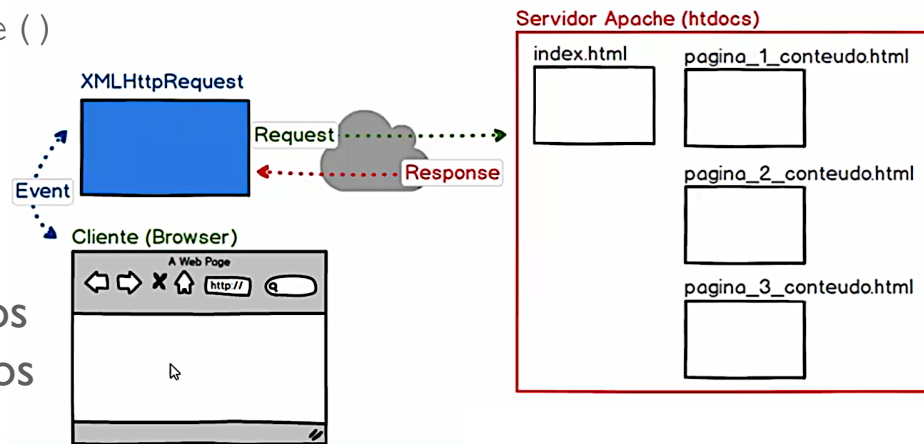
## ■ Principais propriedades do objeto XMLHttpRequest

- `onreadystatechange` Define as funcionalidades para ser chamadas quando uma propriedade for alterada
- `readyState` Mantém o *status* do XMLHttpRequest
- `responseText`: Retorna a resposta em uma *string*
- `responseXML`: Resposta a resposta em formato XML
- `status`: Retorna o *status* numérico das requisições, por exemplo: 200 para “OK”, 403 para “Proibido”; 404 para “Não encontrado”, etc
- `statusText`: Retorna o *status* incluindo número e texto, por exemplo: 200 OK, 403 Forbidden, 404 Not Found, etc

# OBJETO XMLHttpRequest

- O *event listener* `onreadystatechange()` é chamado sempre que o atributo `readyState` sofre alguma alteração

- Cada requisição passa por cinco estados diferentes, que são os valores mostrados abaixo para `readyState`



Valor	Estado	Descrição
0	UNSENT	<code>open()</code> não foi chamado ainda; solicitação não iniciada
1	OPENED	<code>send()</code> não foi chamado ainda; conexão estabelecida
2	HEADERS_RECEIVED	<code>send()</code> foi chamado, e cabeçalhos e <i>status</i> estão disponíveis; requisição foi recebida pelo servidor
3	LOADING	Requisição em processamento; <code>responseText</code> contém dados parciais ( <i>download</i> )
4	DONE	A operação está concluída; requisição finalizada

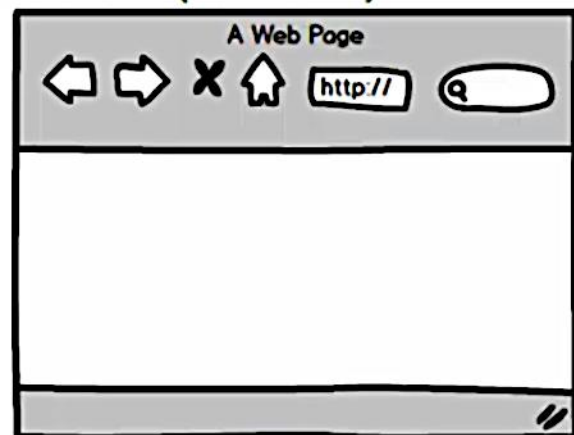
# OBJETO XMLHttpRequest

- Além do estado (`readyState`) há também o *status* da requisição
  - O *status* pode indicar, por exemplo, que o recurso solicitado não existe no servidor
  - Isso é independente do estado da requisição, ou seja, a requisição pode ter sido efetuada com sucesso mas não necessariamente a resposta será a desejada
  - Por isso é que existem dois controles diferentes, o estado e o *status*

# PRÁTICA

- Requisições síncronas: *Refresh* de toda a página

Cliente (Browser)



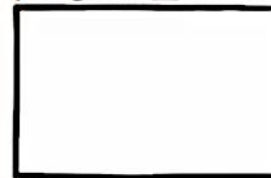
Request



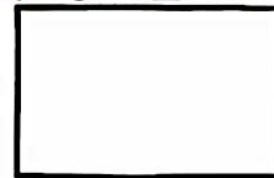
Response

Servidor Apache (htdocs)

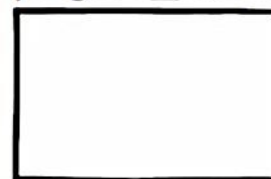
pagina\_1.html



pagina\_3.html

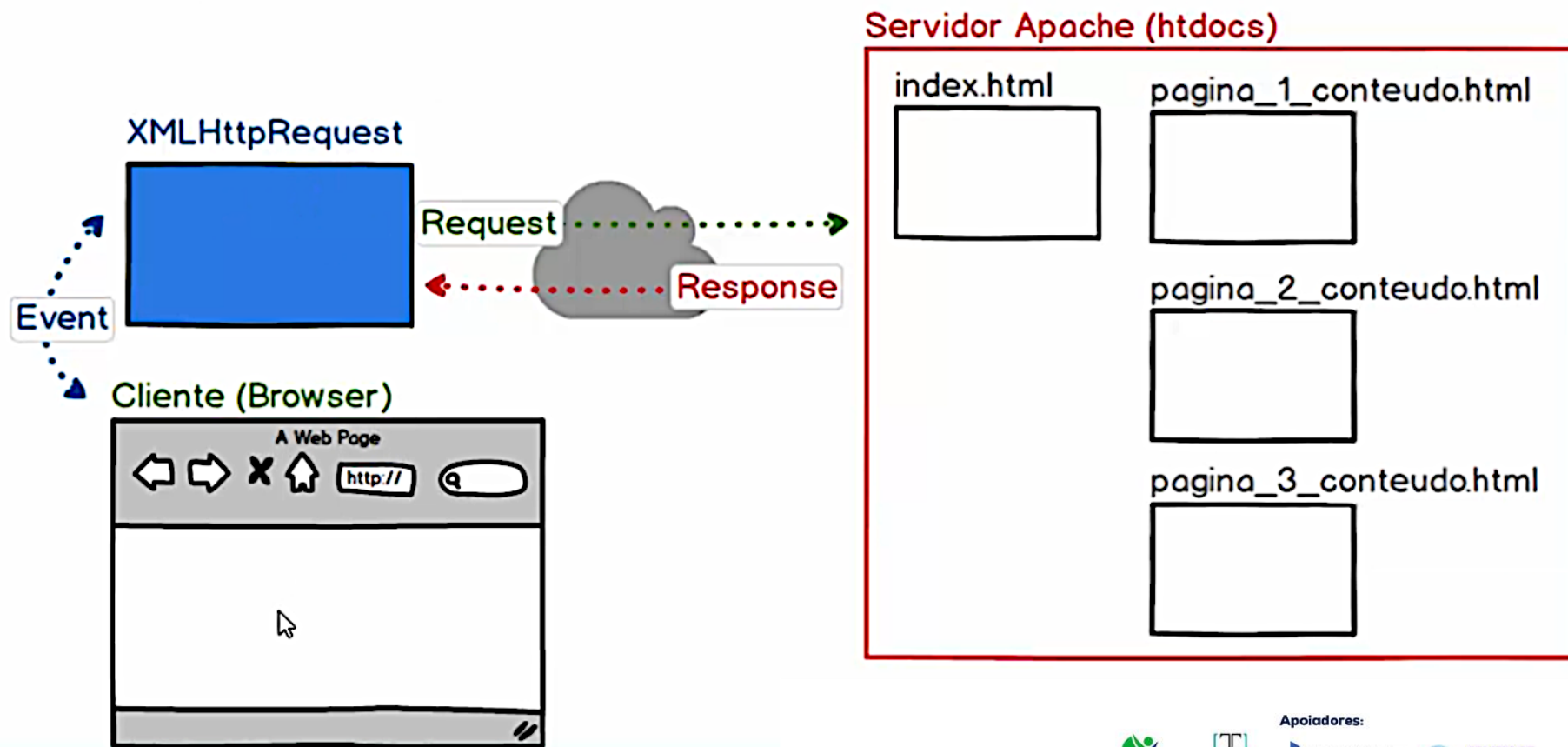


pagina\_2.html



# PRÁTICA

- Requisições assíncronas: *Refresh* de toda a página



# PRÁTICA

## Requisições assíncronas

Requisições Assíncronas

127.0.0.1:5500/requisicoes\_assincronas.html#

### Requisições síncronas e assíncronas

Página 1 Página 2 Página 3

Elementos Console Fontes Rede Desempenho Memória Aplicativo Segurança

Filtro Inverter Ocultar URLs de dados

Tudo Fetch/XHR JS CSS Img Mídia Fonte Doc WS Wasm Manifesto Outro Bloqueou os cookies Solicitações bloqueadas

Solicitações de terceiros

100ms 200ms 300ms 400ms 500ms 600ms 700ms 800ms 900ms 1000ms 1100ms 1200ms

Nome	X	Cabeçalhos	Visualização	Resposta	Iniciador	Tempo
requisicoes_assincronas.html	1	<div class="panel panel-default">				
bootstrap.min.css	2	<div class="panel-body">				
pagina_1_conteudo.html	3	<h4>PÁGINA 1</h4>				
ws	4	<hr>				
	5	<p>				
	6	Lorem ipsum accumsan vulputate magna etiam nulla condimentum adipiscing, l				
	7	</p>				
	8	<p>				
	9	Mollis elit libero potenti iaculis himenaeos curabitur ornare metus, vel r				
	10	</p>				
	11	</div>				
	12	</div>				
	13					
	14					
	15					

4 solicitações 3.5 kB transferidos Linha 4, coluna 1

```
1 <!DOCTYPE HTML>
2 <html lang="pt-BR">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta http-equiv="X-UA-Compatible" content="IE=edge">
7   <meta name="viewport" content="width=device-width, initial-scale=1.0">
8
9   <title>Requisições Assíncronas</title>
10
11   <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.2/dist/css/boot
strap.min.css" rel="stylesheet"
12     integrity="sha384-Zenh87qX5JnK2J10vWa8Ck2rdkQ2Bzep5IDxbcnCeu0xjzrP
F/et3URY9Bv1WTRI" crossorigin="anonymous">
13
14   <script>
15     let ajax = new XMLHttpRequest();
16
17     ajax.open('GET', 'pagina_1_conteudo.html');
18     ajax.send();
19
20     console.log(ajax);
21   </script>
22 </head>
23
24 <body>
25   <nav class="navbar navbar-light bg-light mb-4">
26     <div class="container">
27       <div class="navbar-brand mb-0 h1">
28         <h3>Requisições síncronas e assíncronas</h3>
29       </div>
30     </div>
31   </nav>
32
33   <div class="container">
34     <div class="row mb-2 justify-content-center">
35       <div class="col"></div>
36       <div class="col-auto">
37         <a href="#" class="btn btn-primary">Página 1</a>
38         <a href="#" class="btn btn-primary">Página 2</a>
39         <a href="#" class="btn btn-primary">Página 3</a>
40       </div>
41     </div>
42   </div>
43
44   <div class="row">
45     <div class="col-md-1"></div>
46
47     <div class="col-md-10">
48
49
50
51     <div class="col-md-1"></div>
52   </div>
53 </body>
54 </html>
```

# PRÁTICA

## ■ Requisições assíncronas

Requisições síncronas e assíncronas

Página 1 Página 2 Página 3

requisicoes\_assincronas.html:19

```
▼ XMLHttpRequest {onreadystatechange: null, readyState: 4, timeout: 0, withCredentials: false, upload: XMLHttpRequestUpload, ...}
  onabort: null
  onerror: null
  onload: null
  onloadend: null
  onloadstart: null
  onprogress: null
  onreadystatechange: null
  ontimeout: null
  readyState: 4
  response: "<div class=\"panel panel-default\">\r\n\t<div class=\"panel-body\">\r\n\t\t<h4>PÁGINA 1</h4>\r\n\r\n\t\t<hr>\r\n\r\n\t\t<p>\r\n'
  responseText: "<div class=\"panel panel-default\">\r\n\t<div class=\"panel-body\">\r\n\t\t<h4>PÁGINA 1</h4>\r\n\r\n\t\t<hr>\r\n\r\n\t\t<p>
  responseType: ""
  responseURL: "http://127.0.0.1:5500/pagina_1_conteudo.html"
  responseXML: null
  status: 200
  statusText: "OK"
  timeout: 0
  ▶ upload: XMLHttpRequestUpload {onloadstart: null, onprogress: null, onabort: null, onerror: null, onload: null, ...}
    withCredentials: false
  ▶ [[Prototype]]: XMLHttpRequest
```



# PRÁTICA

## ■ Requisições assíncronas

### ■ Encapsulando em uma função a requisição

### ■ Eventos onclick nos hiperlinks

```
1 <!DOCTYPE HTML>
2 <html lang="pt-BR">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta http-equiv="X-UA-Compatible" content="IE=edge">
7   <meta name="viewport" content="width=device-width, initial-scale=1.0">
8
9   <title>Requisições Assíncronas</title>
10
11   <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.2/dist/css/bootstrap.min.css" rel="stylesheet" integrity=
"sha384-Zenh87qX5JnK2Jl0vWa8Ck2rdkQ2Bzep5IDxbcnCeuOxjzrPF/et3Ury9Bv1WTRi" crossorigin="anonymous">
12
13   <script>
14     function requisitarPagina(url) {
15       let ajax = new XMLHttpRequest();
16
17       ajax.open('GET', url);
18       ajax.send();
19
20       console.log(ajax);
21     }
22   </script>
23 </head>
24
25 <body>
26   <nav class="navbar navbar-light bg-light mb-4">
27     <div class="container">
28       <div class="navbar-brand mb-0 h1">
29         <h3>Requisições síncronas e assíncronas</h3>
30       </div>
31     </div>
32   </nav>
33
34   <div class="container">
35     <div class="row mb-2 justify-content-center">
36       <div class="col"></div>
37       <div class="col-auto">
38         <a href="#" class="btn btn-primary" onclick="requisitarPagina('pagina_1_conteudo.html')">Página 1</a>
39         <a href="#" class="btn btn-primary" onclick="requisitarPagina('pagina_2_conteudo.html')">Página 2</a>
40         <a href="#" class="btn btn-primary" onclick="requisitarPagina('pagina_3_conteudo.html')">Página 3</a>
41       </div>
42     </div>
43
44     <div class="row">
45       <div class="col-md-1"></div>
46
47       <div class="col-md-10">
48
49
50
51
52       </div>
53     </div>
54   </div>
55 </body>
56
57 </html>
```

# PRÁTICA

## ■ Estado 0 do readyState

```
13     <script>
14         function requisitarPagina(url) {
15             let ajax = new XMLHttpRequest();
16
17             // Requisição não iniciada, readyState = 0
18             console.log('Estado readyState:', ajax.readyState);
19
20             ajax.open('GET', url);
21             ajax.send();
22
23             console.log(ajax);
24         }
25     </script>
```

Requisições Assíncronas

127.0.0.1:5500/requisicoes\_assincronas.html#

### Requisições síncronas e assíncronas

Página 1 Página 2 Página 3

Elementos Console Fontes Rede Desempenho Memória Aplicativo Segurança Lighthouse

Estado readyState: 0 [requisicoes\\_assincronas.html:18](#)

[requisicoes\\_assincronas.html:23](#)

► XMLHttpRequest {onreadystatechange: null, readyState: 1, timeout: 0, withCredentials: false, upload: XMLHttpRequestUpload, ...}

# PRÁTICA

## ■ Estado 1 do readyState

```
13 <script>
14   function requisitarPagina(url) {
15     let ajax = new XMLHttpRequest();
16
17     // Requisição não iniciada, readyState = 0
18     console.log('Estado readyState:', ajax.readyState);
19
20     // Conexão estabelecida com o servidor, readyState = 1
21     ajax.open('GET', url);
22     console.log('Estado readyState:', ajax.readyState);
23
24     ajax.send();
25
26     console.log(ajax);
27   }
28 </script>
```

Requisições Assíncronas

127.0.0.1:5500/requisicoes\_assincronas.html#

## Requisições síncronas e assíncronas

Página 1 Página 2 Página 3

Elementos Console Fontes Rede Desempenho Memória Aplicativo Segurança Lighthouse

top Filtro Níveis padrão 1 problema: 1

Estado readyState: 0 [requisicoes\\_assincronas.html:18](#)

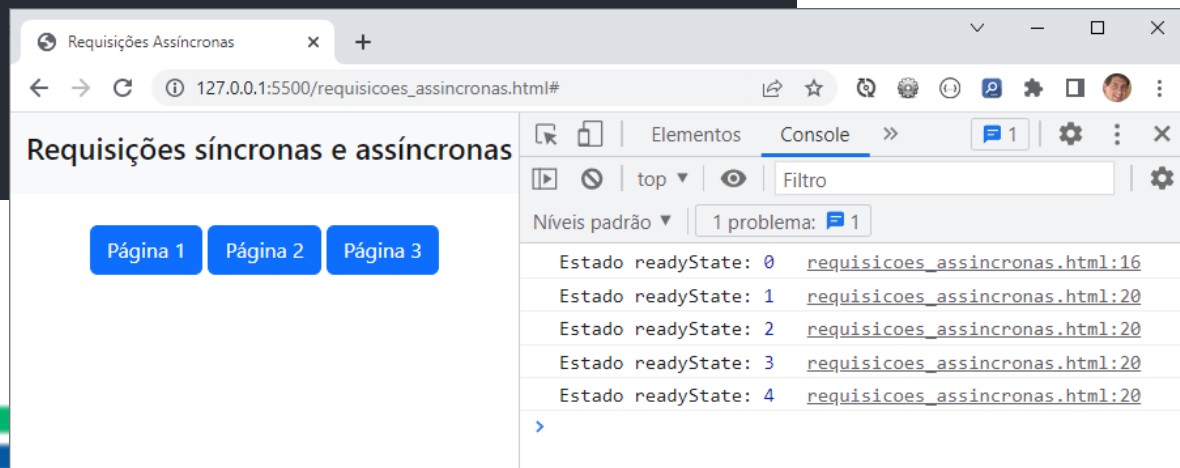
Estado readyState: 1 [requisicoes\\_assincronas.html:22](#)

► XMLHttpRequest {onreadystatechange: null, readyState: 1, timeout: 0, withCredentials: false, upload: XMLHttpRequestUpload, ...} [requisicoes\\_assincronas.html:26](#)

# PRÁTICA

- Configurando o evento `onreadystatechange` para que seja disparado sempre quando o estado da conexão for modificado
- Na prática o evento que é utilizado é o 4 – ao finalizar a requisição pode-se recuperar a resposta fornecida pelo servidor e realizar alguma ação com ela

```
13 <script>
14   function requisitarPagina(url) {
15     let ajax = new XMLHttpRequest();
16     console.log('Estado readyState:', ajax.readyState);
17
18     // Evento que é disparado quando o estado da conexão é modificado
19     ajax.onreadystatechange = () => {
20       console.log('Estado readyState:', ajax.readyState);
21     }
22
23     ajax.open('GET', url);
24     ajax.send();
25   }
26 </script>
```



# PRÁTICA

- Dentro da pasta chamada *aula22* crie outra com o nome *public*
- Mova os arquivos *html* para dentro dessa pasta
- Abra o terminal do VS Code e execute os seguintes comandos
  - `npm init -y`
  - `npm install express`
  - `npm install body-parser`
  - `npm install cors`
  - `npm install --save-dev nodemon`
- Acrescentar no *package.json*
  - `"scripts": {`
  - `"dev": "nodemon server.js",`
  - `"test": "echo \"Error: no test specified\" && exit 1"`
  - `}`,

# PRÁTICA

- Crie o arquivo `server.js` no diretório raiz do projeto
- Configuração do CORS (Cross-Origin Resource Sharing): Mecanismo que permite que recursos restritos em uma página web sejam recuperados por outro domínio fora do domínio ao qual pertence o recurso que será recuperado)
- Para executar
  - `npm run dev`

```
aula22 - original - server.js

1   require('cors')
2
3   const app = express();
4   const porta = 8080;
5
6   // Configura o CORS
7   app.use(cors({origin: '*}));
8
9   // Instancia o servidor
10  app.listen(porta,
11    () => console.log(`Servidor iniciado na porta: ${porta}`)
12  );
13
14  function sleep(ms) {
15    return new Promise(
16      (resolve) => setTimeout(resolve, ms)
17    );
18  }
19
20  // Tratamento da requisição GET
21  app.get('/processa-requisicao', function(req, res) {
22    sleep(3000).then(() => {
23      res.status(200).send('<h2>Olá Mundo</h2>');
24    });
25  });
```

# PRÁTICA

- Nova versão da página
- GIF de carregamento
- Requisição ao servidor
- Tratamento da resposta
- Mensagens de erro

```
1 <!DOCTYPE HTML>
2 <html lang="pt-BR">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta http-equiv="X-UA-Compatible" content="IE=edge">
7   <meta name="viewport" content="width=device-width, initial-scale=1.0">
8
9   <title>Requisições Assíncronas</title>
10
11   <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.2/dist/css/bootstrap.min.css" rel="stylesheet"
12     integrity="sha384-Zenh87qX5JnK2Jl0vWa8Ck2rdkQ2Bzep5IDxbcnCeuOxjzrPF/et3URy9Bv1WTRI" crossorigin="anonymous">
13
14   <script>
15     function requisitarPagina(url) {
16       document.querySelector('#conteudo').innerHTML = '';
17
18       // Carregando o arquivo de imagem
19       if (!document.querySelector('#carregamento')) {
20         let img = document.createElement('img');
21         img.id = 'carregamento';
22         img.src = './loading.gif';
23         img.className = 'rounded mx-auto d-block';
24         document.querySelector('#conteudo').appendChild(img);
25       }
26
27       let ajax = new XMLHttpRequest();
28       console.log('Estado readyState:', ajax.readyState);
29
30       // Evento que é disparado quando o estado da conexão é modificado
31       ajax.onreadystatechange = () => {
32         console.log('Estado readyState:', ajax.readyState);
33
34         if (ajax.readyState == 4)
35           if (ajax.status == 200) {
36             console.log('Requisição finalizada com sucesso!');
37             console.log('O status da resposta é:', ajax.status);
38             document.querySelector('#carregamento').remove();
39           } else if (ajax.status == 404) {
40             document.querySelector('#conteudo').innerHTML = '<h3>Recurso não encontrado!</h3>';
41             document.querySelector('#conteudo').innerHTML += 'O status da resposta é ' + ajax.status;
42             console.log('Recurso não encontrado!');
43           }
44       }
45
46       ajax.open('GET', url);
47       ajax.send();
48     }
49   </script>
50 </head>
```

# PRÁTICA

- Nova versão da página
  - GIF de carregamento
  - Requisição ao servidor
  - Tratamento da resposta
  - Mensagens de erro

```
52 <body>
53   <nav class="navbar navbar-light bg-light mb-4">
54     <div class="container">
55       <div class="navbar-brand mb-0 h1">
56         <h3>Requisições síncronas e assíncronas</h3>
57       </div>
58     </div>
59   </nav>
60
61   <div class="container">
62     <div class="row mb-2 justify-content-center">
63       <div class="col"></div>
64       <div class="col-auto">
65         <a href="#" class="btn btn-primary"
66           onclick="requisitarPagina('pagina_1_conteudo.html')">
67           Página 1
68         </a>
69
70         <a href="#" class="btn btn-primary"
71           onclick="requisitarPagina('pagina_2_conteudo.html')">
72           Página 2
73         </a>
74
75         <a href="#" class="btn btn-primary"
76           onclick="requisitarPagina('pagina_3_conteudo.html')">
77           Página 3
78         </a>
79
80         <a href="#" class="btn btn-primary"
81           onclick="requisitarPagina('http://127.0.0.1:8080/processa-requisicao')">
82           Servidor
83         </a>
84       </div>
85     </div>
86
87   <div class="row">
88     <div class="col-md-1"></div>
89     <div class="col-md-10" id="conteudo"></div>
90     <div class="col-md-1"></div>
91   </div>
92 </body>
93
94 </html>
```



# PRÁTICA

## ■ Resposta no navegador

The screenshot shows a web browser window with the address bar displaying '127.0.0.1:5500/public/requisicoes\_assincronas.html#'. The page content is titled 'Requisições síncronas e assíncronas' and features four buttons: 'Página 1', 'Página 2', 'Página 3', and 'Servidor'. The browser's developer console is open, showing the following log entries:

Estado readyState:	URL
0	<a href="#">requisicoes_assincronas.html:25</a>
1	<a href="#">requisicoes_assincronas.html:29</a>
2	<a href="#">requisicoes_assincronas.html:29</a>
3	<a href="#">requisicoes_assincronas.html:29</a>
4	<a href="#">requisicoes_assincronas.html:29</a>
Requisição finalizada com sucesso!	<a href="#">requisicoes_assincronas.html:32</a>
O status da resposta é: 200	<a href="#">requisicoes_assincronas.html:33</a>

# PRÁTICA

## ■ Resposta no navegador

The screenshot shows a web browser window with the address bar displaying `127.0.0.1:5500/public/requisicoes_assincronas.html#`. The page content includes four buttons: 'Página 1', 'Página 2', 'Página 3', and 'Servidor'. The browser's developer tools are open, showing the 'Rede' (Network) tab. A timeline at the top of the network panel shows a request starting at 1000ms and ending at approximately 6500ms. The list of requests includes `requisicoes_assincronas.html`, `bootstrap.min.css`, `ws`, `loading.gif`, and `processa-requisicao`. The details for the `processa-requisicao` request are expanded, showing a status of 200 OK and a response URL of `http://127.0.0.1:8080/processa-requisicao`.

Requisições síncronas e assíncronas

Página 1 Página 2 Página 3 Servidor

Nome

- requisicoes\_assincronas.html
- bootstrap.min.css
- ws
- loading.gif
- processa-requisicao

5 solicitações | 4.8 kB transferidos

Nome Cabeçalhos Visualização Resposta Iniciador Tempo

▼ Geral

Solicitar URL: `http://127.0.0.1:8080/processa-requisicao`

Método da solicitação: GET

Código de status: 200 OK

Endereço remoto: 127.0.0.1:8080

Política do referenciador: strict-origin-when-cross-origin

# PRÁTICA

## ■ Resposta no navegador

The screenshot shows a web browser window with the address bar displaying `127.0.0.1:5500/public/requisicoes_assincronas.html#`. The page content includes buttons for "Página 1", "Página 2", "Página 3", and "Servidor". Below these buttons, the text "Recurso não encontrado!" is displayed, followed by "O status da resposta é 404".

The Chrome DevTools Network tab is open, showing a list of requests. The request `processa-requisicao` is highlighted in red, indicating a 404 status. The details panel for this request shows the following information:

- Solicitar URL: `http://127.0.0.1:8080/processa-requisicao`
- Método da solicitação: GET
- Código de status: 404 Not Found
- Endereço remoto: 127.0.0.1:8080
- Política do referenciador: strict-origin-when-cross-origin

The screenshot shows the same web browser window as above, but with the Chrome DevTools Console tab open. The console displays the following log entries:

- Estado readyState: 0
- Estado readyState: 1
- ✖ GET `http://127.0.0.1:8080/processa-requisicao` 404 (Not Found)
- Estado readyState: 2
- Estado readyState: 3
- Estado readyState: 4
- Recurso não encontrado!

The console also shows the corresponding file locations for each log entry, such as `requisicoes_assincronas.html:27` and `requisicoes_assincronas.html:41`.

# PRÁTICA

- Versão final com carregamento da resposta vinda do servidor na div

```
1 <!DOCTYPE HTML>
2 <html lang="pt-BR">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta http-equiv="X-UA-Compatible" content="IE=edge">
7   <meta name="viewport" content="width=device-width, initial-scale=1.0">
8
9   <title>Requisições Assíncronas</title>
10
11   <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.2/dist/css/bootstrap.min.css" rel="stylesheet"
12     integrity="sha384-Zenh87qX5JnK2Jl0vWa8Ck2rdkQ2Bzep5IDxbcnCeu0xjzrPF/et3URy9Bv1WTRI" crossorigin="anonymous">
13
14   <script>
15     function requisitarPagina(url) {
16       document.querySelector('#conteudo').innerHTML = '';
17
18       // Carregando o arquivo de imagem
19       if (!document.querySelector('#carregamento')) {
20         let img = document.createElement('img');
21         img.id = 'carregamento';
22         img.src = './loading.gif';
23         img.className = 'rounded mx-auto d-block';
24         document.querySelector('#conteudo').appendChild(img);
25       }
26
27       let ajax = new XMLHttpRequest();
28       console.log('Estado readyState:', ajax.readyState);
29
30       // Evento que é disparado quando o estado da conexão é modificado
31       ajax.onreadystatechange = () => {
32         console.log('Estado readyState:', ajax.readyState);
33
34         if (ajax.readyState == 4)
35           if (ajax.status == 200) {
36             document.querySelector('#conteudo').innerHTML = ajax.responseText;
37             console.log('Requisição finalizada com sucesso!');
38             console.log('O status da resposta é:', ajax.status);
39           } else if (ajax.status == 404) {
40             document.querySelector('#conteudo').innerHTML = '<h3>Recurso não encontrado!</h3>';
41             document.querySelector('#conteudo').innerHTML += 'O status da resposta é ' + ajax.status;
42             console.log('Recurso não encontrado!');
43           }
44       }
45
46       ajax.open('GET', url);
47       ajax.send();
48     }
49   </script>
50 </head>
```

# JSON

## ■ JSON: JavaScript Object Notation

- Formato aberto e compacto de dados, muito utilizado para troca de dados entre sistemas, principalmente em web services utilizando REST e AJAX
- Substitui o XML
- Criado em 2000 por Douglas Crockford e normatizado em 2013
- Legível para humanos, utilizando o formato `chave:valor`



```
1 { "Alunos": [  
2     { "nome": "Henrique", "notas": [ 8, 9, 5 ] },  
3     { "nome": "Chris", "notas": [ 8, 10, 7 ] },  
4     { "nome": "Martin", "notas": [ 10, 10, 9 ] }  
5 ] }
```

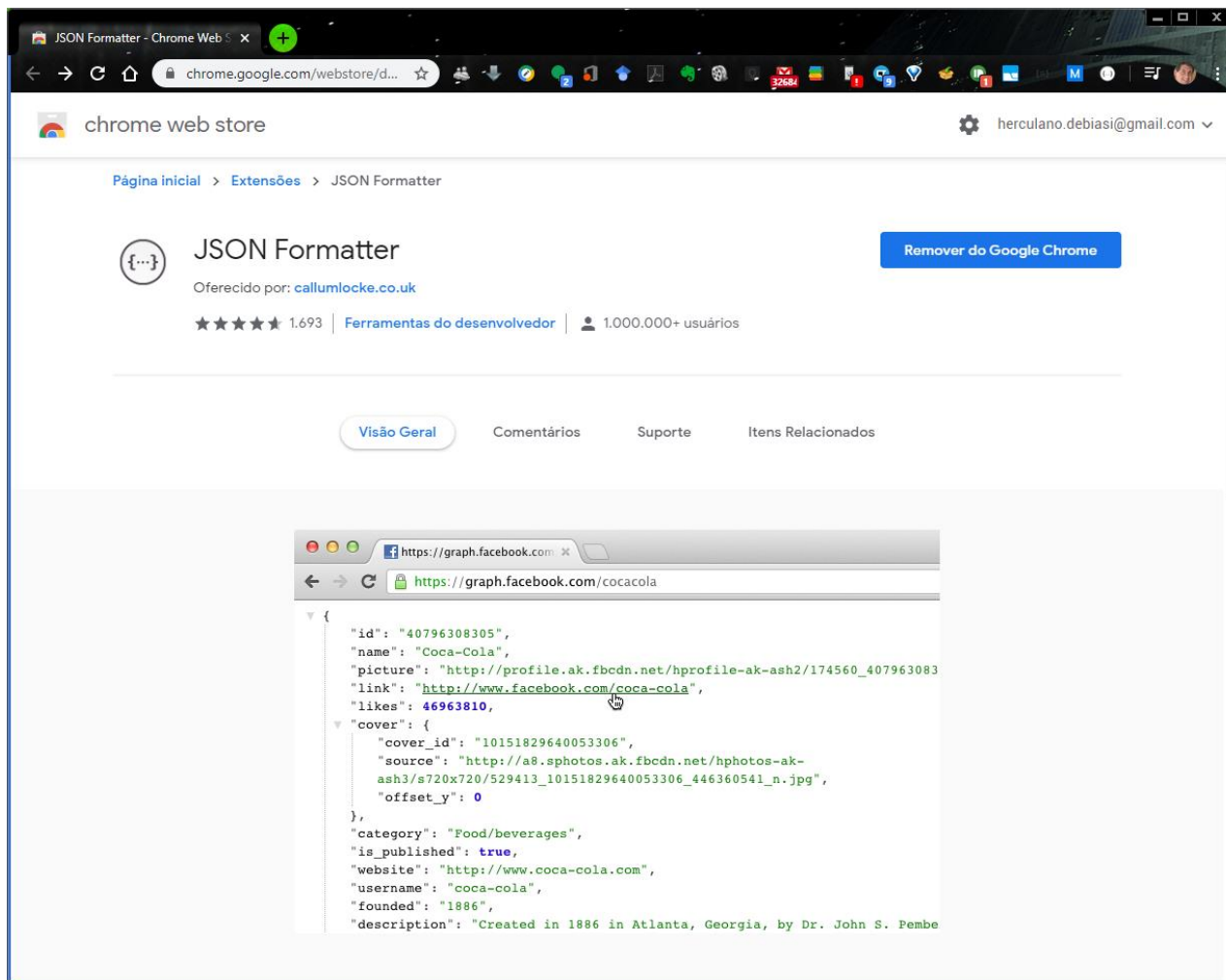
# JSON

- O JavaScript possui suporte embutido para o formato JSON

```
Prompt de Comando - node
> var pessoa={
...   "primeiroNome": "João",
...   "sobreNome": "da Silva",
...   "idade": 25,
...   "endereco": {
...     "rua": "Av.Barão do Rio Branco",
...     "cidade": "Caçador",
...     "estado": "SC",
...     "codigoPostal": "89500-000"
...   },
...   "telefones": [
...     { "tipo": "casa", "numero": "49 123-1234" },
...     { "tipo": "fax", "numero": "49 456-4567" }
...   ],
... }
undefined
> console.log(pessoa.primeiroNome)
João
undefined
>
```

# JSON

## ■ Extensão para formatar JSON no Chrome



# JSON COM JAVASCRIPT

- JSON (JavaScript Object Notation) é um formato para troca de dados bastante empregado atualmente
- JSON é um subconjunto da notação de objetos em JavaScript, o que faz os dados em JSON serem facilmente manipulados pela linguagem JavaScript
- A notação JSON é bastante simples, sendo os dados escritos como pares de chaves e valores separados por vírgulas

```
{ "nome": "Caroline", "sobrenome": "Silva" }
```

- Se a aplicação necessitar de um *array* de objetos, é mandatário acrescentar colchetes como delimitadores da coleção

```
[ { "nome": "Caroline", "sobrenome": "Silva" },  
  { "nome": "Cristiane", "sobrenome": "Machado" } ]
```

- Por meio de uma notação simples, os dados em JSON podem representar coleções inteiras de objetos



# JSON COM JAVASCRIPT

- Para converter os dados no formato JSON em objetos JavaScript, é necessário utilizar as funções `eval` ou `parse`
  - A função `eval()` é capaz de avaliar qualquer expressão JavaScript
  - Seu uso sem uma análise cuidadosa pode resultar em uma falha de segurança, pois a aplicação pode receber um código malicioso que será posteriormente executado
  - O método `stringify()` recebe um objeto JSON e o serializa, ou seja, o transforma em uma *string*
  - O método `parse()` faz o inverso de `stringify()`, ele analisa uma *string* JSON e a transforma em um objeto em JavaScript
  - `parse()` interpreta apenas expressões em JSON, ou seja, não possui a capacidade de executar comandos do JavaScript

# JSON COM JAVASCRIPT

## ■ Nova versão do servidor

```
1  const express = require('express');
2  const cors = require('cors')
3
4  const app = express();
5  const porta = 8080;
6
7  // Configura o CORS
8  app.use(cors({origin: '*})));
9
10 // Instancia o servidor
11 app.listen(porta,
12   () => console.log(`Servidor iniciado na porta: ${porta}`)
13 );
14
15 function sleep(ms) {
16   return new Promise(
17     (resolve) => setTimeout(resolve, ms)
18   );
19 }
20
21 // Tratamento da requisição GET
22 app.get('/processa-requisicao', function(req, res) {
23   sleep(3000).then(() => {
24     res.status(200).send('<h2>Olá Mundo</h2');
25   });
26 });
27
28 // Tratamento da requisição GET
29 app.get('/devolve-json', function(req, res) {
30   const nome = req.query['nome'];
31   console.log(nome);
32   const pessoa = { nome, idade: 42, salario: 123.45, data: new Date() };
33
34   res.setHeader('Content-Type', 'application/json');
35   res.status(201).send(JSON.stringify(pessoa));
36
37   // res.status(201).json(fulano);
38 });
```

# JSON COM JAVASCRIPT

## ■ Nova versão da página

Requisições síncronas e assíncronas

Pedro

Página 1 Página 2 Página 3 Servidor JSON

```
{
  "nome": "Pedro",
  "idade": 42,
  "salario": 123.45,
  "data": "2022-11-11T20:55:40.307Z"
}
```

# JSON COM JAVA

## ■ Nova versão da página

```
1 <!DOCTYPE HTML>
2 <html lang="pt-BR">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta http-equiv="X-UA-Compatible" content="IE=edge">
7   <meta name="viewport" content="width=device-width, initial-scale=1.0">
8
9   <title>Requisições Assíncronas</title>
10
11   <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.2/dist/css/bootstrap.min.css" rel="stylesheet"
12     integrity="sha384-Zenh87qX5JnK2J10vNa8Ck2rdkQ2Bz5IDxbcnCeu0xjzrPF/et3URy98v1WTRI" crossorigin="anonymous">
13
14   <script>
15     function requisitarPagina(url) {
16       document.querySelector('#conteudo').innerHTML = '';
17
18       // Carregando o arquivo de imagem
19       if (!document.querySelector('#carregamento')) {
20         let img = document.createElement('img');
21         img.id = 'carregamento';
22         img.src = './loading.gif';
23         img.className = 'rounded mx-auto d-block';
24         document.querySelector('#conteudo').appendChild(img);
25       }
26
27       let ajax = new XMLHttpRequest();
28       console.log('Estado readyState:', ajax.readyState);
29
30       // Evento que é disparado quando o estado da conexão é modificado
31       ajax.onreadystatechange = () => {
32         console.log('Estado readyState:', ajax.readyState);
33
34         if (ajax.readyState == 4) {
35           if (ajax.status == 200) {
36             document.querySelector('#conteudo').innerHTML = ajax.responseText;
37             console.log('Requisição finalizada com sucesso!');
38             console.log('O status da resposta é:', ajax.status);
39           } else if (ajax.status == 201) {
40             console.log(ajax.responseText);
41             let fulano = JSON.parse(ajax.responseText);
42             console.log(fulano);
43             document.querySelector('#conteudo').innerHTML = `<pre>${JSON.stringify(fulano, undefined, 2)}</pre>`;
44           } else if (ajax.status == 404) {
45             document.querySelector('#conteudo').innerHTML = '<h3>Recurso não encontrado!</h3>';
46             document.querySelector('#conteudo').innerHTML += 'O status da resposta é ' + ajax.status;
47             console.log('Recurso não encontrado!');
48           }
49         }
50       }
51     }
52
53     let nome = document.querySelector('#nome').value;
54     url += '?nome=' + nome;
55     console.log(url);
56     ajax.open('GET', url);
57     ajax.send();
58   }
59 </script>
60 </head>
```

# JSON COM JAVASCRIPT

- Nova versão da página

```
62 <body>
63
64 <nav class="navbar navbar-light bg-light mb-4">
65   <div class="container">
66     <div class="navbar-brand mb-0 h1">
67       <h3>Requisições síncronas e assíncronas</h3>
68     </div>
69   </div>
70
71 <div class="container">
72   <div class="row justify-content-center">
73     <input type="text" value="Fulano" id="nome">
74   </div>
75   <hr>
76   <div class="row mb-2 justify-content-center">
77     <div class="col"></div>
78     <div class="col-auto">
79       <a href="#" class="btn btn-primary"
80         onclick="requisitarPagina('pagina_1_conteudo.html1')">
81         Página 1
82       </a>
83
84       <a href="#" class="btn btn-primary"
85         onclick="requisitarPagina('pagina_2_conteudo.html1')">
86         Página 2
87       </a>
88
89       <a href="#" class="btn btn-primary"
90         onclick="requisitarPagina('pagina_3_conteudo.html1')">
91         Página 3
92       </a>
93
94       <a href="#" class="btn btn-primary"
95         onclick="requisitarPagina('http://127.0.0.1:8080/processa-requisicao')">
96         Servidor
97       </a>
98
99       <a href="#" class="btn btn-primary"
100        onclick="requisitarPagina('http://127.0.0.1:8080/devolve-json')">
101        JSON
102      </a>
103    </div>
104    <div class="col"></div>
105  </div>
106
107 <div class="row">
108   <div class="col-md-1"></div>
109   <div class="col-md-10" id="conteudo"></div>
110   <div class="col-md-1"></div>
111 </div>
112 </div>
113 </body>
114
115 </html>
```

# BSON

## ■ BSON: *Binary* JSON

- Formato de dados utilizado para armazenamento e transferência de dados
- JSON em forma binária
- Projetado para ser eficiente em termos de espaço de armazenamento e velocidade de processamento

## ■ Tipos de dados

- Unicode usando codificação UTF-8
- Inteiros de 32 e 64 bits
- Ponto flutuante no formato IEEE 754
- Datas (datetime, timestamp)
- MinKey, MaxKey
- Arrays
- BinData (*array de bytes binários*)
- ObjectId (identificador único)
- Booleano
- Código JavaScript
- Dados binários MD5
- Expressões regulares no formato PCRE