

P.PORTO

# Javascript e HTML DOM

Programação em Ambiente Web

# Índice

HTML DOM

Javascript Objects e Web APIs

AJAX

JSON

# HTML DOM

O *Document Object Model (DOM)* é um padrão do W3C (World Wide internet Consortium)

O DOM define um padrão para aceder a documentos:

- "*The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document.*"

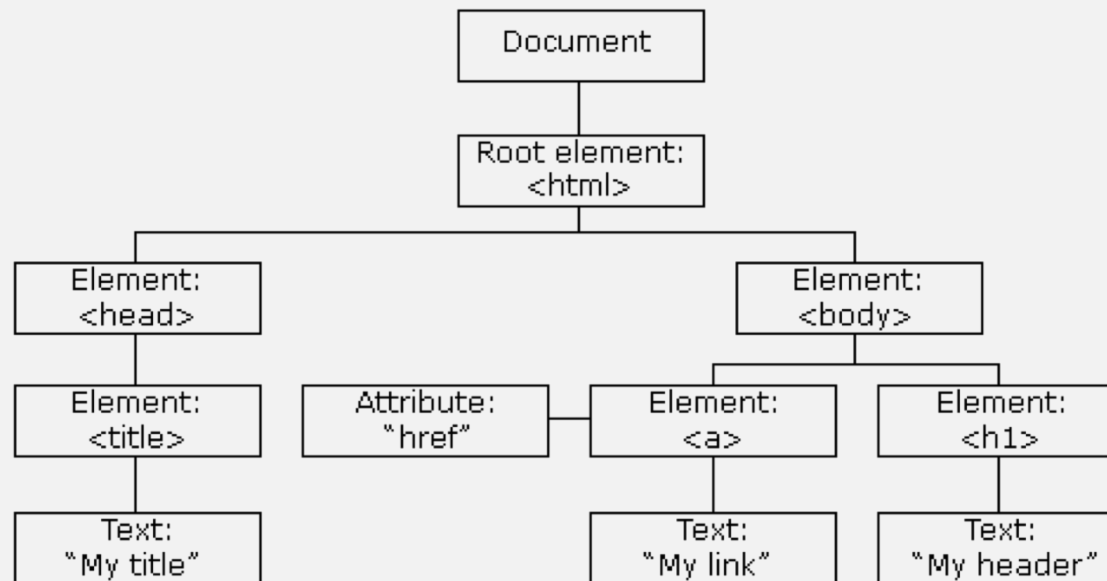
# HTML DOM

O padrão DOM do W3C é separado em 3 partes diferentes:

- Core DOM - modelo padrão para todos os tipos de documentos
- XML DOM - modelo padrão para documentos XML
- **HTML DOM** - modelo padrão para documentos HTML

# HTML DOM

## HTML DOM (Document Object Model)



# HTML DOM

O HTML DOM é um modelo de objetos para HTML que define:

- Elementos HTML como **objetos**
- **Propriedades** para todos os elementos HTML
- **Métodos** para todos os elementos HTML
- **Eventos** para todos os elementos HTML

O HTML DOM é um padrão para obter, alterar, adicionar ou eliminar elementos HTML.

# HTML DOM

O HTML DOM pode ser considerado uma API (Interface de Programação) JavaScript para:

- mudar todos os elementos HTML na página
- alterar todos os atributos HTML na página
- mudar todos os estilos CSS na página
- remover elementos e atributos HTML existentes
- adicionar novos elementos e atributos HTML
- reagir a todos os eventos HTML existentes na página
- criar novos eventos HTML na página

# HTML DOM

## Exemplo do uso da API HTML DOM

- O exemplo a seguir altera o conteúdo (o innerHTML) do elemento `<p>` com `id = "demo"`:

```
<html>
<body>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = "Hello World!";
</script>

</body>
</html>
```



# HTML DOM

## A propriedade innerHTML:

- É a maneira mais fácil de obter o conteúdo de um elemento
- É útil para obter ou substituir o conteúdo um elemento HTML.

# HTML DOM

Para cada página da internet existe, no browser, o objeto **document** previamente

O objeto **document** representa todos os elementos HTML da página

Para aceder a qualquer elemento de uma página em HTML através de JavaScript devemos aceder ao objeto **document** e a partir dele manipular o conteúdo da página

# HTML DOM

## Propriedades do objeto document

Propriedade	Descrição	DOM
document.anchors	Devolve todos os elementos <a> elements that have a name attribute	1
document.baseURI	Devolve o URI base do documento	3
document.body	Devolve o elemento <body>	1
document.cookie	Devolve os cookies do document	1
document.doctype	Devolve o doctype do document	3
document.documentElement	Devolve o elemento <html>	3
document.documentMode	Devolve o mode usado pelo browser	3
document.documentURI	Devolve o URI do document	3
document.domain	Devolve o dominio do servidor	1
document.embeds	Devolve todos os elementos <embed>	3
document.forms	Devolve todos os elementos <form>	1

# HTML DOM

## Propriedades do objeto document

Propriedade	Descrição	DOM
document.images	Devolve todos os elementos <img>	1
document.implementation	Retorna a versão do DOM	3
document.inputEncoding	Devolve encoding do document (character set)	3
document.lastModified	Devolve a data e hora da última alteração do document	3
document.links	Devolve todos os elementos <area> e <a> que tem um atributo href	1
document.readyState	Devolve o estado de loading do document	3
document.referrer	Devolve a URI do link que encaminhou para a página (linking document)	1
document.scripts	Devolve todos os elementos <script>	3
document.strictErrorChecking	Indica se a verificação de erros está activa	3
document.title	Devolve todos os elementos <title>	1
document.URL	Devolve a URL complete do document	1

# HTML DOM

## Exemplo do uso do objeto document

- *document.write* permite adicionar conteúdo à página de internet dinamicamente

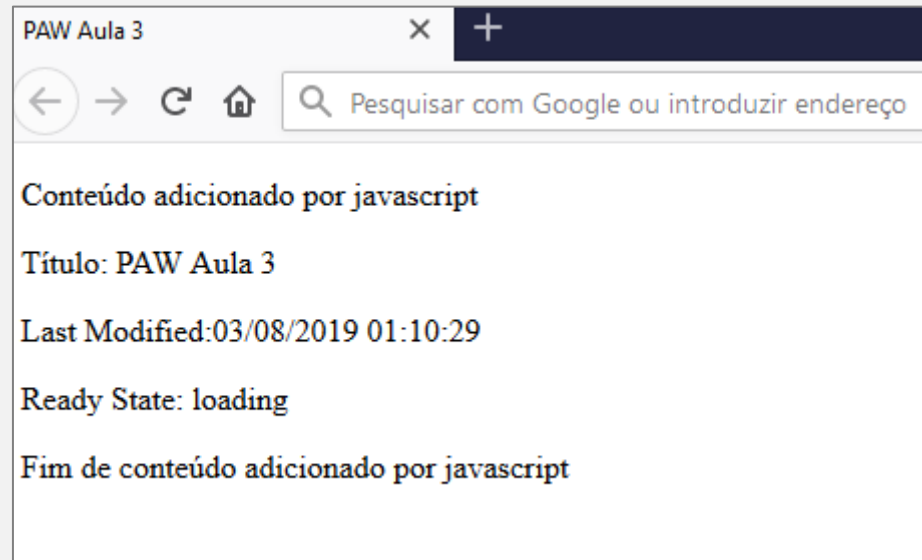
```
<!DOCTYPE html>
<html>

<head>
  <!-- Required meta tags -->
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <title>PAW Aula 3</title>
</head>

<body>
  <script>
    document.write('<p> Conteúdo adicionado por javascript </p>');
    document.write('<p> Título: ');
    document.writeln(document.title);
    document.write('</p>');
    document.write('<p> Last Modified:');
    document.writeln(document.lastModified);
    document.write('</p>');
    document.write('<p> Ready State: ');
    document.writeln(document.readyState);
    document.write('</p>');
    document.write('<p> Fim de conteúdo adicionado por javascript </p>');
  </script>
</body>
</html>
```

# HTML DOM

## Exemplo do uso do objeto document



# HTML DOM

Para aceder a elementos com a representação de objetos HTML em JavaScript, é necessário encontrar os elementos HTML

Podemos seleccionar:

- elementos HTML por id
- elementos HTML pelo nome da tag
- elementos HTML por nome de classe
- elementos HTML por seletores CSS
- elementos HTML por coleções de objetos HTML

# HTML DOM

Encontrar elementos HTML a partir do objeto document

Método	Descrição
<code>document.getElementById(<i>id</i>)</code>	Encontrar um elemento por id
<code>document.getElementsByTagName(<i>name</i>)</code>	Encontrar um elemento por tag
<code>document.getElementsByClassName(<i>name</i>)</code>	Encontrar um elemento por classe



# HTML DOM

## Exemplo HTML DOM

- Alterar texto de um elemento HTML

```
<html>
<body>

<p id="p1">Original</p>

<script>
document.getElementById("p1").innerHTML = "Texto Alterado";
</script>

</body>
</html>
```

# HTML DOM

## Mudar elementos em HTML

Propriedade	Descrição
<i>element.innerHTML = new html content</i>	Alterar o html interno (innerHTML) do elemento
<i>element.attribute = new value</i>	Mudar o valor de um atributo de um element html
<i>element.setAttribute(attribute, value)</i>	Mudar o valor de um atributo de um element html
<i>element.style.property = new style</i>	Mudar o estilo (CSS) de um element HTML

# HTML DOM

## Adicionar eventos a elementos HTML

Método	Descrição
<pre>document.getElementById(<i>id</i>).onclick = function(){     <i>code</i> }</pre>	Adicionar a função ao evento onclick do elemento com o id passado como argumento
<pre>document     .getElementById(<i>id</i>)     .addEventListener( evento, function(){ <i>code</i> } )</pre>	Adicionar a função ao evento do elemento com o id passado como argumento
<pre>document.getElementById(<i>id</i>)     .removeEventListener( evento, function(){ <i>code</i> } )</pre>	Remove a função do evento onclick do elemento com o id passado como argumento

# HTML DOM

## Exemplo HTML DOM

```
<html>

<body>
  <p id="p2">Hello World!</p>
  <button type="button" id="myButton"> Mudar Cor </button>

  <script>
    function changeColor() {
      document.getElementById("p2").style.color = "red";

      document.getElementById("myButton").removeEventListener("click", changeColor);
    }

    document.getElementById("myButton").addEventListener("click", changeColor);
    //document.getElementById("myButton").onclick = changeColor;
  </script>
</body>

</html>
```

# HTML DOM

## Exemplo HTML DOM

- Comportamento para reagir ao evento click é adicionado ao carregar a página HTML
- Após o primeiro evento click no botão, o botão deixa de responder a novos eventos

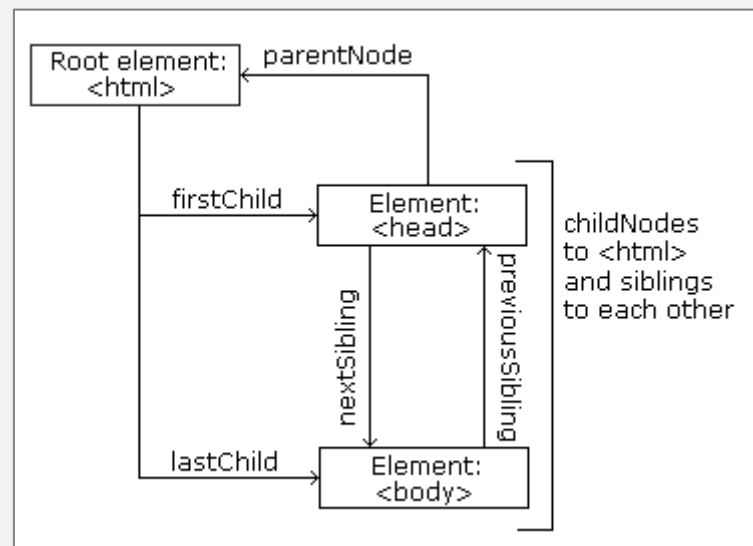
Podemos alterar o exemplo e separar totalmente o código JavaScript num ficheiro “.js” e importar para o ficheiro “.html”

# HTML DOM

## Relação entre nodos HTML

Na figura ao lado podemos ler:

- `<html>` é o nó raiz
- `<html>` não tem pais
- `<html>` é o pai de `<head>` e `<body>`
- `<head>` é o primeiro filho de `<html>`
- `<body>` é o último filho de `<html>`



# HTML DOM

## Relação entre nodos HTML

No html ao lado podemos ler:

- <head> tem um child: <title>
- <title> tem um child (um nó de texto): "Tutorial DOM"
- <body> tem dois child: <h1> e <p>
- <h1> tem um child: "Título 1"
- <p> tem um child: "Paragrafo"
- <h1> e <p> são siblings

```
<html>

  <head>
    <title>PAW Tutorial</title>
  </head>

  <body>
    <h1>Título 1 </h1>
    <p>Paragrafo</p>
  </body>
</html>
```

# HTML DOM

## Adicionar, alterar ou remover elementos

Método	Descrição
<code>document.createElement(<i>element</i>)</code>	Criar um elemento HTML
<code>document.removeChild(<i>element</i>)</code>	Remove um elemento HTML
<code>document.appendChild(<i>element</i>)</code>	Adicionar um elemento HTML
<code>document.replaceChild(<i>new</i>, <i>old</i>)</code>	Substitui um elemento HTML
<code>document.write(<i>text</i>)</code>	Acrecenta HTML na página



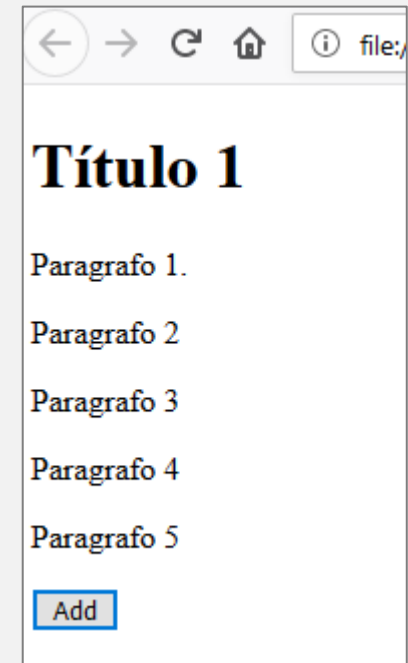
# HTML DOM

```
<!DOCTYPE html>
<html>
  <head>
    <title>PAW Tutorial</title>
    <meta charset="utf-8">
  </head>
  <body>
    <h1>Título 1 </h1>
    <div id="div1">
      <p id="p1">Paragrafo 1.</p>
    </div>
    <button onclick="addParagraph()"> Add</button>

    <script>
      var counter = 1;

      function addParagraph(){
        var para = document.createElement("p");
        var node = document.createTextNode("Paragrafo " + counter);
        para.appendChild(node);

        var element = document.getElementById("div1");
        element.appendChild(para);
        counter++;
      }
    </script>
  </body>
</html>
```



# HTML DOM

Podemos usar JavaScript e HTML DOM para editar vários elementos numa página de uma só vez

Quando fazemos uma procura por tagName ou por classe no objeto document, será devolvida uma coleção de objetos

A coleção tem um aspeto de array mas não o é (ex: não é possível invocar os métodos push ou pop)

# HTML DOM

```
<!DOCTYPE html>
<html>
  <head>
    <title>PAW Tutorial</title>
    <meta charset="utf-8">
  </head>
  <body>
    <h1>Título 1 </h1>
    <div id="div1">
      <p>Paragrafo 1.</p>
      <p>Paragrafo 2.</p>
      <p>Paragrafo 3.</p>
      <p>Paragrafo 4.</p>
      <p>Paragrafo 5.</p>
      <p>Paragrafo 6.</p>
      <p>Paragrafo 7.</p>
      <p>Paragrafo 8.</p>
    </div>
    <script>
      var i;
      var elements = document.getElementsByTagName("p");
      for (i = 0; i < elements.length; i++) {
        if (i % 2 == 0 ){
          elements[i].style.backgroundColor = "red";
        }
      }
    </script>
  </body>
</html>
```

## Título 1

Paragrafo 1.

Paragrafo 2.

Paragrafo 3.

Paragrafo 4.

Paragrafo 5.

Paragrafo 6.

Paragrafo 7.

Paragrafo 8.

# HTML DOM

JavaScript pode também ser usado para a validação de formulários

A validação de formulário HTML pode ser feita com recurso a JavaScript e o método `checkValidity()`

Este método verifica os inputs de um formulário com base nas restrições declaradas em elementos HTML

```
<input id="id1" type="number" min="100" max="300" required>
```

# HTML DOM

O JavaScript pode também ser usado para a validação de formulários

A validação de formulário HTML pode ser feito com recurso a JavaScript, o método `checkValidity()` e o método `setCustomValidity()`

`checkValidity()` os inputs de um formulário com base nas restrições declaradas em elementos HTML

`setCustomValidity()` declara as mensagens de erro a apresentar caso o valor seja inválido

# HTML DOM

## Exemplo form A

```
<form>
  <div>
    <label for="choose">Would you prefer a banana or a cherry?</label>
    <input type="text" id="choose" name="i_like" required minlength="6" maxlength="6">
  </div>
  <div>
    <label for="number">How many would you like?</label>
    <input type="number" id="number" name="amount" value="1" min="1" max="10">
  </div>
  <div>
    <button>Submit</button>
  </div>
</form>
```

# HTML DOM

- Exemplo form B

```
<form>
  <label for='email'>Please provide your email</label>
  <input type='email' id='email' name='email' />
  <button>Submit</button>
</form>
```


```
var email = document.getElementById('email');

email.addEventListener("input", function (event){
  if (email.validity.typeMismatch){
    email.setCustomValidity('I expected a valid email')
  } else {
    email.setCustomValidity('')
  }
})
```

# HTML DOM

- Exemplo form b
  - Resultado após um mau input

Please provide your email

 I expected a valid email



# HTML DOM

- Exemplo C

```
<input id="id1" type="number" min="100" max="300" required>
<button onclick="myFunction()">OK</button>

<p id="demo"></p>

<script>
function myFunction() {
  var inpObj = document.getElementById("id1");
  if (!inpObj.checkValidity()) {
    document.getElementById("demo").innerHTML = inpObj.validationMessage;
  }
}
</script>
```

# Javascript Objects e Web APIs

Existem vários interfaces e Objetos JavaScript que podem ser acedidos a partir scripts executados num browser

- <https://developer.mozilla.org/en-US/docs/Web/API>

Estes objetos estão apenas presentes nos browsers

- não estão presentes em outros ambientes de execução como NodeJS

# Javascript Objects e Web APIs

Uma lista mais completa pode ser encontrada em:

- <https://developer.mozilla.org/en-US/docs/Web/API>

Podemos explorar formas de interação com utilizadores usando estes objetos e as interfaces disponíveis para:

- geolocalização
- notificações
- animações
- Interagir com dispositivos Bluetooth
- ...

# Javascript Objects e Web APIs

Consideremos o objeto window por exemplo

- <https://developer.mozilla.org/en-US/docs/Web/API/Window>

Através do objeto window é possível aceder a:

- Características do ecrã (screen)
- Endereço corrente da pagina no browser (location)
- Histórico (history)
- Timings
- Cookies
- Características do browser utilizador (browser)

# Javascript Objects e Web APIs

Consideremos agora a Storage API:

- [https://developer.mozilla.org/en-US/docs/Web/API/Storage\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Storage_API)

As aplicações da internet podem armazenar dados localmente no browser do utilizador

Antes do standard HTML5, os dados de aplicações precisavam ser armazenados em cookies, incluídos em todos os pedidos ao servidor

A api Storage é mais segura e pode armazenar maiores quantidades de dados localmente, sem afetar o desempenho da página

Ao contrário de cookies, o limite de armazenamento é maior (pelo menos 5MB) e as informações não são automaticamente transferidas para o servidor

# Javascript Objects e Web APIs

Existem dois objetos para armazenar dados no cliente:

- window.localStorage - armazena dados sem data de expiração
- window.sessionStorage - armazena dados para uma sessão (os dados são perdidos quando o browser é fechado)

Antes de usar o armazenamento na internet , verifique o suporte ao browser para localStorage e sessionStorage

```
if (typeof(Storage) !== "undefined") {  
    // Code for localStorage/sessionStorage.  
} else {  
    // Sorry! No Web Storage support..  
}
```

# Javascript Objects e Web APIs

## Exemplos:

- Armazenar informação e consultar

```
// Store
localStorage.setItem("lastname", "Smith");

// Retrieve
document.getElementById("result").innerHTML = localStorage.getItem("lastname");
```

- Remover informação

```
localStorage.removeItem("lastname");
```

# Javascript Objects e Web APIs

O objeto sessionStorage é igual ao objeto localStorage, exceto pelo fato de armazenar os dados para apenas uma sessão. Os dados são eliminados quando o utilizador fecha o navegador específico do browser

O exemplo a seguir conta o número de vezes que um utilizador clicou em um botão, na sessão atual:

```
if (sessionStorage.clickcount) {  
    sessionStorage.clickcount = Number(sessionStorage.clickcount) + 1;  
} else {  
    sessionStorage.clickcount = 1;  
}  
document.getElementById("result").innerHTML = "You have clicked the button " +  
sessionStorage.clickcount + " time(s) in this session.";
```



# AJAX

AJAX (Asynchronous JavaScript And XML)

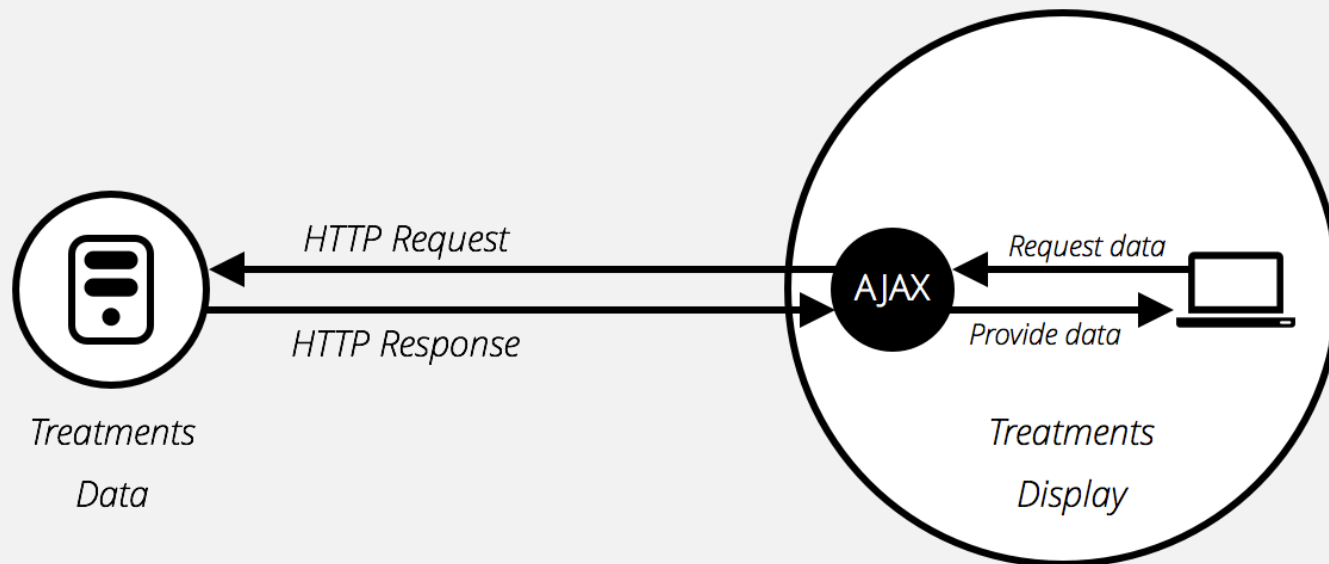
AJAX é uma técnica para aceder a servidores da internet a partir de uma página da internet

AJAX usa apenas:

Um objeto XMLHttpRequest interno do browser (para solicitar dados de um servidor da internet )

HTML DOM e JavaScript (para exibir ou usar os dados)

# AJAX



Fonte: <https://mdn.mozillademos.org/files/6477/moderne-web-site-architecture@2x.png>

# AJAX

O objeto XMLHttpRequest pode ser usado para trocar dados com um servidor da internet

Permite que as páginas da internet sejam atualizadas de forma assíncrona

Permite atualizar partes de uma página da internet , sem recarregar a página inteira

# AJAX

Sintaxe para criar um objeto XMLHttpRequest:

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

Enviar uma solicitação para um servidor

Para enviar uma solicitação para um servidor, usamos os métodos open () e send () do objeto XMLHttpRequest

```
xhttp.open("GET", "file.xml", true);  
xhttp.send();
```

# AJAX

Métodos	Descrição
<code>open(<i>method</i>, <i>url</i>, <i>async</i>)</code>	Especifica o tipo de pedido  <i>method</i> : tipo do pedido GET ou POST <i>url</i> : localização do ficheiro no servidor <i>async</i> : true (assíncrono) ou false (síncrono)
<code>send()</code>	Envia o pedido para o servidor (GET)
<code>send(<i>string</i>)</code>	Envia o pedido para o servidor (POST)

# AJAX

- Pedidos GET ou POST?
  - Um pedido GET é mais simples e rápido que o POST e pode ser usado na maioria dos casos.
  - No entanto, deve-se usar pedidos POST quando:
    - Ficheiros em cache não são opção (atualizar um ficheiro ou base de dados no servidor).
    - Se envia uma grande quantidade de dados para o servidor (o POST não tem limitações de tamanho).
    - Se envia a informação confidencial do utilizador (que pode conter caracteres desconhecidos)
  - O pedido POST é mais robusto e seguro que o GET.

# AJAX

## Exemplo pedido GET

```
xhttp.open("GET", "demo_get.asp?id=" + Math.random(), true);  
xhttp.send();
```

## Exemplo pedido POST

```
xhttp.open("POST", "ajax_test.php", true);  
xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");  
xhttp.send("id=" + Math.random());
```

# AJAX

A propriedade `onreadystatechange` define uma função a ser executada quando o `readyState` é alterado.

```
xhttp.onreadystatechange = function() {  
    if (this.readyState == 4 && this.status == 200) {  
        document.getElementById("demo").innerHTML = this.responseText;  
    }  
};  
xhttp.open("GET", "ajax_info.txt", true);  
xhttp.send();
```

A propriedade `status` e a propriedade `statusText` contêm o estado do objeto `XMLHttpRequest`.



# AJAX

Deve ser criada uma função para executar cada XMLHttpRequest e uma função de callback para cada pedido AJAX.

A função deve conter o URL, a função (callback) a chamar quando a resposta tiver sucesso e, opcionalmente, a função (callback) a chamar se o pedido originar um erro

# AJAX

## Exemplo

```
<!DOCTYPE html>
<html>
<body>

<div id="demo">
<h2>The XMLHttpRequest Object</h2>
<button type="button" onclick="loadDoc()">Change Content</button>
</div>

<script>
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("demo").innerHTML =
        this.responseText;
    }
  };
  xhttp.open("GET", "file.xml", true);
  xhttp.send();
}
</script>

</body>
</html>
```

# AJAX

## Métodos do objeto XMLHttpRequest

Método	Descrição
new XMLHttpRequest()	Cria um novo objeto XMLHttpRequest
abort()	Cancela o pedido
getAllResponseHeaders()	Devolve os headers da resposta ao pedido
getResponseHeader()	Devolve uma parte específica do header de resposta ao pedido
open( <i>method, url, async, user, psw</i> )	Especifica: <ul style="list-style-type: none"><li>• o método de solicitação: o tipo de pedido (ex: GET )</li><li>• url: o local do arquivo</li><li>• assíncrono: verdadeiro (assíncrono) ou falso (síncrono)</li><li>• utilizador: nome de utilizador opcional</li><li>• psw: password opcional</li></ul>
send()	Envia a solicitação para o servidor Usado para solicitações GET
send( <i>string</i> )	Envia a solicitação para o servidor Usado para solicitações GET
setRequestHeader()	Adiciona um par rótulo / valor ao cabeçalho a ser enviado

# AJAX

## Propriedades de um objecto XMLHttpRequest

Propriedade	Descrição
onreadystatechange	Define uma função a ser chamada quando a propriedade readyState é alterada
readyState	Mantém o status do XMLHttpRequest: 0: pedido não inicializado 1: conexão do servidor estabelecida 2: solicitação recebida 3: solicitação de processamento 4: solicitação finalizada e resposta pronta
responseText	Devolve os dados da resposta como uma string
responseXML	Devolve os dados de resposta como dados XML
status	Devolve o número do status de uma solicitação 200: "OK" 403: "Proibido" 404: "Não encontrado" ...
statusText	Devolve o statusText (e.g. "OK" or "Not Found")

# AJAX

## Exemplo

- Carregar postos de carregamento eléctrico a partir da api openchargemap

```
<!DOCTYPE html>
<html>
  <head>
    <title>PAW Tutorial</title>
    <meta charset="utf-8">
  </head>
  <body>
    <table id="myTable">
      <thead>
        <tr>
          <th>Country</th>
          <th>Latitude</th>
          <th>Longitude</th>
          <th>Address</th>
        </tr>
      </thead>
      <tbody id="tbody">
        <tr>
          <td></td>
          <td></td>
          <td></td>
          <td></td>
        </tr>
      </tbody>
    </table>
```

# AJAX

## Exemplo

- Carregar postos de carregamento eléctrico a partir da api openchargemap (continuação)

```
<script>

var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
        var resp = JSON.parse(this.responseText);
        processRequest(resp);
    }
}

xhttp.open("GET", "https://api.openchargemap.io/v3/poi/?output=json&countrycode=US&maxresults=10",
true);
xhttp.send();

function processRequest(resp){
    var i, country, lat, lon, address;
    var tbody = document.getElementById('tbody');

    for (i=0; i<resp.length;i++){
        country = resp[i]["AddressInfo"]["Country"]["ISOCode"];
        lat = resp[i]["AddressInfo"]["Latitude"];
        lon = resp[i]["AddressInfo"]["Longitude"];
        address = resp[i]["AddressInfo"]["AddressLine1"];

        addLine(tbody, country, lat, lon, address);
    }
}
```

# AJAX

## Exemplo

- Carregar postos de carregamento eléctrico a partir da api openchargemap (continuação)

```
function addLine(tableBody, country, lat, lon, address){  
    var row = tableBody.insertRow(0);  
    row.insertCell(0).innerHTML= country;  
    row.insertCell(1).innerHTML= lat;  
    row.insertCell(2).innerHTML= lon;  
    row.insertCell(3).innerHTML= address;  
    //alert(address);  
}  
  
</script>  
</body>  
</html>
```

# AJAX

## Exemplo

Country	Latitude	Longitude	Address
US	46.611568	-112.020013	2701 North Montana Ave
US	33.11674	-96.667297	1220 N Central Expy
US	40.0346915	-75.5796284	635 Lancaster Ave
US	33.780004	-116.469577	68300 Gay Resort DR
US	39.967117	-75.181341	2600 Benjamin Franklin Pkwy
US	33.88401	-84.3027	1 Corsair DR Ste 108
US	37.368797	-79.173357	4640 Murray PL
US	39.49713506177801	-105.3818296711076	12102 S Elk Creek Road
US	40.41328145	-86.8272959141323	4205 Commerce Drive
US	32.5362511	-84.8957642	245 Oakland Parkway



# JSON

## JSON (JavaScript Object Notation)

- O formato JSON é usado para serializar e transmitir dados estruturados
- É usado principalmente para transmitir dados entre um servidor e aplicações da internet.
- Serviços da internet e APIs usam o formato JSON para fornecer dados públicos.
- Pode ser usado com linguagens de programação modernas.

# JSON

```
{
  "id" : 100,
  "message" : "Sales target reached.",
  "recipients" : [
    {
      "name" : "John Robinson",
      "email" : "johnr@bigcompany.com",
      "title" : "VP Sales and Marketing"
    },
    ...,
    {
      "name" : "Margo Haim",
      "email" : "margoh@bigcompany.com",
      "title" : null
    }
  ]
}
```

```
{
  "book": [
    {
      "id": "01",
      "language": "Java",
      "edition": "third",
      "author": "Herbert Schildt"
    },
    {
      "id": "07",
      "language": "C++",
      "edition": "second",
      "author": "E.Balagurusamy"
    }
  ]
}
```

# JSON

Suporta os seguintes tipos de dados:

Type	Description
Number	double- precision floating-point format in JavaScript
String	double-quoted Unicode with backslash escaping
Boolean	true or false
Array	an ordered sequence of values
Value	it can be a string, a number, true or false, null etc
Object	an unordered collection of key:value pairs
Whitespace	can be used between any pair of tokens
null	empty

# JSON

Podemos aceder ao conteúdo de um ficheiro JSON como num objecto de JavaScript após a execução do método `JSON.parse()`

```
<script>
    var data = '{"name": "mkyong","age": 30,"address": {"streetAddress": "88 8nd Street","city": "New York"},

    var json = JSON.parse(data);

    alert(json["name"]); //mkyong
    alert(json.name); //mkyong

    alert(json.address.streetAddress); //88 8nd Street
    alert(json["address"].city); //New York

    alert(json.phoneNumber[0].number); //111 111-1111
    alert(json.phoneNumber[1].type); //fax

    alert(json.phoneNumber.number); //undefined
</script>
```

# Referências

## JavaScript

- <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference>

## DOM

- [https://developer.mozilla.org/en-US/docs/Web/API/Document\\_Object\\_Model](https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model)

## Web APIs

- <https://developer.mozilla.org/en-US/docs/Web/API>
- [https://developer.mozilla.org/en-US/docs/Web/API/Storage\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Storage_API)

## XMLHttpRequest

- <https://developer.mozilla.org/pt-BR/docs/Web/API/XMLHttpRequest>

## JSON

- [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/JSON](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON)

P.PORTO

# Javascript e HTML DOM

Programação em Ambiente internet