

1 O que é Ajax?

A palavra AJAX deriva de *Asynchronous Javascript And Xml*. AJAX é uma técnica que permite que as páginas da Web sejam atualizadas de forma **assíncrona**, trocando pequenas quantidades de dados entre o cliente e o servidor sem recarregar a página web. Basicamente, o Ajax usa o objeto **XMLHttpRequest (XHR)** do *browser* para enviar e receber informações de e para um servidor web de forma assíncrona, em segundo plano, sem bloquear a página ou interferir na experiência do utilizador.

Ajax tornou-se tão popular que é difícil encontrar uma aplicação que não faça uso deste recurso. O exemplo de algumas aplicações: Gmail, Google Maps, Google Docs, YouTube, Facebook, Flickr e tantos outros aplicativos.

Nota: o Ajax não é uma tecnologia, é apenas um termo para descrever o processo de troca de dados entre o servidor web e o cliente, de forma assíncrona, através do JavaScript, sem atualizar a totalidade da página.

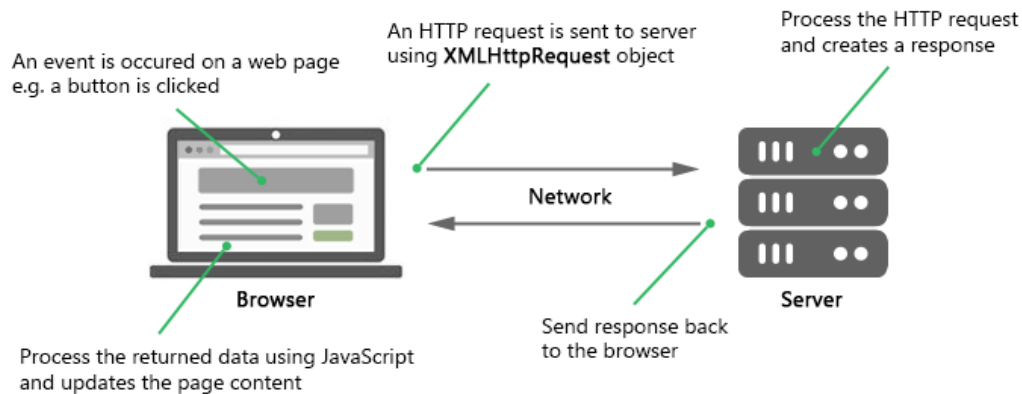
A interação assíncrona implica que o *browser* pode continuar a processar eventos na página após o envio da requisição. Enfatiza-se que, os dados são passados em segundo plano, e podem ser carregados automaticamente na página sem realizar uma atualização da página.

2 Qual é o XMLHttpRequest?

Para executar a comunicação Ajax, o JavaScript usa um objeto especial incorporado no *browser* — o objeto **XMLHttpRequest** — que faz as solicitações (*requests*) HTTP ao servidor e recebe dados como resposta.

Todos os *browsers* modernos (Chrome, Firefox, IE7+, Safari, Opera) suportam o objeto XMLHttpRequest.

A seguinte ilustração demonstra o funcionamento de uma comunicação Ajax:



Resumindo o que foi dito até ao momento:

O objeto **XMLHttpRequest** permite a comunicação com os scripts no lado do servidor. O cliente envia requisições HTTP ou HTTPS diretamente ao servidor web, carrega os dados e devolve uma resposta ao script que solicitou a requisição (cliente/*browser*). O objeto XMLHttpRequest pode enviar informações em diferentes formatos: JSON, XML, HTML entre outros.

Uma vez que as requisições Ajax são geralmente assíncronas, a execução do script no browser continua após o envio da requisição, ou seja, o *browser* não interrompe a execução do script até que a resposta do servidor regresse.

A seguir discutiremos cada etapa envolvida neste processo:

Enviar uma requisição (*request*) e recuperar uma resposta (*response*)

Antes de executar a comunicação Ajax entre cliente e o servidor, o primeiro passo a realizar consiste em instanciar um objeto a partir da classe XMLHttpRequest.

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

O próximo passo consiste no envio da requisição (*request*) ao servidor recorrendo aos métodos **open()** e **send()** do objeto XMLHttpRequest. O método open() especifica o tipo

de requisição, normalmente aceita dois parâmetros: o tipo de *request HTTP* a utilizar, como "GET", "POST", etc., e a URL a enviar no *request*, assim:

```
request.open("GET", "info.txt");
```

ou

```
request.open("POST", "add-user.php");
```

Finalmente, procede-se ao envio da requisição ao servidor utilizando o método **send()** do objeto XMLHttpRequest.

```
request.send();
```

ou

```
request.send(body);
```

Nota: o método send() aceita um parâmetro opcional que permite especificar o corpo do pedido. É usado principalmente nas requisições *HTTP POST*, uma vez que os pedidos *HTTP GET* seguem nos cabeçalhos dos pedidos.

O método GET é, geralmente, usado para enviar uma pequena quantidade de dados ao servidor. Por outro lado, o método POST é usado para enviar grandes quantidades de dados, como os dados de formulário.

A seguir vamos analisar como funcionam os pedidos Ajax.

Passo 1 - Como fazer uma requisição HTTP

Para fazer uma requisição HTTP ao servidor usando JavaScript, crie uma instância da classe XMLHttpRequest que fornece essa funcionalidade. Esta classe foi originalmente introduzida no Internet Explorer como um objeto ActiveX chamado XMLHttpRequest. Os *browsers* Mozilla, Safari e outros também implementaram a classe XMLHttpRequest que suporta os métodos e propriedades do objeto ActiveX original da Microsoft.

O código que realiza esta operação é o seguinte:

```
if (window.XMLHttpRequest) {  
    // code for IE7+, Firefox, Chrome, Opera, Safari  
    pedido=new XMLHttpRequest();  
} else { // code for IE6, IE5  
    pedido=new ActiveXObject("Microsoft.XMLHTTP");  
}
```

De seguida, é preciso decidir o que fazer depois de receber a resposta do servidor ao seu pedido. Nesta etapa, devemos dizer ao objeto da requisição HTTP qual a função JavaScript que irá manipular o processamento da resposta recorrendo à mudança de estado do evento *onreadystatechange*.

```
pedido.onreadystatechange=function() {  
    // processa a resposta do servidor  
}
```

Depois de ter declarado o que vai acontecer assim que receber a resposta, tem que efetuar a requisição. Para tal, é preciso chamar os métodos **open()** e **send()**:

```
pedido.open("GET","obtemCursos.php?q="+valor, true);  
pedido.send();
```

Descrição das funções:

Método	Descrição
<code>open(method, url, async)</code>	Especifica o tipo de <i>request</i> <i>method</i> : GET or POST <i>url</i> : ficheiro requisitado ao servidor <i>async</i> : true (asynchronous) ou false (synchronous)
<code>send()</code>	Envia a requisição (request) ao servidor

Passo 2 – Manipular a resposta do servidor

Como vimos, a função que realiza o tratamento da resposta do servidor é dada por:

```
pedido.onreadystatechange=function() {  
    // processar a resposta do servidor  
}
```

Nesta função devemos verificar o estado da requisição. Se o estado da requisição tomar o valor igual a "4", significa que a resposta do servidor foi recebida por completo e está tudo OK para continuar o processo.

```
pedido.onreadystatechange=function() {  
    if (this.readyState==4) {  
  
    }  
}
```

Reforça-se que a propriedade **readyState** representa um inteiro que especifica o status da requisição HTTP. Além disso, a função atribuída ao manipulador de eventos é chamada sempre que esta propriedade muda de estado.

Valor	Estado	Descrição
0	UNSENT	Um objeto XMLHttpRequest foi criado, mas o método open() ainda não foi chamado (ou seja, a requisição não inicializada).
1	OPENED	O método open() foi chamado (ou seja, conexão com o servidor foi estabelecida).
2	HEADERS_RECEIVED	O método send() foi chamado (ou seja, o servidor recebeu a requisição).
3	LOADING	O servidor está a processar a solicitação.
4	DONE	A requisição foi processada e a resposta está pronta.

O próximo passo consiste em verificar o **código de resposta do servidor HTTP** (todos os possíveis códigos estão listados no site do W3C).

```
pedido.onreadystatechange=function() {  
    if (this.readyState==4 && this.status==200) {  
        document.getElementById("mostra").innerHTML=this.responseText;  
    }  
}
```

O exemplo verifica o sucesso da operação de retorno da requisição HTTP AJAX, verificando se o código de resposta é 200.

Resumo:

Propriedade	Descrição
<i>onreadystatechange</i>	Define a função a ser chamada quando a propriedade <i>readyState</i> muda
<i>readyState</i>	Possui o <i>status</i> do XMLHttpRequest. 0: pedido não inicializada 1: conexão do servidor estabelecida 2: pedido recebido 3: pedido em processamento 4: pedido concluído e a resposta está pronta
<i>status</i>	200: "OK" 403: "Proibido" 404: "Página não encontrada"

Pode obter um resumo dos valores da propriedade status em:
<https://www.tutorialrepublic.com/html-reference/http-status-codes.php>

Exemplo de aplicação no lado do cliente:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Demosnstração JavaScript Ajax GET </title>
<script>
function displayFullName() {
    // Cria um objeto XMLHttpRequest
    var request = new XMLHttpRequest();

    // Inicializa o request object
    request.open("GET", "greet.php?fname=John&lname=Clark");

    // Define o evento ouvinte readystatechange
    request.onreadystatechange = function() {
        // Verifica se o request está completo e teve sucesso
        if(this.readyState === 4 && this.status === 200) {
            // Insere a resposta (response) do servidor no elemento HTML
            document.getElementById("result").innerHTML = this.responseText;
        }
    };

    // Enviar o request ao servidor
    request.send();
}
</script>
</head>
<body>
    <div id="result">
    </div>
    <button type="button" onclick="displayFullName()">Mostra Nomes</button>
</body>
</html>
```

Tratamento da requisição pelo servidor

```
<?php
if(isset($_GET["fname"]) && isset($_GET["lname"])) {
```

```

    $fname = htmlspecialchars($_GET["fname"]);
    $lname = htmlspecialchars($_GET["lname"]);

    // Cria o nome completo pela junção do nome e apelido
    $fullname = $fname . " " . $lname;

    // Visualiza a mensagem
    echo "Olá, $fullname! Bem-vindo ao nosso website.";
} else {
    echo "Bem-vindo ao nosso website.";
}
?>

```

Mais um exemplo, agora envolvendo uma requisição POST HTTP.

Exemplo de aplicação no lado do cliente:

```

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>JavaScript Ajax POST Demo</title>
<script>
function postComment() {
    // Cria o objeto XMLHttpRequest
    var request = new XMLHttpRequest();

    request.open("POST", "/exemplos/confirmation.php");

    request.onreadystatechange = function() {
        if(this.readyState === 4 && this.status === 200) {
            document.getElementById("result").innerHTML = this.responseText;
        }
    };

    // Recupera os dados do formulário
    var myForm = document.getElementById("myForm");

    var formData = new FormData(myForm);

    // Envia a requisição ao servidor

```



```

        request.send(formData);
    }
</script>
</head>
<body>
    <form id="myForm">
        <label>Nome:</label>
        <div><input type="text" name="name"></div>
        <br>
        <label>Comentário:</label>
        <div><textarea name="comment"></textarea></div>
        <p>
            <button type="button" onclick="postComment()">    Comentário
        </button>
        </p>
    </form>
    <div id="result">
        <p>A resposta do servidor será colocada aqui</p>
    </div>
</body>
</html>

```

Tratamento da requisição pelo servidor

```

<?php
if($_SERVER["REQUEST_METHOD"] == "POST") {

    $name = htmlspecialchars(trim($_POST["name"]));
    $comment = htmlspecialchars(trim($_POST["comment"]));

    // Check if form fields values are empty
    if(!empty($name) && !empty($comment)) {
        echo "<p>Olá, <b>$name</b>. O seu comentário foi recebido com
sucesso.<p>";
        echo "<p>Escreveu: <b>$comment</b></p>";
    } else {
        echo "<p>Por favor, preencha os campos do formulário!</p>";
    }
} else {
    echo "<p>Desculpe, algo não correu bem. Tente novamente.</p>";
}

```

```
}  
?>
```

Notas:

Se não usou o objeto FormData para enviar dados de formulário, por exemplo, se enviar os dados do formulário para o servidor numa *query string*, ou seja, `request.send(key1=value1&key2=value2)`, é preciso definir explicitamente o cabeçalho da requisição recorrendo ao método `setRequestHeader()`:

```
request.setRequestHeader("Content-type","application/x-www-form-urlencoded");
```

O método `setRequestHeader()` deve ser chamado depois do método `open()` e antes do método `send()`.

3 Exemplo Passo a Passo

Começamos por criar um ficheiro HTML, considere o nome **get_request.html**

Digite o seguinte código:

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
  <meta charset="UTF-8">  
  <meta http-equiv="X-UA-Compatible" content="IE=edge">  
  <meta name="viewport" content="width=device-width, initial-scale=1.0">  
  <title>Document</title>  
</head>  
<body>  
  <h1>Lista de Utilizadores</h1>  
  <div class="container">  
    <ul class="list">  
    </ul>  
  </div>  
  <button type="submit" id="btn">Pesquisar</button>  
  <script src="rotinas.js"></script>  
</body>  
</html>
```

Vamos criar o ficheiro javascript que irá tratar da requisição, considere o nome `rotinas.js`

```
// Detect browser type
if (window.XMLHttpRequest) {
    // code for IE7+, Firefox, Chrome, Opera, Safari
    request=new XMLHttpRequest();
} else { // code for IE6, IE5
    request=new ActiveXObject("Microsoft.XMLHTTP");
}
request.open("GET", "lista.php");
request.send();
```

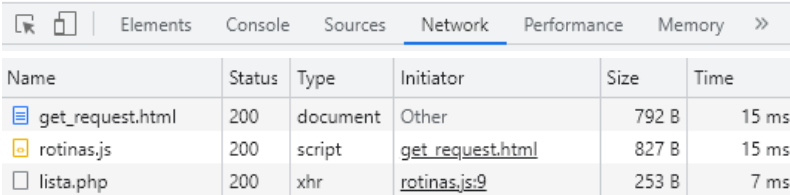
O JavaScript começa por identificar o tipo de browser utilizado, observe que verifica se o utilizador está a utilizar um browser antigo; prepara o tipo de requisição indicando qual o ficheiro que irá tratar essa requisição; e por fim envia a requisição.

Crie o ficheiro que irá tratar a requisição, neste caso chamar-se-á **lista.php**. Neste momento pode ficar vazio.

Carregue o ficheiro **get_request.html** no browser a partir do servidor:

http://localhost/AJAX/1-introducao/get_request.html

Na ferramenta de **Inspecionar** do *browser* (F12), no separador Network, poderá verificar os ficheiros envolvidos.



Name	Status	Type	Initiator	Size	Time
get_request.html	200	document	Other	792 B	15 ms
rotinas.js	200	script	get_request.html	827 B	15 ms
lista.php	200	xhr	rotinas.js:9	253 B	7 ms

Verifique que o ficheiro **lista.php** foi **requisitado** pelo objeto **XMLHttpRequest** (xhr) e que o processo decorreu na normalidade, pois devolveu o **status** 200. Pode acontecer que seja visualizado o status 304, normalmente surge quando se faz um refresh à página. Mas o que significa este código? O HTTP 304, também conhecido como “304 Não Modificado”, é um código que comunica ao browser que: “O recurso solicitado não foi modificado desde a última vez que o acedeu”.

Se mudar o nome do ficheiro lista.php para lista1.php no ficheiro rotinas.js, o browser devolve o status 404 que informa que o ficheiro o ficheiro lista1.php não foi encontrado.

Chegou o momento do script **rotinas.js** detetar se a requisição teve sucesso, com o status 200. Para este efeito existe um evento javascript pertencente ao objeto **XMLHttpRequest** que possui o nome **onreadystatechange**.

Este processo pode ser implementado de duas formas:

```
request.onreadystatechange = function() {  
    if(this.status === 200) {  
        console.log("Olá");  
    }  
};
```

A outra forma:

```
request.addEventListener("readystatechange", function(){  
    if(this.status === 200) {  
        console.log("Olá");  
    }  
});
```

Observe que a palavra “Olá” foi escrita 2 vezes, mais tarde iremos ver melhor o porquê. Tenha em atenção que a palavra reservada this representa o objeto request.

Agora altere o código anterior para:

```
// Detect browser type  
if (window.XMLHttpRequest) {  
    // code for IE7+, Firefox, Chrome, Opera, Safari  
    request=new XMLHttpRequest();  
} else { // code for IE6, IE5  
    request=new ActiveXObject("Microsoft.XMLHTTP");  
}  
request.open("GET", "lista.php");  
request.send();  
  
request.addEventListener("readystatechange", function(){  
    if(this.status === 200) {
```

```

        console.log(request.readyState);
    }
});

```

Execute o ficheiro HTML no browser e analise a informação constante no separador Network da ferramenta inspecionar.

Name	Status	Type	Initiator
get_request.html	304	document	Other
rotinas.js	200	script	get_request.html
lista.php	200	xhr	rotinas.js:9

3 requests	1.4 kB transferred	1.1 kB resources	Finish: 240 ms
------------	--------------------	------------------	----------------

Console	Issues
---------	--------

2	4
---	---

A consola devolveu 2 valores: 2 e 4. Qual o significado destes valores?

Da tabela apresentada anteriormente, chegamos à conclusão de que o código 2 informa que foi estabelecida uma conexão com o servidor, e que o código 4 informa que o servidor processou a requisição (é este o objetivo!!).

Valor	Estado	Descrição
0	UNSENT	Um objeto XMLHttpRequest foi criado, mas o método open() ainda não foi chamado (ou seja, a requisição não inicializada).
1	OPENED	O método open() foi chamado (ou seja, conexão com o servidor foi estabelecida).
2	HEADERS_RECEIVED	O método send() foi chamado (ou seja, o servidor recebeu a requisição).
3	LOADING	O servidor está a processar a solicitação.
4	DONE	A requisição foi processada e a resposta está pronta.

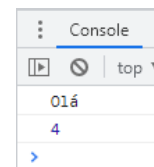
Tenha em atenção que o evento **onreadystatechange** verifica se a propriedade **readyState** foi modificada.

Então ficamos a saber que, se a propriedade status tiver o valor 200 e readyState tiver o valor 4, a requisição foi processada com sucesso. Vamos alterar o código do script rotinas.js

```
request.addEventListener("readystatechange", function(){  
    if(this.readyState === 4 && this.status === 200){  
  
        console.log(request.readyState);  
    }  
});
```

Execute o ficheiro HTML e verifique a saída da consola:

A palavra “Olá” só ocorreu 1 vez.



Acrescente a instrução **console.log(request);** ao evento *readystatechange* dentro da estrutura if. Verifique o resultado na consola do *browser*.

```
XMLHttpRequest {onreadystatechange:  
  ad: XMLHttpRequestUpload, ...} 1  
  onabort: null  
  onerror: null  
  onload: null  
  onloadend: null  
  onloadstart: null  
  onprogress: null  
  onreadystatechange: null  
  ontimeout: null  
  readyState: 4  
  response: ""  
  responseText: ""  
  responseType: ""  
  responseURL: "http://localhost  
  responseXML: null  
  status: 200  
  statusText: "OK"  
  timeout: 0  
  upload: XMLHttpRequestUpload {  
    withCredentials: false  
  }  
  [[Prototype]]: XMLHttpRequest
```

Daqui podem-se observar as diferentes propriedades e eventos que estão associados ao objeto XMLHttpRequest.

Acrescente o seguinte código ao ficheiro lista.php

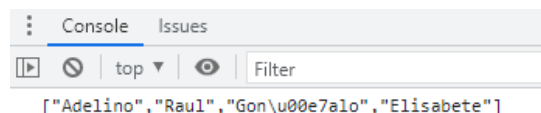
```
<?php
$utilizadores = [
    "Adelino",
    "Raul",
    "Gonçalo",
    "Elisabete"
];
// devolve uma representação JSON do array
echo json_encode($utilizadores);
```

A função **echo** cria a resposta que o servidor devolve ao cliente. A propriedade **responseText**, no lado do cliente, recebe esses valores resultantes da requisição.

Altera o código do ficheiro rotinas.js

```
request.addEventListener("readystatechange", function(){
    if(this.readyState === 4 && this.status === 200){
        // console.log(request);
        console.log(request.responseText);
    }
});
```

Observa o resultado na consola do *browser*.



Vamos definir o formato da resposta, neste caso, no formato JSON

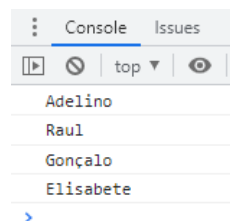
```
...
request.open("GET", "lista.php");
// Define o formato da resposta associada ao request
request.responseType = "json";
request.send();
...
```

No entanto, este tipo de dados só é compatível com os tipos "" e texto.

De seguida, apresentaremos os valores devolvidos:

```
// Detect browser type
if (window.XMLHttpRequest) {
    // code for IE7+, Firefox, Chrome, Opera, Safari
    request=new XMLHttpRequest();
} else { // code for IE6, IE5
    request=new ActiveXObject("Microsoft.XMLHTTP");
}
request.open("GET", "lista.php");
// Define o formato da resposta associada ao request
request.responseType = "json";
request.send();
request.addEventListener("readystatechange", function(){
    if(this.readyState === 4 && this.status === 200){
        // console.log(request);
        // console.log(request.responseText);
        let resposta = request.response;
        for(i=0; i<resposta.length; i++){
            console.log(resposta[i]);
        }
    }
});
```

O output será:



Vamos melhorar a interface e alterar o output:

```
...
let resposta = request.response;
var lista = document.querySelector(".list");
for(i=0; i<resposta.length; i++){
    lista.innerHTML += "<li>" + resposta[i] + "</li>";
}
...
```


Vamos otimizar o código e colocar o botão a trabalhar. Assim, podemos sugerir o seguinte código:

```
const ajax = () => {
    // Detect browser type
    if (window.XMLHttpRequest) {
        // code for IE7+, Firefox, Chrome, Opera, Safari
        request=new XMLHttpRequest();
    } else { // code for IE6, IE5
        request=new ActiveXObject("Microsoft.XMLHTTP");
    }
    request.open("GET", "lista.php");
    // Define o formato da resposta associada ao request
    request.responseType = "json";
    request.send();

    request.addEventListener("readystatechange", function(){
        if(this.readyState === 4 && this.status === 200){
            let resposta = request.response;
            var lista = document.querySelector(".list");
            for(i=0; i<resposta.length; i++){
                lista.innerHTML += "<li>" + resposta[i] + "</li>";
            }
        }
    });
}

let btn = document.getElementById("btn");
btn.addEventListener("click", function(){
    ajax();
})
```

Bibliografia consultada:

https://developer.mozilla.org/pt-BR/docs/Web/Guide/AJAX/Getting_Started