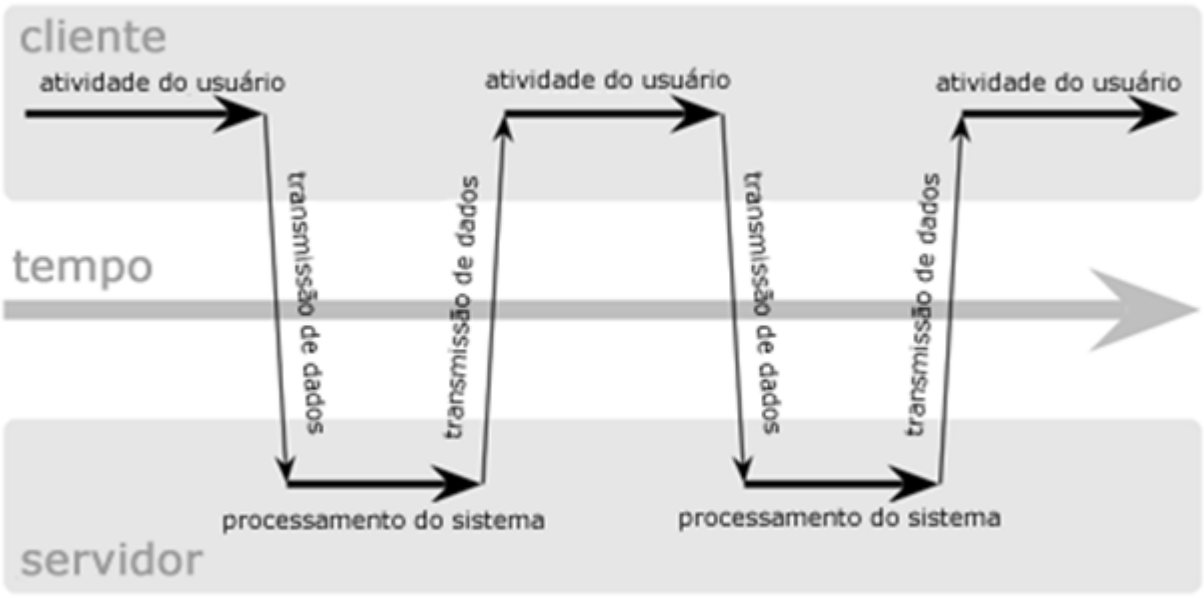
Resumo

A.J.A.X

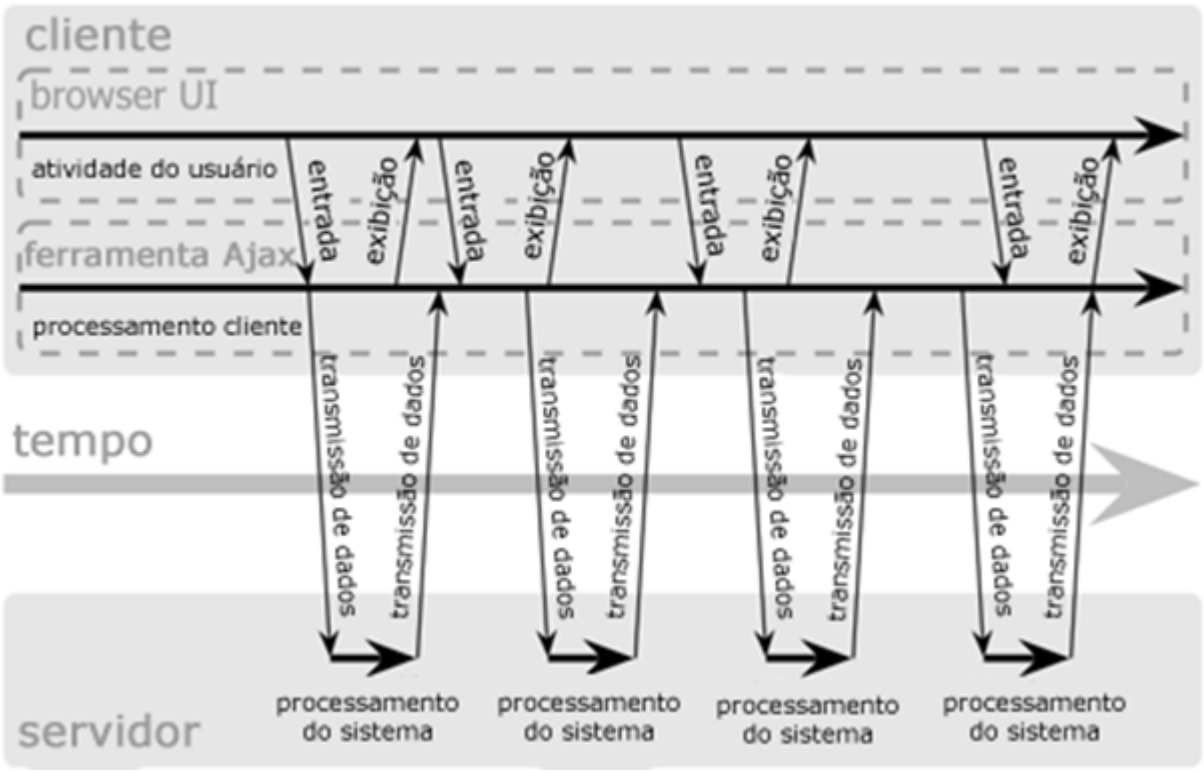
Asynchronous JavaScript XML

AJAX

modelo de aplicação web clássico (síncrono)



modelo de aplicação web Ajax (assíncrono)



História
O objeto XMLHttpRequest

Todos os navegadores modernos suportam o objeto XMLHttpRequest.

O objeto XMLHttpRequest pode ser usado para trocar dados com um servidor da web nos bastidores. Isso significa que é possível atualizar as partes de uma página da web, sem recarregar a página inteira. As principais vantagens de usar AJAX são:

- Atualizar uma página da web sem recarregar a página
- Solicitar dados de um servidor - após o carregamento da página
- Receber dados de um servidor - após o carregamento da página
- Envie dados para um servidor - em segundo plano

O objeto XMLHttpRequest envia os dados para solicitar dados do servidor com uma solicitação HTTP GET ou POST.

Dois métodos comumente usados para uma solicitação-resposta entre um cliente e um servidor são: GET e POST.

GET é mais simples e rápido do que POST e pode ser usado na maioria dos casos.

No entanto, sempre usamos complicações POST quando:

- Um arquivo em cache não é uma opção (atualizar um arquivo ou banco de dados no servidor).
- Enviando uma grande quantidade de dados para o servidor (POST não tem limitações de tamanho).
- Enviando entrada do usuário (que pode conter caracteres desconhecidos), POST é mais robusto e seguro do que GET.

Sintaxe para criar um objeto XMLHttpRequest:

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

Navegadores mais antigos (IE5 e IE6) usam um objeto ActiveX em vez do XMLHttpRequest:
Sintaxe:
variavel = new ActiveXObject("Microsoft.XMLHTTP");

Obs:

Acesso em vários domínios

Por razões de segurança, os navegadores modernos não permitem o acesso entre domínios.

Isso significa que tanto a página da web quanto ao arquivo XML que ela tenta carregar deve estar incluído no mesmo servidor.

O 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.

Method	Description
new XMLHttpRequest()	Creates a new XMLHttpRequest object
abort()	Cancels the current request
getAllResponseHeaders()	Retorna todos os cabeçalhos de resposta como uma string, ou null se nenhuma resposta foi recebida.
getResponseHeader()	Retorna a string contendo o texto do cabeçalho especificado, ou null se quer a resposta ainda não foi recebida ou o cabeçalho não existe na resposta.
open(*method, url, async, user, psw*)	Inicializa um pedido. Este método é para ser usado a partir do código JavaScript; para inicializar um pedido do código nativo *method*: the request type GET or POST *url*: the file location *async*: true (asynchronous) or false (synchronous) *user*: optional user name *psw*: optional password Parameters method O método HTTP para usar, como "GET", "POST", "PUT", "DELETE", etc. ignorado para URLs não-HTTP (S). url O URL para o qual enviar a solicitação. async Um parâmetro booleano opcional, por padrão true , indicando se a operação deve ou não ser executada de forma assíncrona. Se esse valor for false , o send() método não retorna até que a resposta seja recebida. Se true , a notificação de uma transação concluída é fornecida usando ouvintes de evento. Isso deve ser true

	se o multipart atributo for true , ou uma exceção será lançada. user O nome de usuário opcional para usar para fins de autenticação; por padrão, essa é uma seqüência vazia. password A senha opcional para usar para fins de autenticação; por padrão, essa é uma seqüência vazia.
send()	Envia a solicitação. Se o pedido é assíncrono (que é o padrão), este método retorna assim que o pedido for enviado. Se o pedido é síncrono, este método não retorna até a resposta chegar. Used for GET requests
send(*string*)	Envia a solicitação. Se o pedido é assíncrono (que é o padrão), este método retorna assim que o pedido for enviado. Se o pedido é síncrono, este método não retorna até a resposta chegar. Used for POST requests
setRequestHeader()	Define o valor de uma solicitação HTTP header. Você deve chamar setRequestHeader() após open() , mas antes de send().

Property	Description
onreadystatechange	A função de objeto JavaScript que é chamado sempre que o atributo readyState sofre alteração. A função de callback é chamada a partir da thread existente na interface de usuário.
readyState	Holds the status of the XMLHttpRequest. 0 open() não foi chamado ainda. 1 send() não foi chamado ainda. 2 send() foi chamado, e cabeçalhos e status estão disponíveis. 3 Loading, download; responseText contém dados parciais. 4 A operação está concluída.
responseText	A resposta à *request*, em formato texto, retorna null se a solicitação não teve êxito ou que ainda não foi enviada.
responseXML	Returns the response data as XML data
status	Returns the status-number of a request 200: "OK" 403: "Forbidden" 404: "Not Found" For a complete list go to the Http Messages Reference
statusText	A resposta à *request*, em formato texto, retorna null se a solicitação não teve êxito ou que ainda não foi enviada.

response	Retorna um objeto JavaScript de tipo ArrayBuffer, Blob ou Document, de acordo com o que estiver contido no responseType. Retorna null se a request não esteja completa ou não obteve sucesso.
timeout	O número de milissegundos de um pedido pode tomar antes de ser automaticamente encerrada. Um valor de 0 (que é o padrão) significa que não há tempo limite.

Sintaxe Básica

```
function chamandoViaAjax() {
        var xhttp = new XMLHttpRequest();
        xhttp.onreadystatechange = function() {
          if (xhttp.readyState == 4 && xhttp.status == 200) { // verifica o status
           document.getElementById("demo").innerHTML = xhttp.responseText; // executa a ação no html. O xhttp.responseText, pega o texto escrito pelo servlet no response.
          }
        };
        xhttp.open("GET", "chama_o_ servlet", true);  // chama o servlet de forma assincrona
        xhttp.send();
      }
```

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

```
function criandoFuncaoAjax(){
        var xhttp=null;
        if(window.ActiveXObject) // verifica se estamos no inferior ao IE 6.
        {
                xhttp = new ActiveXObject("Microsoft.XMLHTTP");
                alert("Internet Explorer");// so para teste
        }       else if(window.XMLHttpRequest)
                {
                  xhttp = new XMLHttpRequest();
                  alert("Firefox"); // so para teste
                }else   {
                alert("Navegador não suporta Ajax");
                }

        if(xhttp!=null){

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

         }


}
```

Pedido Síncrono

Para executar uma solicitação síncrona, altere o terceiro parâmetro no open() método para false:

```
xhttp.open("GET", "ajax_info.txt", false);
```

Pedido Assíncrono

As solicitações do servidor devem ser sentidas de forma assíncrona.

O parâmetro assíncrono do método open() deve ser definido como verdadeiro:

```
xhttp.open("GET", "ajax_test.asp", true);
```

Ao enviar de forma assíncrona, o JavaScript não precisa esperar pela resposta do servidor, mas pode:

- execute outros scripts enquanto espera pela resposta do servidor
- lidar com a resposta depois que ela estiver pronta

Função AJAX

O **JavaScript** é uma linguagem de programação bem conhecida. Entre suas funcionalidades, está a capacidade de gerenciar conteúdos dinâmicos de um site e permitir a interação dinâmica com o usuário. O XML (*eXtensible Markup Language*), como o nome sugere, é uma variação de linguagem de marcação no estilo do HTML. Enquanto o HTML é utilizado para exibir dados, o XML os armazena e transmite. O AJAX permite a troca de informações simultânea sem interferir com outras funções. O Ajax esta na biblioteca Jquery, logo para usa-la basta acrescentar o JQuery em sua página.

Dicas:

AJAX significa JavaScript assíncrono e XML. Se você ver outro termo **XHR**, que é uma abreviatura para a solicitação XML HTTP, é a mesma coisa.

Descrição de algumas funções:

- $.get() é uma maneira mais geral de fazer pedidos GET. Ele lida com a resposta de muitos formatos, incluindo xml, html, texto, script, json e jonsp.
- $.post() é uma maneira mais geral de fazer pedidos POST. Ele faz exatamente o mesmo trabalho como $.get (), exceto pelo fato de que ele faz uma solicitação POST em vez disso
- $.ajax()**:** Faz a mesma coisa que as funções anteriores, mas com maior controle e detalhes em especificações.

$.ajax({definição no padrão objeto})

Sintaxe Base

```
$.ajax({
        url: 'a url',                 // define o caminho, url, servlet, jsp e etc
        method: 'POST',         // define o método de envio – post, get, put

});
```

Outros parâmetros de ajax().

```
$.ajax({
        url: 'a url',               // define o caminho, url, servlet, jsp e etc
        method: 'POST',         // define o método de envio – post, get, put e etc
        data: { nome: "Pedro", email: "pedro@email.com" }, // chave : valor – define os parâmetros a serem enviados pelo ajax
        async:   true,              // define se o ajax funcionará de forma assíncrona ou síncrona
        contentType: 'application/json; charset=utf-8',   // define codificação dos dados enviado
        dataType: 'json',         // define o tipo do dado recebido, 'json' , 'html',
        data : JSON.stringify(objeto), // alternativa de converter objeto em data
        timeout: 3000,   // define o tempo  limite para o ajax, em milésimo de segundos. 3 segundos.
        success: function(retorno){}, // define a ação quando o ajax é bem sucedido
        error: function(){},     // define a ação quando a erro ao executar o ajax
        beforeSend : function(){}, // define a ação antes de chamar o ajax
        complete: function(){} ,   // define a ação quando o ajax é concluído

});
```

No ajax, nós passamos um objeto. Todo objeto é uma lista de parâmetros separados por virgulas. Cada parâmetro é definido por uma **chave : valor** , ou **chave : função**. Em objeto usa-se dois pontos e não o sinal de recebe.

Descrição

- **Error:** Evento que se produz quando a solicitação produz um erro, basicamente por tentar acessar uma URL inexistente ou porque o servidor não responda no tempo esperado.
- **Success:** Evento que se produz quando a solicitação ajax teve sucesso.
- **Complete:** Evento que se lança quando a solicitação se completou, independentemente de ter sido com sucesso ou com falha. A ordem em que se produzem os eventos faz com que "complete" se execute por último. Ou seja, primeiro se produzirá um "error" ou "success" (dependendo se a solicitação Ajax pode ou não obter uma resposta positiva) e depois se produzirá "complete" independentemente do resultado.
- **JSON.stringify**: serve simplesmente para converter o objeto JSON para um formato que seja possível trasmitir via uma requisição HTTP.
- **contentType: false**: o header 'contentType' não será colocado na requisição HTTP. É sempre bom especificar o tipo de dado sendo enviado - no seu caso 'application/json'.
- **method**: O método HTTP a ser usado para o pedido (por exemplo, POST, GET, PUT).
- **sync**: basicamente, executando de modo assíncrono, o resto do código não irá esperar a resposta do servidor para prosseguir. Se **async: false**, o resto do código só será executado após a resposta do servidor.
- **url**: Endereço no servidor que receberá a requisição.
- **username**: Um nome de usuário a ser usado com XMLHttpRequest em resposta a uma solicitação de autenticação de acesso HTTP.
- **password**: Uma senha para ser usada com XMLHttpRequest em resposta a uma solicitação de autenticação de acesso HTTP.

Exemplo 01

Crie uma pagina em HTML, esta pagina pode ser o resultado final de um JSP, que contenha duas inputs, nome e cpf.
Criaremos uma AJAX simples para acessar um servlet, que irá retornar o nome concatenado com o cpf.

HTML

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>AJAX - Exemplo</title>
<!-- <script     src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"
type="text/javascript"></script>   -->
<script type="text/javascript" src="/jquery/jquery-3.5.1.min.js"></script>
</head>
<body>
<label for="icpf">CPF:</label>
<input id="icpf" placeholder="Digite seu CPF" name="cpf" type="text" /><br>

<label for="inome">Nome:</label>
<input id="inome" placeholder="Digite seu Nome" name="nome" type="text" /><br>

<p id="ip1">Paragrafo:</p>
<p id="ip2">.....</p>
<script>

$(document).ready(function(){

  $("#inome"). change(function() {
      $.ajax({
          url : 'SvEx01',  // define o caminho, url, jsp ou servlet a ser chamado
          method: 'GET',    // define o metodo de envio: post, get, put
          data: { nome: $('inome').val(), cpf: $('icpf').val() },
           // chave : valor, chave : valor, ( parametros do servlet)
          success: function( x ){      $("#ip2").text("ajax bem sucessido " + x);          },
          error: function(){           $("#ip2").text("ajax erro"); },

      });
  });
});
</script>
</body>
</html>
```

Servlet
Crie um servlet

```
package servlet_ajax;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/SvEx01")
public class SvEx01 extends HttpServlet {
	private static final long serialVersionUID = 1L;



   protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

			String text = "Meu primeiro teste do Ajax chamando um servlet ";

			String nome = request.getParameter("nome");
			String cpf = request.getParameter("cpf");
			response.setContentType("text/html;charset=UTF-8");
			response.getWriter().write(text +":" + nome + ", " + cpf);
			// Write no corpo do response.
	}


	protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
		// TODO Auto-generated method stub
		doGet(request, response);
	}


}
```

Descrição dos métodos AJAX(), GET() e POST().

$.ajax() Serve para fazer requisições assíncronas com qualquer método HTTP, entre eles o GET, POST...

$.get() É igual ao anterior porém só serve para requisições do tipo GET.

Sintaxe:

$.get(*URL, callback*);

O parâmetro de *URL* aponta para uma URL predeterminada que você deseja solicitar.

O parâmetro *call-back* opcional é o nome da função executada após o pedido for bem-sucedida.

$ .post () Para solicitar dados do servidor através de solicitações HTTP POST.

sintaxe:

$.post(*URL,data,callback*);

O parâmetro de *URL* aponta para uma URL predeterminada que você deseja solicitar.

O parâmetro *de dados* opcional especifica os dados enviados junto com a solicitação.

O parâmetro *call-back* opcional é o nome da função executada após o pedido for bem sucedida.

CALLBACK

Na programação assíncrona, callbacks são passadas como funções para serem executadas após um certo evento.
Uma função callback é uma função passada a outra função como argumento, que é então invocado dentro da função externa para completar algum tipo de rotina ou ação.
Callback ou "chamada de retorno", em geral é passado uma função em um parametro de uma função, para que quando essa função terminar a execução ela fazer a chamanda deste metodo.
Exemplo você vai fazer uma requisição a um servidor via HTTP, essa chamada é assíncrona (execuções assíncrona são muito comuns eles são executados sem interferir o fluxo normal de seu código). Essa chamanda no servidor pode demorar alguns segundos, porem ao chamar o restante do seu código continua sendo executado. Então como fazer para que quando os ervidor responder você consiga pocessar o seu retorno? ai que que entra os callback.
Ao fazer uma requisição você vai passar o metodo que será executado ao concluir sua requisição. Esse é o metodo de callback.
Dessa forma quando a chamada for retornada o metodo passado será executado.

JSON

JSON: **J** ava **S** cript **O** bject **N** otation.
JSON é uma sintaxe para armazenamento e troca de dados.
JSON é um texto escrito com notação de objeto JavaScript.
A sintaxe JSON é derivada da sintaxe de notação de objeto JavaScript, mas o formato JSON é apenas texto. O código para ler e gerar dados JSON pode ser escrito em qualquer linguagem de programação.

Sintaxe:

O formato JSON é quase idêntico aos objetos JavaScript.

Em JSON, as *chaves* devem ser strings, escritas com aspas dupla, seguido de dois pontos e o valor.

```
{ "name":"John" }
```

Em Java Script um objeto tem a seguinte sintaxe

```
var objeto = { name: "João Carlos", idade: 31, cidade: "Rio de Janeiro" };
```

Ao converte o objeto em JSON seriamos:

```
var valorJSON = { “nome”: "João Carlos", “idade”: “31”, “cidade”: "Rio de Janeiro" };
```

Para acessar um JSON, basta usar a sintaxe. VariávelJson.nomeAtributo

```
valorJSON.nome;    // João Carlos
valorJSON.idade;   // 31
valorJSON.cidade; // Rio de Janeiro
```

Métodos

JSON.parse()
Analisar uma seqüência como JSON, opcionalmente transformar o valor produzido e suas propriedades, e retornar o valor.

JSON.stringify()
Retorna uma string JSON correspondente ao valor especificado, opcionalmente, pode incluir apenas determinados propriedades ou substituir valores de propriedade de acordo com a definição feita pelo usuário.

Exemplo

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>AJAX - Exemplo</title>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"
type="text/javascript"></script>
</head>
<body>

<script>
function usarJSONParse(){

        const varJson = '{"result":true, "count":42}';  // crie um JSON
        const obj = JSON.parse(varJson);              // convertendo JSON para Objeto

        console.log(obj.count);                        // acessando os dados do objeto
        // impresso no console: 42                                  // objeto.atributo

        console.log(obj.result);
        // impresso no console: true
        }

function usarJSONStringify(){

        var n = 'uma string';
        var vet = {'casa', 'dia', 'terra', 'um', 'conjunto', 'valores'};

        var njson = JSON.stringify(n);         // convertendo a variavel n em JSON
        var vetjson = JSON.stringify(vet);  // convertendo a variavel vet(array) em JSON


        var textoN = JSON.parse(njon);       // converte njon em variável
        var vetorVet = JSON.parse(vetjon);   // converte vetson em um vetor
}


</script>
</body>
</html>
```

Exemplo

```
<script>
        // convertendo um valor numero em JSON
        let num = 700;
        console.log(num + ", no formato JSON = " + JSON.stringify(num));

        // convertendp um array em JSON
        let vetor = ['bom ', 'dia', 'alunos', 'da', 'step'];
        console.log(vetor + ", no formato JSON = " + JSON.stringify(vetor));

        // convertendo um objeto em JSON
        let objeto = {id:10, nome:'paulo', cidade: 'rio de janeiro'}
        console.log(objeto + ", no formato JSON = " + JSON.stringify(objeto));



</script>
```

Estrutura geral
tipo/subtipo

A estrutura de um *MIME type* é muito simples; Consiste de um tipo e um subtipo, duas strings, separados por um '/'. Nenhum espaço é permitido. O tipo representa a categoria e pode ser um tipo *discreto* ou *multipart*. O subtipo é específico para cada tipo.

Um *MIME type* é case-insensitive mas tradicionalmente é escrito tudo em minúsculas.

DataType: Define o tipo de dados esperado pelo servidor. Os tipos podem ‘text’, ‘xml’, ‘html’, ‘script’, ‘json’.

- dataType: "xml" - Retorna um documento xml que pode ser processado via JQuery. o retorno deve ser tratado como nodes XML (text/xml ou application/xml)
- dataType: "json" -Avalia a resposta como JSON e retorna um *objeto javascript*
- dataType: "script" -Carrega um script externo em formato de string
- dataType: "html" – (text/html)

Tipo	Descrição	Exemplos de subtipos típicos
text	Representa qualquer documento que contenha texto e é teoricamente legivel para o ser humano.	text/plain, text/html, text/css, text/javascript
image	Representa qualquer tipo de imagens. Os vídeos não estão incluídos, embora imagens animadas (como gif animado) sejam descritas com um tipo de imagem.	image/gif, image/png, image/jpeg, image/bmp, image/webp
audio	Representa qualquer tipo de arquivo de audio	audio/midi, audio/mpeg, audio/webm, audio/ogg, audio/wav
video	Representa qualquer tipo de arquivo de video	video/webm, video/ogg
application	Representa qualquer tipo de dados binários.	application/octet-stream, application/pkcs12, application/vnd.mspowerpoint, application/xhtml+xml, application/xml, application/pdf

Troca de dados

Ao trocar dados entre um navegador e um servidor, os dados podem ser apenas texto.

JSON é texto e podemos converter qualquer objeto JavaScript em JSON e enviar JSON ao servidor.

Também podemos converter qualquer JSON pedido de servidor em objetos JavaScript.

Dessa forma, podemos trabalhar com os dados como objetos JavaScript, sem análises e traduções complicadas.

```
var myObj = {name: "John", age: 31, city: "New York"};
var myJSON = JSON.stringify(myObj);

var myJSON = '{"name":"John", "age":31, "city":"New York"}';
var myObj = JSON.parse(myJSON);
```

Como o formato JSON é apenas texto, ele pode ser facilmente enviado de e para um servidor e usado como formato de dados por qualquer linguagem de programação.

serialize()

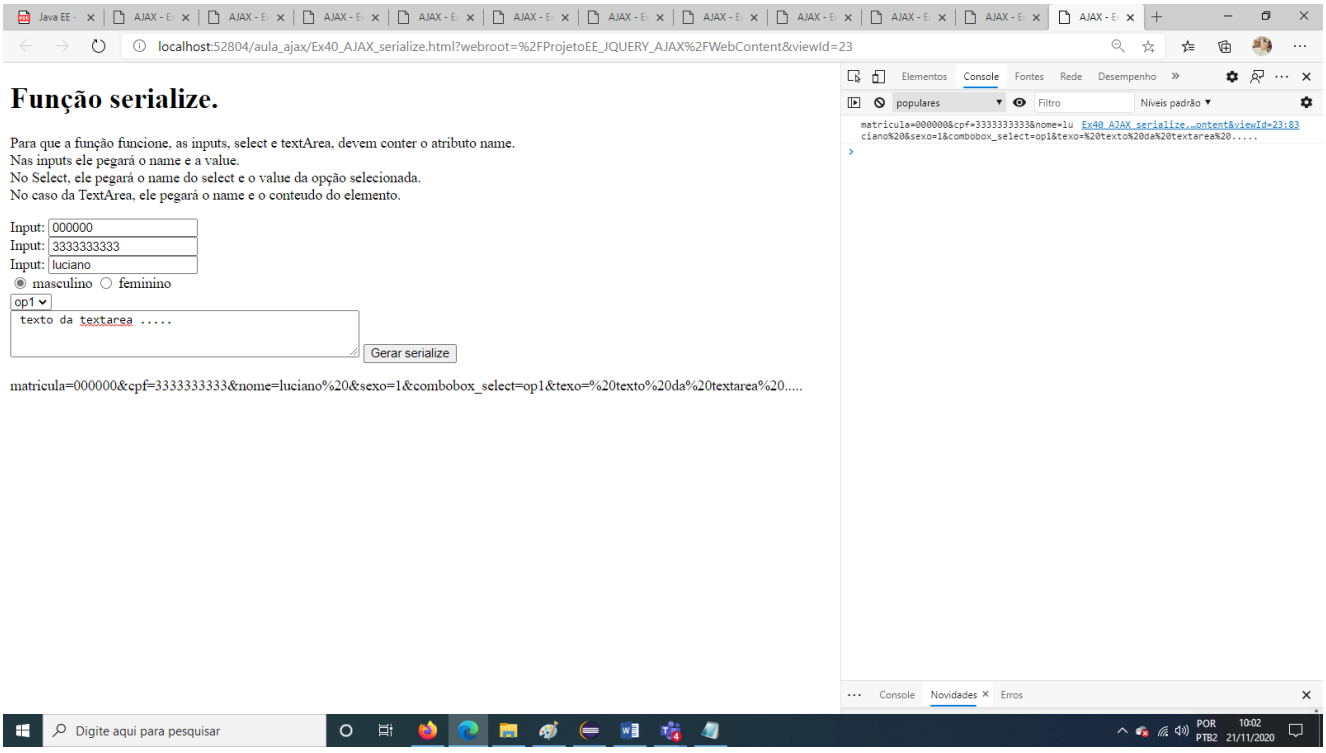
Usando jQuery eu consigo usar o método .serialize() para retornar em forma de string todos os itens do formulário com seus respectivos valores, como, por exemplo, o formulário mais abaixo, irá me retornar:
Os elementos que não contêm um value atributo são representados com o valor da string vazia.

Sintaxe:
var params = $(form).serialize();

Exemplo

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>AJAX - Exemplo</title>
<script type="text/javascript" src="/jquery/jquery-3.5.1.min.js"></script>
</head>
<body>

<h1>Função serialize.</h1>
<p> Para que a função funcione, as inputs, select e textArea, devem conter o atributo name.<br>
Nas inputs ele pegará o name e a value. <br>
No Select, ele pegará o name do select e o value da opção selecionada.<br>
No caso da TextArea, ele pegará o name e o conteudo do elemento.</p>

        <form>

        <label for="imatricula">Input:</label>
        <input id="imatricula" placeholder="Digite sua matricula" name="matricula" type="text" />
<br>

        <label for="icpf">Input:</label>
        <input id="icpf" placeholder="Digite seu CPF" name="cpf" type="text" /><br>

        <label for="inome">Input:</label>
        <input id="inome" placeholder="Digite seu nome" name="nome" type="text" /><br>

        <input type="radio" name="sexo" value="1" checked> masculino
        <input type="radio" name="sexo" value="2"> feminino <br />

        <select name="combobox_select">
                <option value="op1">op1</option>
                <option value="op2">op2</option>
                <option value="op3">op3</option>
                <option value="op4">op4</option>
```

```
        </select><br>

        <textarea rows="3" cols="50" name="texo"> texto da textarea</textarea>

        <input id="ibotao"  type="button" value="Gerar serialize" />
        </form>


<p id="iresultado"> </p>

        <script>

        $(document).ready(function(){

                $("#ibotao").click( function(){
                        let lista = $("form").serialize();
                        console.log(lista);//escrevendo no console

                        $("#iresultado").text(lista); // escrevendo no proprio html
        // será escrito no formato de passagem de argumento, similar ao metodo get.
                 });

        });
</script>
</body>
</html>
```



serializeArray()

Sintaxe:
var params = $(form).serializeArray();

Exemplo

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>AJAX - Exemplo</title>
<script type="text/javascript" src="/jquery/jquery-3.5.1.min.js"></script>
</head>
<body>

<h1>Função serializeArray.</h1>
<p> A função serializeArray() é similar a função serialize(). A diferença é o retorno, um gera um objeto e a outra gera uma array. Como array, podemos acrescentar valores usando o comando push()</p>

	<form>

	<label for="imatricula">Input:</label>
	<input id="imatricula" placeholder="Digite sua matricula" name="matricula" type="text" />
<br>

	<label for="icpf">Input:</label>
	<input id="icpf" placeholder="Digite seu CPF" name="cpf" type="text" /><br>

	<label for="inome">Input:</label>
	<input id="inome" placeholder="Digite seu nome" name="nome" type="text" /><br>

	<input type="radio" name="sexo" value="1" checked> masculino
	<input type="radio" name="sexo" value="2"> feminino <br />

	<select name="combobox_select">
		<option value="op1">op1</option>
		<option value="op2">op2</option>
		<option value="op3">op3</option>
		<option value="op4">op4</option>
	</select><br>

	<textarea rows="3" cols="50" name="texo"> texto da textarea</textarea>

	<input id="ibotao"  type="button" value="Gerar serialize" />
	</form>
```

```
<br><br><br><br><br>
<p>Acrescentando elementos no resultado da função serialize(). Para fazer isto, basta informar o name e o value e incluir esses parametros.</p>
<br>Sintaxe:<br>
nomeLista.push({name: "nome", value: "valor"});


<label >Defina o name:</label>
<input id= "idefinirName" type="text"/><br>
<label >Defina o value:</label>
<input id= "idefinirValue" type="text"/>

<input id="iserialize"  type="button" value="Acrescantar no resultado da serialize" />


<p id="iresultado"> </p>

        <script>

        $("#iserialize").click( function(){
                        let lista = $("form").serializeArray();
                        lista.push( {name: $("idefinirName").val(), value: $("idefinirValue").val()}  );
                        //temos 6 inputs no form e foi acrescentado mais um objeto
                        console.log(lista);//escrevendo no console
        $("#iresultado").text(lista); // escrevendo no proprio html
                });

        });
</script>
</body>
</html>
```