# SUMÁRIO

# 1 LINGUAGEM DE PROGRAMAÇÃO JAVA SCRIPT E JQUERY – PARTE II

## 1.1 RECURSOS AVANÇADOS DA LINGUAGEM JAVA SCRIPT

### 1.1.1 DOM E MANIPULAÇÃO DE ELEMENTOS

DOM é a sigla em inglês para Document Object Model. Este objeto de modelo de documento define uma hierarquia dos elementos html de forma que os mesmos possuam um relacionamento, o qual pode ser manipulado através do Java Script. O relacionamento destes objetos forma uma árvore de relacionamentos conhecida como árvore DOM. Abaixo segue um exemplo de código HTML com a árvore DOM gerada.

```
<!-- My document -->
<html>
	<head>
		<title>My Document</title>
	</head>
	<body>
		<h1>Header</h1>
		<p>
			Paragraph
		</p>
	</body>
</html>
```

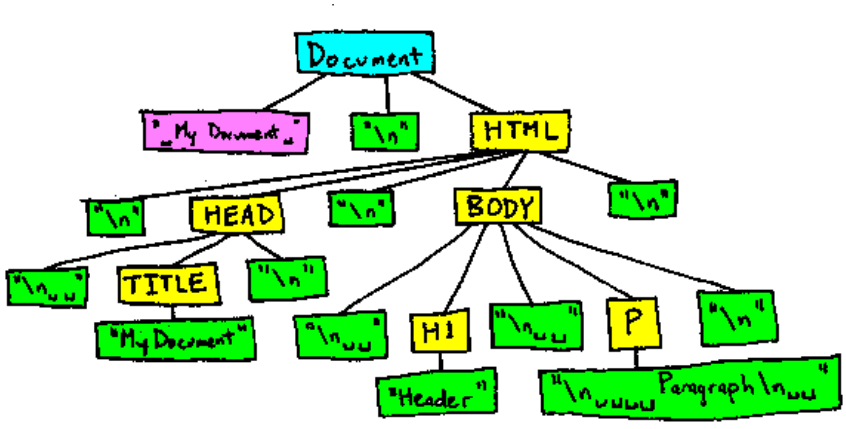


**Figura 1 - Árvore DOM do código HTML acima (obtido do site da Mozilla.org)**

Através do Java Script pode-se manipular o conteúdo desta árvore de forma dinâmica. Este processo de manipulação pode ser realizado de várias formas através do uso de métodos do objeto “document” conforme veremos a seguir:

**document.getElementById("Nome do id"):** Este método retorna um objeto da árvore DOM conforme o id fornecido para o mesmo. O objeto retornado pode ter as suas propriedades alteradas de acordo com a sua necessidade. Exemplo:

```
<html>
	<head>
		<script type="text/javascript" language="javascript">
			function Executar()
			{
				var objeto = document.getElementById("quadrado");

				objeto.style.position = "absolute";
			}
		</script>
	</head>
	<body onLoad="executar()">
		<div id="quadrado"></div>
	</body>
</html>
```

Neste exemplo, estamos obtendo a div de id "*quadrado*" e alterando a propriedade css "*position*" através do JavaScript;

**document.getElementsByTagName("Nome da tag"):** Este método retorna um array de objetos da árvore DOM conforme o nome da tag fornecido para o mesmo. Da mesma forma que o método anterior, pode-se alterar as propriedades de todos os elementos retornados pelo array. Exemplo:

```
<html>
	<head>
		<script type="text/javascript" language="javascript">
			function executar()
			{
				var objeto = document.getElementsByTagName("p");
				var i;

				for (i=0;i<objeto.length;i++)
				{
					alert(objeto[i].innerHTML);
				}
			}
		</script>
	</head>
	<body onLoad="executar()">
		<p>Paragrafo 1...</p>
		<p>Paragrafo 2...</p>
		<p>Paragrafo 3...</p>
	</body>
</html>
```

Neste exemplo estão sendo selecionados todos os parágrafos e através de um laço no vetor retornado é exibido o conteúdo de cada um dos parágrafos.

**document.getElementsByName("Nome do elemento"):** Este método retorna um array de objetos da árvore DOM conforme o nome do elemento. Da mesma forma que o método anterior, pode-se alterar as propriedades de todos os elementos retornados pelo array. Exemplo:

```
<html>
	<head>
		<script type="text/javascript" language="javascript">
			function executar()
			{
				var objeto = document.getElementsByName("par");
				var i;

				for (i=0;i<objeto.length;i++)
				{
					alert(objeto[i].innerHTML);
				}
			}
		</script>
	</head>
	<body onLoad="executar()">
		<p name="par">Paragrafo 1...</p>
		<p name="par">Paragrafo 2...</p>
		<p>Paragrafo 3...</p>
	</body>
</html>
```

**document.getElementsByClassName("Nome da classe"):** Este método retorna um array de objetos da árvore DOM conforme a classe atribuída ao elemento. Da mesma forma que o método anterior, pode-se alterar as propriedades de todos os elementos retornados pelo array. Exemplo:

```
<html>
	<head>
		<script type="text/javascript" language="javascript">
			function executar()
			{
				var objeto = document.getElementsByClassName("p1");
				var i;

				for (i=0;i<objeto.length;i++)
				{
```

Neste exemplo, serão retornados apenas os elementos que possuírem a classe de nome “p1” para o array. O elemento com classe “p2” não será considerado.

**objeto.innerHTML:** Esta propriedade permite aplicar um conteúdo textual e de html a um determinado elemento. Por exemplo, podemos definir o conteúdo de uma DIV dinamicamente através desta propriedade. Deve-se observar que esta propriedade existe apenas para os elementos que permitem a inclusão de conteúdo html. Exemplo:

```
<html>
	<head>
		<script type="text/javascript" language="javascript">
			function executar()
			{
				var objeto = document.getElementsById("elemento");

				objeto.innerHTML = "<b>Um texto dinamico</b>";
			}
		</script>
	</head>
	<body onLoad="executar()">
		<div id="elemento"></div>
	</body>
</html>
```

**objeto.activeElement:** Esta propriedade retorna o elemento da árvore DOM que está selecionado no momento. Exemplo:

```
<html>
	<head>
		<script type="text/javascript" language="javascript">
			function AchaElemento()
			{
				var objeto = document.activeElement;

				alert("O elemento ativo: id = " + objeto.id);
			}
		</script>
	</head>
	<body>
```

**document.hasFocus:** Esta propriedade permite identificar o documento ou qualquer elemento dentro do documento possui foco no momento. Esta propriedade sempre retornará um valor booleano.

Além desta propriedade, pode-se manipular a árvore DOM com outras propriedades as quais permite adicionar, selecionar e remover elementos da árvore de forma dinâmica. Abaixo será apresentado um exemplo de código JavaScript o qual realiza estas operações.

```
<html>
    <head>
        <script type="text/javascript" language="javascript">
               function CarregaDinamicamente()
               {
                  var novoParagrafo = document.createElement("p");

                  var conteudo = document.createTextNode("conteudo do paragrafo");

                  novoParagrafo.appendChild(conteudo);

document.getElementsByTagName("body")[0].appendChild(novoParagrafo);
               }
        </script>
    </head>
    <body onLoad="CarregaDinamicamente()">

    </body>
</html>
```

A propriedade "*createElement*" cria dinamicamente um elemento HTML, enquanto que o método "*createTextNode*" cria dinamicamente um elemento de tipo texto, como por exemplo um texto de parágrafo, um conteúdo de uma célula, etc. Neste caso foi criado um paragrafo (tag p) e um conteúdo para este parágrafo (texto "conteúdo do parágrafo"). O próximo passo será associar o conteúdo do parágrafo com o próprio parágrafo, pois no momento foram apenas criados os elementos. Isso pode ser feito através do método "*appendChild*" aplicado no parágrafo criado. Desta forma o conteúdo estará sendo associado ao mesmo. Ao final de todo este processo o parágrafo criado estará sendo associado a página html através do método "appendChild" aplicado ao corpo da página.

### 1.1.2 FUNÇÕES ANÔNIMAS

O JavaScript além de ser uma linguagem com características de orientação a objetos também é uma linguagem funcional, onde podem ser atribuídas funções para a execução de cada regra do sistema. Para garantir este título a linguagem, existe um recurso conhecido como função anônima.

Uma função anônima permite que a mesma seja criada sem um nome e que a mesma possa ser atribuída a uma variável como se fosse um objeto qualquer. Estas funções podem ser definidas em qualquer local no código Java Script e permitem proteger variáveis contra o mal uso. Exemplo de uma função anônima:

```
<html>
	<head>
		<script type="text/javascript" language="javascript">
			var obj = (function() {

				var a, b, c;
				a = 2;
				b = 3;
				c = a + b;

				return c;
				})();

			function ExecutaFuncao()
			{
				alert(obj);
			}

		</script>
	</head>
	<body onLoad="ExecutaFuncao()">

	</body>
</html>
```

Quando executada esta função, o sistema irá exibir uma mensagem com o valor “5”, pois quando a variável “*obj*” for utilizada, o seu comportamento será a execução da função criada e atribuída a variável.

Um outro exemplo de uso de funções anônimas é o de atribuir uma função dinamicamente a um evento de objetos. Quando criamos um objeto dinamicamente, em diversos casos temos a necessidade de atribuir um evento e ao evento uma função. Para realizar esta ação, podemos fazer da seguinte forma:

```
<html>
	<head>
		<script type="text/javascript" language="javascript">

			function ExecutaFuncao()
			{
				var botao = document.createElement("button");
				botao.style.width="100px";
				botao.style.height="100px";

				botao.onclick = function (){
```

Neste exemplo criou-se um botão dinamicamente e ao mesmo foi atribuída uma função anônima para que quando o botão for clicado, será exibida uma mensagem de alerta.

### 1.1.3 ITERAGINDO COM O TEMPO

No JavaScript é possível programarmos a execução de uma determinada função após um determinado tempo, o qual é especificado em milissegundos (1000 milissegundos = 1 segundo). Esta função pode ser programada para ser executada apenas uma vez ou então, pode ser executada em intervalos de tempo pré-determinados em milissegundos até ser solicitada a sua interrupção.

Para executar o código javascript somente uma vez, deve ser empregada a função "*setTimeout*". Para executar o código diversas vezes deve ser utilizada a função "*setInterval*" para iniciar a sua execução e a função "*clearInterval*" para concluir a execução.

O uso deste tipo de função permite ao programador criar linhas de execução independentes da linha de execução padrão do programa. Ou seja, é possível executar diversas ações ao mesmo tempo. Isso pode ser útil, por exemplo, nos casos onde é preciso exibir a hora atual, sendo incrementada a cada segundo, enquanto é realizada a navegação na página com o uso de outras rotinas também em javascript.

**setTimeOut:**

Sintaxe: *setTimeout(função, tempo);*

*Função:* É a função que será executada apenas uma vez conforme o intervalo de tempo;

*Tempo:* Intervalo de tempo em milissegundos que definine o momento em que a função será executada.

É importante observar que este intervalo de tempo é iniciado no momento em que a rotina é chamada pelo javavascript. Exemplo:

```
<html>
	<head>
		<script>
			function carrega()
			{
				setTimeout(function(){
					var p = document.getElementById("paragrafo");
					p.innerHTML = "Ola Mundo";
				},2000);
			}
		</script>

	</head>
	<body onload="carrega()">
		<p id="paragrafo"></p>
	</body>
</html>
```

Neste exemplo, o sistema irá escrever “Ola Mundo” dois segundos depois da página ser carregada no navegador.

**setInterval e clearInterval:**

Estas duas rotinas funcionam em conjunto. A rotina *setInterval* inicial a execução de uma rotina repetindo a mesma a cada intervalo informado e retornando um valor indicando a linha de execução criada. Para que seja possível parar esta execução é necessário utilizar a função *clearInterval* passando por parâmetro o valor retornado anteriormente pela função *setInterval*.

Sintaxe: *setInterval(função, tempo);*

*Função:* código a ser executado a cada período;

*Tempo:* tempo para repetição da execução;

*clearInterval(valor);*

*Valor:* Valor retornado pela rotina *setInterval*;

```
<html>
	<head>
		<script>
			var linhaExecucao;

			function inicio()
			{
				linhaExecucao = setInterval(function(){
						var p = document.getElementById("paragrafo");
						p.innerHTML += "Ola Mundo<br>";
					},1000);
			}

			function finaliza()
			{
				clearInterval(linhaExecucao);
			}
		</script>
```

Este script vai escrever o termo “Ola Mundo” de forma repetida a cada segundo até que o usuário clicar no botão “Fim”.

### 1.1.4 ROTINAS DE GEOLOCALIZAÇÃO

Com o HTML5 é possível utilizar a API de Geolocalização, a qual permite obter-se informações sobre a localização atual, como latitude, longitude, altura, velocidade de movimentação, entre outras funcionalidades.

Esta API utilizará recursos de GPS ou triangulação, conforme a disponibilidade dos mesmos, sendo que inicialmente sempre será utilizado triangulação, pelo fato de ser mais rápido para o browser obter esta informação, melhorando o desempenho da aplicação. Porém a tendência da localização por triangulação é ser menos precisa que a do GPS.

Para ser possível utilizar a API de geolocalização é necessário aceitar a pergunta do navegador a qual solicita a obtenção das informações de localização. Exemplo:

```
<html>
	<head>
		<script>

			function posiciona()
			{
				navigator.geolocation.getCurrentPosition(showPosition);
			}

			function showPosition(position) {
				var latlon = position.coords.latitude + "," +
					position.coords.longitude;

				var url = "http://maps.google.com/maps?q=" + latlon;

				window.location = url;
			}
		</script>

	</head>
	<body onload="posiciona()">

	</body>
</html>
```

### 1.1.5 SUBSTITUINDO OS COOKIES

Como vimos anteriormente os cookies são uma maneira de salvar informação no navegador, entretanto, esta maneira de armazenar informação possui limitações de quantidade de informação e também apresenta dificuldade em ler e escrever as informações.

Para auxiliar na tarefa de salvar e recuperar informações locais o HTML 5 apresenta um recurso chamado de Storage API.

A API Storage permite duas formas de armazenamento de dados no lado do cliente:

- *sessionStorage;*
- *localStorage.*

O sistema de localStorage e sessionStorage fazem parte da API Storage. Seu tipo de armazenamento é de chave: valor, sendo a chave sempre uma string. O objeto desta API apresenta 4 métodos:

*getItem(chave):* obtém um valor armazenado no Storage;

*setItem(chave, valor):* armazena um valor no Storage;

*removeItem(chave):* exclui um valor do Storage;

*clear():* limpa o storage;

A diferença entre os dois é que o sessionStorage apenas armazena os dados durante a sessão do usuário, caso ele feche o navegador ou a aba, seus dados serão excluídos. Enquanto que o localStorage não possui expiração definida. Exemplo:

```
<html>
	<head>
		<script>
			var linhaExecucao;

			function gravar()
			{
				sessionStorage.setItem("nome","Joao da Silva");
			}

			function ler()
			{
				var retorno;
				retorno = sessionStorage.getItem("nome");
				alert(retorno);
			}
		</script>

	</head>
	<body>
		<input type="button" onclick="gravar()" value="Grava Informacao" > </i>
		<input type="button" onclick="ler()" value="Ler Informacao" > </button>
	</body>
</html>
```

### 1.1.6 CANVAS API

O HTML 5 trouxe consigo uma API gráfica interpretada sem o uso de plugins, a qual é manipulada através da tag *canvas*. Esta API permite desenhar na tela do navegador através do javascript.

Para iniciar um desenho, deve-se incluir no documento html a tag canvas, a qual delimita a área do desenho e no javascript inicializar o canvas através do método *getContext*.

```
<html>
	<head>
		<script>
			function desenhar()
			{
				var context;
				context =
document.getElementById("desenho").getContext("2d");
			}
		</script>

	</head>
	<body>
		<input type="button" onclick="desenhar()" value="Desenhar" >
		<canvas id="desenho" width="400" height="400"></canvas>
	</body>
</html>
```

Com o Canvas é possível desenhar na página e atualizar dinamicamente estes desenhos através de scripts conforme a necessidade do usuário. Esta possibilidade permite inúmeros uso desta API substituindo os atuais plugins como o Flash pelo Canvas. Abaixo vemos um exemplo de desenho construído com o Canvas.

```
<html>
	<head>
		<script>
			function desenhar()
			{
				var context;
				context =
				document.getElementById("desenho").getContext("2d");

				//iniciar o desenho
				context.beginPath();

				//posicionar o cursor para o inicio do desenho
				context.moveTo(150,50);

				//desenho das linhas
				context.lineTo(220,250);
				context.lineTo(50,125);
				context.lineTo(250,125);
				context.lineTo(80,250);
				context.lineTo(150,50);

				//pintar o desenho de amarelo
				context.fillStyle="#ff0";
				context.fill();

				//pintar as bordas de vermelho
				context.strokeStyle="#f00";
				context.stroke();
```

O Canvas também permite movimentar e criar animações nos objetos da tela. Analise o exemplo abaixo:

```
<html>
	<head>
		<script>
			var context;

			function desenhar()
			{
				context =
					document.getElementById("desenho").getContext("2d");

				setInterval(desenhaAnimacao,200);
			}

			function desenhaAnimacao()
			{
				context.clearRect(0,0,960,480);
				context.translate(30,30);
				context.rotate(0.1);
				context.beginPath();
				context.moveTo(10,0);
				context.lineTo(20,30);
				context.lineTo(0,30);
				context.closePath();
				context.stroke();
			}
		</script>

	</head>
	<body onload="desenhar()">
		<canvas id="desenho" width="960" height="480"></canvas>
	</body>
</html>
```

Neste exemplo, estamos criando um desenho customizado. O método beginPath() define ao Canvas que iniciaremos o desenho usando as APIs do PATH, as quais podem ser associadas a um “lápis” na tela.

O moveTo() move nosso “lápis” para o ponto 10,0(x,y). A partir do ponto 10,0, desenhamos uma linha no método lineTo() para o ponto 20,30, Neste momento, é como se fosse desenhada uma linha no mesmo. O mesmo fazemos depois para o ponto 0,30. O método closePath() fecha automaticamente nosso desenho para o ponto inicial, no caso 10,0.

O método stroke() finaliza o desenho. Se este método não for chamado no final, o desenho não aparecerá na tela. Isso, porque o método lineTo() apenas adiciona as coordenadas numa fila, que depois são finalmente desenhadas com o stroke().

O método translate() move o canvas inteiro para a posição x,y passada nos argumentos. Já o rotate(), gira o canvas inteiro para o ângulo passado (em radianos).

Essas duas chamadas dos métodos devem ser colocadas antes de beginPath(). Se forem colocadas depois do stroke(), nada vai acontecer, porque as transformações com translate() e rotate() só afetam os próximos desenhos.

## 1.2 FRAMEWORKS JAVA SCRIPT

### 1.2.1 CONCEITO

Frameworks JavaScript também são conhecidos como bibliotecas de funcionalidades JavaScrpt, as quais reduzem a necessidade de construirmos todas as funcionalidades que precisamos de forma manual, pois a biblioteca já apresenta os recursos prontos. Além de possuir recursos prontos, o framework também simplifica a maneira como programamos em JavaScript, podendo fornecer algumas vantagens conforme listado abaixo:

- Simplificação da forma de utilizar JavaScript;
- Novos recursos e elementos visuais;
- Simplifica a iteração com o usuário e também os recursos do servidor;
- Torna o desenvolvimento em JavaScript independente de browser pois a maioria dos frameworks apresenta as suas funcionalidades de forma independente de browser (crossbrowser);

- Redução de incompatibilidades;
- Reaproveitamento de código;

Para que possa ser utilizado um framework o primeiro passo será estudar o framework para analisar se as funcionalidades fornecidas por ele atendem a necessidade da aplicação que está sendo desenvolvida. Neste ponto deve-se observar que é possível utilizar vários frameworks ao mesmo tempo, pois cada um deles apresenta características específicas de acordo com a funcionalidade proposta por ele.

Após escolher o framework deve-se obter o mesmo e deixa-lo junto a aplicação para poder referencia-lo na página ou então referencia-lo através da própria internet. De acordo com o framework, o mesmo poderá ser composto por um ou vários arquivos do tipo javascript e css.

Um exemplo de framework é o Jquery, o qual pode ser obtido no endereço http://code.jquery.com/jquery-2.1.1.js. Após fazer o download do arquivo "Jquery-2.1.1.js" devemos referenciar o mesmo na página html conforme o exemplo abaixo para ser possível

```
<html>
	<head>
		<script src="Jquery-2.1.1.js"></script>

	</head>

	<body onload="inicializa()">

	</body>
</html>
```

### 1.2.2 IDENTIFICANDO RECURSOS HTML COM O FRAMEWORK MODERNIZR

O Modernizr é uma biblioteca de detecção que permite verificar o suporte da maioria das características do HTML5 e CSS3. O download da biblioteca pode ser obtido no endereço web http://modernizr.com/.

O Modernizr é um script JavaScript que roda automaticamente assim que o mesmo é adicionado no head do documento. No exemplo abaixo a biblioteca está sendo utilizado para verificar se o navegador apresenta suporte a API Canvas do HTML5.

```
<html>
	<head>
		<script src="modernizr.js"></script>
		<script type="text/javascript" language="javascript">
			function verificaCanvas()
			{
				var div = document.getElementById('resultado');

				if (Modernizr.canvas){
					div.innerHTML += "Canvas = ON";
				}
				else{
```

### 1.2.3 FRAMEWORK JQUERY

O Framework Jquery é um biblioteca cross-browser desenvolvida para simplificar a programação de uma página web. Ela foi criada por John Resig em 2006 e atualmente, segundo consulta realizada na Wikipedia ela é empregada em 77% dos 10 mil sites mais visitados no mundo.

Esta biblioteca apresenta uma série de funcionalidades, das quais pode0se citar:

- Resolução de incompatibilidades entre navegadores;
- Redução significativa de código;
- Reutilização de código;
- Altamente expansível e com diversas funcionalidades adicionais:
    - jQuery UI para elementos de interface;
    - jQuery Mobile para desenvolvimento mobile;
- Apresenta grande facilidade de uso dos recursos Ajax e manipulação DOM através dos seletores;
- Permite a implementação de recursos de CSS nas versões 1, 2 e 3.

O jQuery pode ser acessado pelo site http://jquery.com/ e o download pode ser feito no endereço http://jquery.com/download/. Na página de download existem quatro versões de download que podem ser feitas. Existe a versão compactada (compressed) e descompactada (uncompressed). A versão compactada também é conhecida como versão de produção. Esta

versão não apresenta comentários no código e possui redução de espaços e nome de variáveis para que o arquivo seja menor possibilitando melhor desempenho.

A versão descompactada por sua vez apresenta comentários no código o que permite um melhor entendimento do que as rotinas do framework se propõe.

Além disso, também existe a versão 1.x e a versão 2.x. Ambas versões possuem as mesmas funcionalidades, porém a versão 2.x não possui compatibilidade com navegadores antigos (internet explorer 6, 7 e 8).

Abaixo pode-se observar um exemplo de código. O código comentado demonstra como a ação é feito com JavaScript sem o uso do framework jQuery.

```
<html>
	<head>
		<script src="Jquery-2.1.1.js"></script>
		<script>
			function executa()
			{
				//document.getElementById("teste").value = 5;
				$("#teste").val(5);
			}
		</script>
	</head>

	<body onload="executa()">
		<input type="text" id="teste"></input>
	</body>
</html>
```

#### 1.2.3.1 SELETORES JQUERY

O jQuery apresenta uma forma de selecionar os elementos muito dinâmica e simplificada, sendo muito semelhante aos seletores de CSS, mas com recursos mais avançados. Agora serão apresentadas alguns dos seletores que podem ser empregados com jQuery.

**Seletor por Id:** Permite a seleção através do Id de um elemento. Exemplo:

**Html:**
- `<td id=”célula”></td>`

**JavaScript:**
- `document.getElementById(“célula”)`

**Jquery:**
- `$(“#celula”)`

**Seletor por Classe:** Permite a seleção de elementos através da propriedade class dos elementos, retornando um array de elementos. Exemplo:

```
Html:
    ◦ <td class="celula"></td>

JavaScript:
    ◦ document.getElementsByClassName("célula")

Jquery:
    ◦ $(".celula")
```

**Seletor por Elemento:** Permite a seleção de elementos através do nome do elemento, retornando um array de elementos. Exemplo:

```
Html:
    ◦ <td></td>

JavaScript:
    ◦ document.getElementsByTagName("td")

Jquery:
    ◦ $("td")
```

e
s
selecionados.

```
Html:
    ◦ <td></td>

JavaScript:
    ◦ document.getElementsByTagName("td")

Jquery:
    ◦ $("td")
```

Além dos seletores apresentados acima, existem diversos outros seletores que podem ser utilizados. Para saber mais sobre estes seletores, deve-se acessar a documentação do jQuery através do endereço http://api.jquery.com/category/selectors/.

#### 1.2.3.2 EVENTOS JQUERY

O uso do jQuery permite ampliar a quantidade e tipos de eventos que já eram disponibilizados no JavaScript. O primeiro evento a ser considerado com o jQuery é o evento "*ready*". Este evento é muito importante, pois o mesmo será disparado quando a página concluir a criação da árvore DOM contendo os elementos da página, diferentemente do evento "*onLoad*" do JavaScript clássico, o qual é executado quando a página concluí o

carregamento de todos os elementos, inclusive o seu conteúdo, como imagens, tornando a execução das instruções contidas no evento "*ready*" muito mais rápida do que o "*load*". Abaixo segue exemplo de como utilizar o evento ready.

```
<html>
	<head>
		<script>
		$(document).ready(function(){

		   //Aqui devem ser programados os demais eventos . . .

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

Internamente ao evento "*ready*" podem ser adicionados demais eventos e códigos em JavaScript. Estes eventos podem ser adicionados neste local também com o emprego de funções anônimas vistas anteriormente. Agora segue exemplo de como programar no evento "*ready*".

```
<html>
	<head>
		<script>
		$(document).ready(function(){

			//Aqui foi escrita uma mensagem no corpo da página
			$("body").text("O documento esta carregado");

			$("#minhacamada").click(function(e){
				alert("evento de clique executado no elemento de id
				'minhacamada' com função anonima...");
			});
		})
		</script>
	<head>
	<body>
	</body>
</html>
```

Abaixo será apresentada uma tabela com outros eventos que podem ser utilizados com jQuery.

| **Eventos de Mouse** | |
|---|---|
| click() | Evento gerado quando se produz um click em um elemento da página. |

| | |
|---|---|
| dblclick() | Evento gerado quando se produz um duplo clique sobre um elemento. |
| hover() | Executada quando o mouse entra e sai de cima de um elemento. |
| mousedown() | Executado quando o usuário pressiona o botão do mouse. |
| mouseup() | Executado quando o usuário solta o botão do mouse. |
| mouseenter() | Executado quando o mouse for situado sobre o emento. |
| mouseleave() | Executado quando o mouse sai de cima de um elemento. |
| mousemove() | Executado quando o mouse é movimentado sobre o elemento. |
| mouseout() | Executado quando o mouse sai do elemento. |
| Eventos de teclado | |
| keydown() | Executado no momento em que uma tecla é pressionada, independente de se soltar ou manter a tecla pressionada. |
| keypress() | Executado quando uma tecla é pressionada. Executado uma vez se a tecla é pressionada e liberada. Executado diversas vezes se a tecla for pressionada e mantida. |
| keyup() | Executado no momento em que uma tecla é liberada após ser pressionada. |
| Eventos combinados de teclado e mouse | |
| focusin() | Evento executado quando um elemento ganha o foco. Este foco pode ser causado pelo clique do mouse ou pela tabulação do |

| | |
|---|---|
| | teclado. |
| focusout() | Este evento ocorre quando um elemento perde o foco, podendo ser perdido pelo uso do mouse ou tabulação do teclado. |
| focus() | Executado quando um elemento ganha o foco. |

#### 1.2.3.2 ALGUMAS FUNCIONALIDADES DO JQUERY

Além dos novos eventos e seletores o jQuery apresenta inúmeras facilidades que podem ser empregadas no desenvolvimento em JavaScript. Dentre estas facilidades está a execução de rotinas Ajax e também facilitadores relativos a manipulação de elementos. O uso de recursos Ajax será visto em outra disciplina, enquanto que agora serão apresentados alguns dos recursos que facilitam a programação em JavaScript.

**$.trim(*str*):** Este comando remove espaços na esquerda e na direita de uma strings. Exemplo:

```
<script>
        var nome = "  Juliano  ";
        nome = $.trim(nome);
</script>
```

**$.inArray(*valor, array*):** Retorna a primeira ocorrência do valor passado, que for encontrada no vetor passado. Exemplo:

```
<script>
        var nomes = new Array();
        var valorEncontrado;

        nomes[0] = "José";
        nomes[1] = "Carlos";
        nomes[2] = "Cassio";

        valorEncontrado = $.inArray("Carlos",nomes);

        alert(valorEncontrado);

</script>
```

**$.isArray(objeto):** Retorna verdadeiro se o objeto passado for um array JavaScript.

**$.unique(array):** Retorna um array de elementos somente com elementos únicos.

**$.type(objeto):** Determina o tipo do elemento. Exemplo:

```
<script>
        if ($.type(true) == "boolean")
        {
                document.write("Este tipo é booleano");
        }
</script>
```

**.val():** Retorna o valor (conteúdo) de um determinado objeto.

**.text():** Seta o valor para um determinado objeto. Exemplo:

```
<script>
        var conteudo = $("camponome").val();
        $("p").text(conteudo);
</script>
```

**.css():** Atribui ou permite a leitura de uma determinado valor de uma propriedade CSS do elemento em questão. Exemplo:

```
<script>
        //atribuindo a cor amarela ao fundo de um paragrafo
        $("p").css("background-color","yellow);
</script>
```

```
<script>
        //lendo a cor de fundo de um paragrafo para uma variável
        Var corDeFundo;
        corDeFundo = $("p").css("background-color");
</script>
```

### 1.2.4 FRAMEWORK JQUERY UI

O jQuery UI é uma biblioteca JavaScript criada em 2007 por um grupo de desenvolvedores que atualmente possuem como gerente de projeto Richard D. Worth.

A biblioteca jQuery UI foi criada com o objetivo de ser uma biblioteca para ser utilizada em conjunto com a biblioteca jQuery, objetivando a criação de elementos para interfaces de usuários ricas.

Esta biblioteca permite ao desenvolvedor criar elementos de interface com o usuário com um mínimo de código e sintaxe semelhante a biblioteca jQuery.

A biblioteca jQuery UI, bem como uma vasta documentação da mesma podem ser encontrados no site http://jqueryui.com/.

Os elementos que fazem parte da biblioteca jQuery UI foram classificados de acordo com as suas funcionalidades, o que pode ser observado na tabela abaixo.

| Grupo | Nome | Descrição |
|---|---|---|
| Componentes | Draggable (arrastar) | Cria elementos possíveis de serem arrastados pela interface. |
| | Droppable (soltar) | Cria elementos para receber os elementos arrastáveis. |
| | Resizable (dimensionar) | Cria elementos cujas dimensões podem ser controladas pelo usuário. |
| | Selectable (selecionar) | Cria elementos que possam ser selecionados pelo usuário, seja individualmente ou em grupos. |
| | Sortable (ordenar) | Cria elementos que possam ser ordenados pelo usuário com ação de arrastar e soltar. |
| Widgets | Accordion | Cria o efeito acordeão, para ocultar/mostrar conteúdo. |
| | Autocomplete | Apresenta ao usuário uma lista de sugestões de palavras à medida que ele digita em um campo texto. |
| | Button | Possibilidade de criar vários tipos de botões. |
| | Datepicker | Cria uma janela popup para seleção da data a ser digitada em um campo destinado a coletar uma data. |
| | Dialog | Cria vários tipos de janelas de diálogo, como janelas modais. |
| | Progressbar | Cria uma barra indicativa do andamento de uma tarefa. |
| | Slider | Cria um botão arrastável em uma guia para seleção de um valor compreendido em determinada faixa. |

| | | |
|---|---|---|
| | Tabs | Cria uma interface cuja navegação é feita com o uso de abas. |
| Efeitos | Color Animation | Este efeito tem por objetivo animar as cores de um elemento. |
| | Toggle class | Esses efeitos destinam-se a animar elementos baseados na manipulação dos seus atributos de classe. |
| | Add class | |
| | Remove class | |
| | Swich class | |
| | Effect | Aplica em um elemento uma série de efeitos de animação padrão da biblioteca jQuery, tais como os efeitos de esmaecimento, pulsação, balanço, sacudir, etc. |
| | Toggle | Aplica um dos efeitos padrão da biblioteca jQuery com a finalidade de alternar entre ocultar e exibir um conteúdo. |
| | Hide | Efeito destinado a ocultar um conteúdo. |
| | Show | Efeito destinado a mostrar um conteúdo. |
| Utilidades | Position | Destina-se a controlar e manipular o posicionamento de elementos na interface. |
| | Widgets | Destina-se a criação de widgets personalizados. |

Abaixo segue exemplo extraído da própria documentação, o qual demonstra a utilização do jQuery UI para o elemento datepicker.

```
<html>
<head>
  <title>jQuery UI Datepicker - Default functionality</title>
  <link rel="stylesheet" href="jquery-ui.css">
  <script src="jquery-1.10.2.js"></script>
  <script src="jquery-ui.js"></script>
  <script>
  $(function() {
    $( "#datepicker" ).datepicker();
  });
  </script>
</head>
<body>

         <p>Date: <input type="text" id="datepicker"></p>

</body>
</html>
```

## 1.3 FONTES BIBLIOGRÁFICAS

[BKK, 2005] Breitmann, Karin Koogan. Web semântica: a internet do future – Rio de Janeiro: LTC, 2005

[SMS, 2008] Silva, Maurício Samy. Criando sites com HTML: Sites de alta qualidade com HTML e CSS – São Paulo: Novatec Editora, 2008

[KJ, 2009] Kalbach, James. Design de navegação; tradução: Eduardo Kessler Piveta. – Porto Alegre: Bookman, 2009

[SMS, 2011] SILVA, Maurício Samy. HTML 5: a linguagem de marcação que revolucionou a web. 1 ed.. São Paulo: Novatec, 2011.

[PM, 2011] Pilgrim, Mark. HTML 5: entendendo e executando. – Rio de Janeiro, RJ: Alta Books, 2011.

Wikipédia: a enciclopédia livre. Acesso em setembro de 2014.

Modernzr: http://www.modernizr.com/ Acesso em novembro de 2014

[MSS, 2012] Silva, Mauricio Samy. jQuery UI: Componentes de interface rica. – São Paulo, SP: Novatec, 2012.