Unidade 8: DHTML - O HTML Dinâmico

Prof. Daniel Caetano

Objetivo: Capacitar o aluno para entententer e manipular todos os elementos do XHTML e CSS com o uso de JavaScript.

Bibliografia: W3,2009; MCLAUGHLIN,2008; MOZILLA,2009.

INTRODUÇÃO

Nas aulas anteriores foram apresentadas diversas tecnologias, como o HTML, CSS e JavaScript. De maneira intuitiva, foram apresentadas algumas interações superficiais entre essas tecnologias para produzir efeitos interessantes.

Entretanto, a criação de sites mais elaborados e aplicações para a Web exige um conhecimento mais aprofundado e apurado destas tecnologias e da forma com que os elementos de uma página - HTML e CSS - são organizados na memória, estrutura essa que ficou conhecida pelo nome de Document Object Model (DOM, Modelo de Objeto de Documento).

Nesta aula serão apresentados os detalhes do DOM e como manipular a maior parte de seus elementos usando JavaScript.

1. O QUE É DHTML

O DHTML surgiu na época do HTML4 e, por essa razão, algumas coisas são, hoje, diferentes do que eram há 10 anos atrás. As tecnologias básicas do DHTML são: HTML, CSS e JavaScript.

Para que o DHTML fosse possível, na ocasião do HTML4 todos os navegadores foram programados para que cada elemento da página fosse um objeto: o parágrafo é um objeto, a imagem é um objeto, a lista é um objeto... Cada objeto pode ter outros objetos dentro de si e todos os objetos de uma página estão organizdos entro de um objeto mestre chamado **document**.

A organização destes objetos na memória e a maneira de interagir com eles através do JavaScript ficou conhecida pelo nome de Modelo de Objeto de Documento (DOM, Document Object Model, em inglês).

Os sites dinâmicos e as aplicações web nada mais são que páginas comuns em que scripts muito bem elaborados manipulam o conteúdo do HTML e as propriedades do CSS de maneira a obter efeitos muito próximos aos de uma aplicação tradicional.

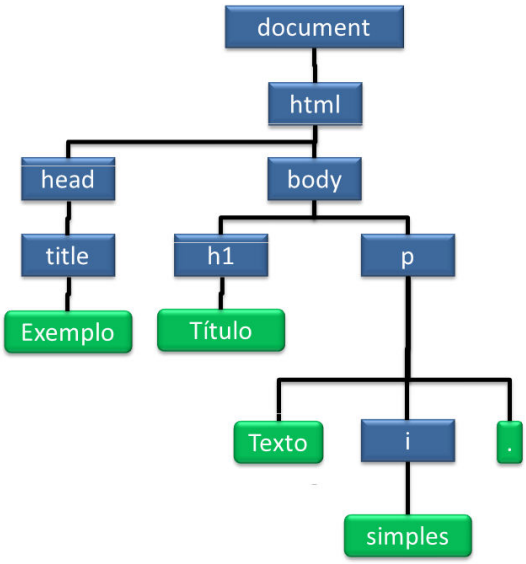
2. O QUE É O DOM?

Como já foi descrito, o Modelo de Objeto de Documento (DOM) é a essência do DHTML... mas o que é o DOM?

O DOM mapeia um documento HTML na forma de **uma árvore composta por nós**. Cada elemento marcado no HTML (parágrafo, lista, item de lista, imagem, div etc...) vira um nó no DOM. Cada nó pode ter outros nós dentro.

O nó raíz, que contém todas as nós do documento, é chamado criativamente de **document**. Observe um pequeno exemplo de uma estrutura em árvore construída pelo DOM para um pequeno HTML:

```
<html>
    <head>
        <title>Exemplo</title>
    </head>
    <body>
        <h1>Título</h1>
        <p>Texto <i>simples</i>.</p>
    </body>
</html>
```



Esses objetos podem ser **manipulados** pelo javascript. Por exemplo, se usarmos a seguinte linha JavaScript:

```
document.write("<p>Olá mundo!</p>");
```

Estaremos inserindo um ramo na árvore DOM. Onde exatamente? Aí depende... do ponto em que o código foi executado!

Considere os exemplos a seguir, que irão acrescentar esse texto **antes** e **depois** do <h1>, respectivamente.

Insere ANTES do h1:

```
<html>
	<head>
		<title>Exemplo</title>
	</head>
	<body>
		<script type="text/javascript"><!--
			document.write("<p>Olá mundo!</p>");
		--></script>
		<h1>Título</h1>
		<p>Texto <i>simples</i>.</p>
	</body>
</html>
```

Insere DEPOIS do h1:

```
<html>
	<head>
		<title>Exemplo</title>
	</head>
	<body>
		<h1>Título</h1>
		<script type="text/javascript"><!--
			document.write("<p>Olá mundo!</p>");
		--></script>
		<p>Texto <i>simples</i>.</p>
	</body>
</html>
```

O ponto onde o ramo é inserido depende do estágio da construção da árvore.

3. ACESSANDO O DOM

Manipular bem o DOM é a chava para uma aplicação interessante. Para manipular o DOM, primeiramente precisamos descobrir como **pegar uma referência para um elemento**. Por exemplo: se quiser mudar um texto de cor, preciso ter uma maneira de indicar para o navegador **qual** é esse elemento. Para pegar uma referência, o DOM nor fornece três funções principais, que serão vistas a seguir.

3.1. Pegando os elementos pelo ID

Observe o código abaixo:

```
<html>
	<head>
		<title>Exemplo</title>
	</head>
	<body>
		<h1 id="topo">Título</h1>
		<p>Texto <i>simples</i>.</p>
	</body>
</html>
```

Se quisermos escrever um JavaScript que modifica, por exemplo, a cor de fundo do <h1> cujo ID é "topo", podemos pegar uma referência para este elemento e modificá-lo assim:

```
var temp = document.getElementById("topo");	/* Pega a referência */
temp.style.backgroundColor = "red";
```

Como só pode existir um elemento com um dado **ID**, a função getElementById sempre retorna um único elemento. Observe que essa função é do objeto **document**.

3.2. Pegando os elementos pelo NOME

Observe o código abaixo:

```
<html>
	<head><title>Exemplo</title></head>
	<body>
		<form>
			<input type="text" name="cpf" />
		</form>
	</body>
</html>
```

Se quisermos escrever um JavaScript que modifica, por exemplo, a cor de fundo do campo <input> cujo nome é "cpf", podemos pegar uma referência para este elemento e modificá-lo assim:

```
var temp = document.getElementsByName("cpf");/* Pega as referências */
temp[0].style.backgroundColor = "red";
```

Observe que, como pode existir mais de um campo com o mesmo atributo **name**, a função getElementsByName sempre retorna **um vetor de elementos**, que podem ser acessados um a um pelo índice numérico.

Para saber quantos elementos foram retornados, basta acessar o **length**, assim:

```
var temp = document.getElementsByName("cpf");/* Pega as referências */
var num = temp.length;
window.alert(num);
```

Observe que a função **getElementsByName** é do objeto **document**.

3.3. Pegando os elementos pela TAG

Observe o código abaixo:

```
<html>
	<head>
		<title>Exemplo</title>
	</head>
	<body>
		<p>Texto <i>simples</i>.</p>
		<p>Outro texto.</p>
	</body>
</html>
```

Se quisermos escrever um JavaScript que modifica, por exemplo, a cor de fundo do segundo parágrafo, podemos pegar uma referência para este elemento e modificá-lo assim:

```
var temp = document.getElementsByTagName("p");	/* Pega as referências */
temp[1].style.backgroundColor = "red";
```

Observe que, como pode existir mais de um elemento marcado com a mesma **tag** a função getElementsByTagName sempre retorna **um vetor de elementos**, que podem ser acessados um a um pelo índice numérico.

Para saber quantos elementos foram retornados, basta acessar o **length**, assim:

```
var temp = document.getElementsByTagName("p");        /* Pega as referências */
var num = temp.length;
window.alert(num);
```

Observe que a função **getElementsByTagName** existe não apenas no **document**, mas também em qualquer elemento. Por essa razão, é possível fazer um código assim:

```
var temp = document.getElementsByTagName("p");        /* Pega as referências */
var elemento = temp[1].getElementsByTagname("i");
elemento.style.backgroundColor = "red";
```

Isso irá mudar apenas a cor de fundo da parte marcada com "i" dentro do segundo parágrafo encontrado.

3.4. Atributos que permite acessar outros elementos

Alguns atributos dos elementos permitem acessar outros elementos do XHTML:

parentNode	Referência para o nó pai
childNodes	Vetor dos nós filhos
firstChild	Referência para o primeiro nó filhote
lastChild	Referência para o último nó filhote
nodeValue	Valor do nó atual
nextSibling	Referência para o próximo elemento (inclui TextElements!)
previousSibling	Referência para o elemento anterior (inclui TextElements!)

4. MANIPULANDO O DOM

Podemos mudar os elementos do DOM de diversas formas. Uma delas é usando o atributo innerHTML, que permite modificar o conteúdo de um nó:

```
document.body.innerHTML = "<p>Olá!</p>"
```

É possível ainda usar funções para criar e remover elementos:

Nome	função	Quem tem
createElement("t")	Cria um elemento de tag	document
createTextNode("t")	Cria um elemento de texto	document
appendChild(e)	Acrecenta um nó filho ao elemento	todos os elementos
removeChild(e)	Remove um nó filho	todos os elementos
replaceChile(e1,e2)	Substitui um nó filho por outro	todos os elementos

5. MANIPULANDO O CSS PELO DOM

Podemos mudar os valores INLINE de CSS pelo DOM:

```
document.body.style.color = "blue";
```

Para mudar a classe de um elemento, basta usar:

```
document.body.className = "novaClasse";
```

Para acrescentar uma classe ao elemento:

```
document.body.className += " novaClasse";
```

A leitura do CSS, entretanto, não é tão óbvia. Isso não vai funcionar:

```
var cor = document.body.style.color;
```

Apenas alguns valores podem ser lidos diretamente, assim:

atributo	valor lido
offsetHeight	altura atual do elemento, em pixels
offsetWidth	largura atual do elemento, em pixels
offsetLeft	margem esquerda atual do elemento
offsetTop	margem superior atual do elemento

Outros valores precisam ser lidos de uma outra maneira, usando uma função:

```
function getStyle(oElm, strCssRule){
    var strValue = "";
    if(document.defaultView && document.defaultView.getComputedStyle) {
        strValue = document.defaultView.getComputedStyle(oElm, "").getPropertyValue(strCssRule);
        }
    else if(oElm.currentStyle) {
       strCssRule = strCssRule.replace(/\-(\w)/g, function (strMatch, p1) {
            return p1.toUpperCase();
            });
        strValue = oElm.currentStyle[strCssRule];
        }
    return strValue;
    }
```

Isso pode ser usado assim:

```
var el = document.getElementById("artigo");
var cor = getStyle(el,"background-color");
```

6. BIBLIOGRAFIA

W3 schools - CSS Tutorial. Disponível em < http://www.w3schools.com/ >. Visitado em 10 de Março de 2009.

MCLAUGHLIN, B. Use a Cabeça! Ajax. Alta Books, 2008.

MOZILLA Developer Connection. Disponível em < http://developer.mozilla.org/pt >. Visitado em 30 de março de 2009.