## Doctype

Doctype deve ser a primeira linha de código do documento antes da tag HTML, ele indica para o navegador e para outros meios qual a especificação de código utilizar, pois antes tínhamos Doctypes diferentes no xHTML.

```html

<!DOCTYPE html!>

```

## O elemento HTML

O elemento principal do DOM é sempre a tag HTML.

```html

<html lang="pt-br">

```

O atributo `LANG` é necessário para que os user-agents saibam qual a linguagem principal do

documento.

Lembre-se que o atributo LANG não é restrito ao elemento HTML, ele pode ser utilizado em qualquer

outro elemento para indicar o idioma do texto representado.

Para encontrar a listagem de códigos das linguagens, acesse:

http://www.w3.org/International/questions/qa-choosing-language-tags.

## HEAD

A Tag HEAD é onde fica toda a parte inteligente da página. No HEAD ficam os metadados.

Metadados são informações sobre a página e o conteúdo ali publicado.

No nosso exemplo há uma metatag responsável por chavear qual tabela de caractéres a página está

utilizando.

```html

<meta charset="utf-8">

```

Nas versões anteriores ao HTML5, essa tag era escrita da forma abaixo:

```html

<meta http-equiv="Content-Type" content="text/html; charset=utf-8">

```

## Novos elementos

Abaixo segue uma lista dos novos elementos e atributos incluídos no HTML5:

- section: A tag section define uma nova seção genérica no documento. Por exemplo, a home

de um website pode ser dividida em diversas seções: introdução ou destaque, novidades,

informação de contato e chamadas para conteúdo interno.

- nav: O elemento nav representa uma seção da página que contém links para outras partes

do website. Nem todos os grupos de links devem ser elementos nav, apenas aqueles

grupos que contém links importantes. Isso pode ser aplicado naqueles blocos de links

que geralmente são colocados no Rodapé e também para compor o menu principal do

site.

- article: O elemento article representa uma parte da página que poderá ser distribuído e

reutilizável em FEEDs por exemplo. Isto pode ser um post, artigo, um bloco de comentários

de usuários ou apenas um bloco de texto comum.

- aside: O elemento aside representa um bloco de conteúdo que referência o conteúdo que envolta do elemento aside. O aside pode ser representado por conteúdos em sidebars em textos impressos, publicidade ou até mesmo para criar um grupo de elementos nav e outras informações separados do conteúdo principal do website.

- hgroup: Este elemento consiste em um grupo de títulos. Ele serve para agrupar elementos de

título de H1 até H6 quando eles tem múltiplos níveis como título com subtítulos e etc.

- header: O elemento header representa um grupo de introdução ou elementos de navegação.

O elemento header pode ser utilizado para agrupar índices de conteúdos, campos de

busca ou até mesmo logos.

- footer: O elemento footer representa literalmente o rodapé da página. Seria o último elemento

do último elemento antes de fechar a tag HTML. O elemento footer não precisa

aparecer necessariamente no final de uma seção.

- time: Este elemento serve para marcar parte do texto que exibe um horário ou uma data precisa no calendário gregoriano.

- main:

# HTML5

Aula de 5 horas sobre HTML5 para a pós-graduação da UNOESC em Chapecó - SC.

## Elementos modificados

O elemento `b` passa a ter o mesmo nível semântico que um `span`, mas ainda mantém o estilo de negrito no texto. Contudo, ele não dá nenhuma importância para o text marcado com ele.

O elemento `i` também passa a ser um SPAN. O texto continua sendo itálico e para usuários de leitores de tela, a voz utilizada é modificada para indicar ênfase. Isso pode ser útil para marcar frases em outros idiomas, termos técnicos e etc.

## Novos tipos de campos

### input

O elemento input aceita os seguintes novos valores para o

atributo type:

#### tel

Telefone. Não há máscara de formatação ou validação, propositalmente,

visto não haver no mundo um padrão bem

definido para números de telefones. É claro que você pode

usar a nova API de validação de formulários para isso. Os agentes de usuário podem permitir a integração com sua agenda de contatos, o que é particularmente útil em

telefones celulares

#### search

Um campo de busca. A aparência e comportamento do campo pode mudar ligeiramente dependendo

do agente de usuário, para parecer com os demais campos de busca do sistema.

#### email

E-mail, com formatação e validação. O agente de usuário pode inclusive promover a integração

com sua agenda de contatos.

#### url

Um endereço web, também com formatação e validação.

#### Datas e horas

O campo de formulário pode conter qualquer um desses

valores no atributo type:

- datetime

- date

- month

- week

- time

- datetime-local

Todos devem ser validados e formatados pelo agente de usuário, que pode inclusive mostrar um calendário, um seletor de horário ou outro auxílio ao preenchimento que estiver disponível no sistema do usuário.

O atributo adicional step define, para os validadores e auxílios ao preenchimento, a diferença mínima entre dois horários. O valor de step é em segundos, e o valor padrão é 60. Assim, se você usar step="300" o usuário poderá fornecer como horários 7:00, 7:05 e 7:10, mas não 7:02 ou 7:08.

#### number

Aceita APENAS números.

Veja um exemplo do tipo number com seus atributos opcionais:

```html

<input name="valuex" type="number" value="12.4" step="0.2" min="0" max="20" />

```

#### range

Serve para faixas de valores

Vamos modificar o exemplo acima, apenas o valor de type, mudando de “number" para “range":

```html

<input name="valuex" type="range" value="12.4" step="0.2" min="0" max="20" />

```

#### color

O campo com type="color" é um seletor de cor. O agente de usuário pode mostrar um controle de seleção de cor ou outro auxílio que estiver disponível. O valor será uma cor no formato #ff6600.

## TIPOS DE DADOS E VALIDADORES

O HTML5 avançou bastante nos recursos de formulários, facilitando muito a vida de quem precisa desenvolver aplicações web baseadas em formulários.

### autofocus

Ao incluir em um campo de formulário o atributo autofocus, assim:

```html

<input name="login" autofocus >

```

O foco será colocado neste campo automaticamente ao carregar a página. Diferente das soluções em Javascript, o foco estará no campo tão logo ele seja criado, e não apenas ao final do carregamento da página. Isso evita o problema, muito comum quando você muda o foco com Javascript, de o usuário já estar em outro campo, digitando, quando o foco é mudado.

### Placeholder text

Você já deve ter visto um “placeholder". Tradicionalmente, vínhamos fazendo isso:

```html

<input name="q" value="Search here" onfocus="if(this.value=='Search here')this.value='">

```

Agora podemos fazer assim:

```html

<input name="q" placeholder="Search here">

```

### required

Para tornar um campo de formulário obrigatório (seu valor precisa ser preenchido) basta, em HTML5, incluir o atributo required:

```html

<input name="login" required>

```

### maxlength

O atributo maxlength limita a quantidade de caracteres em um campo de formulário. No HTML5, o elemento textarea também pode ter maxlength!

## Validação de formulários

Uma das tarefas mais enfadonhas de se fazer em Javascript é validar formulários. Infelizmente, é também uma das mais comuns. HTML5 facilita muito nossa vida ao validar formulários, tornando automática boa parte do processo. Em muitos casos, todo ele. Você já viu que pode tornar seus campos “espertos" com os novos valores para o atributo type, que já incluem validação para datas,

emails, URLs e números.

### pattern

O atributo pattern nos permite definir expressões regulares de validação, sem Javascript. Veja um exemplo de como validar CEP:

```html

<form>

<label for="CEP">CEP:

<input name="CEP" id="CEP" required pattern="\d{5}-?\d{3}" />

</label>

<input type="submit" value="Enviar" />

</form>

```

### novalidate e formnovalidate

Podem haver situações em que você precisa que um formulário não seja validado. Nestes casos, basta incluir no elemento form o atributo novalidate.

Outra situação comum é querer que o formulário não seja validado dependendo da ação de submit.

Nesse caso, você pode usar no botão de submit o atributo formnovalidate. Veja um exemplo:

```html

<form>

<label>nome: <input name="nome" required></label>

<label>email: <input name="email" type="email" required></label>

<label>mensagem: <textarea name="mensagem" required></textarea></label>

<input type="submit" name="action" value="Salvar rascunho" formnovalidate>

<input type="submit" name="action" value="Enviar">

</form>

```

## Custom validators

É claro que as validações padrão, embora atendam a maioria dos casos, não são suficientes para todas as situações. Muitas vezes você vai querer escrever sua própria função de validação Javascript.

Há alguns detalhes na especificação do HTML5 que vão ajudá-lo com isso:

1. O novo evento oninput é disparado quando algo é modificado no valor de um campo de formulário. Diferente de onchange, que é disparado ao final da edição, oninput é disparado ao editar. É diferente também de onkeyup e onkeypress, porque vai capturar qualquer modificação no valor do campo, feita com mouse, teclado ou outra interface qualquer.

2. O método setCustomValidity pode ser invocado por você. Ele recebe uma string. Se a string

for vazia, o campo será marcado como válido. Caso contrário, será marcado como inválido.

Com isso, você pode inserir suas validações no campo de formulário e deixar o navegador fazer o resto. Não é mais preciso capturar o evento submit e tratá-lo. Veja, por exemplo, este formulário com validação de CPF:

```html

<!DOCTYPE html>

<html lang="pt-BR">

<head>

<meta charset="UTF-8" />

<title>Custom validator</title>

<!-- O arquivo cpf.js contém a função validaCPF, que

recebe uma string e retorna true ou false. -->

<script src="cpf.js"></script>

<script>

function vCPF(i){

i.setCustomValidity(validaCPF(i.value) ? "" : 'CPF inválido!')

}

</script>

</head>

<body>

<form>

<label>CPF: <input name="cpf" oninput="vCPF(this)" /></label>

<input type="submit" value="Enviar" />

</form>

</body>

</html>

```

Você percebeu que o elemento do `input` é passado via `this` para a função `vCPF` a qual chama para esse elemento a função `setCustomValidity`, nesse caso se você passar uma \*String\* vazia siginifica que validou e qualquer string enviada, diferente de `""` será mostrada ao usuário como erro.

#### Exercício

Criar um formulário com os novos elementos do HTML5 e uma validação para CNPJ utilizando `oninput`

## DETALHES E CONTEÚDO EDITÁVEL

Vejamos mais duas coisas que você certamente já fez mais de uma vez e foram simplificadas pelo

HTML5.

Detalhes e sumário

Veja um exemplo de uso dos novos elementos details e summary:

```html

<details>

<summary>Copiando <progress max="39248" value="14718"> 37,5%</summary>

<dl>

<dt>Tamanho total:</dt>

<dd>39.248KB</dd>

<dt>Transferido:</dt>

<dd>14.718</dd>

<dt>Taxa de transferência:</dt>

<dd>127KB/s</dd>

<dt>Nome do arquivo:</dt>

<dd>HTML5.mp4</dd>

</dl>

</details>

```

![fechado](https://cldup.com/qJrb99CGJJ-3000x3000.png)

![aberto](https://cldup.com/vjuQx5156k-1200x1200.png)

## Drag and Drop

A API de Drag and Drop é relativamente simples. Basicamente, inserir o atributo

`draggable='true'` num elemento o torna arrastável. E há uma série de eventos que você pode

tratar.

Os eventos do objeto sendo arrastado são:

- dragstart: O objeto começou a ser arrastado. O evento que a função recebe tem um atributo target, que contém o objeto sendo arrastado.

- drag: O objeto está sendo arrastado

- dragend: A ação de arrastar terminou

O objeto sobre o qual outro é arrastado sofre os seguintes eventos:

- dragenter: O objeto sendo arrastado entrou no objeto target

- dragleave: O objeto sendo arrastado deixou o objeto target

- dragover: O objeto sendo arrastado se move sobre o objeto target

- drop: O objeto sendo arrastado foi solto sobre o objeto target

### Detalhes importantes

A ação padrão do evento dragover é cancelar a ação de dragging atual. Assim, nos objetos que devem receber drop, é preciso setar uma ação de dragover com, no mínimo, return false.

Seleções de texto são automaticamente arrastáveis, não precisam do atributo draggable. E se você quiser criar uma área para onde seleções de texto possam ser arrastadas, basta tratar esses mesmos eventos.

Por fim, todas funções de tratamento de evento de drag recebem um objeto de evento que contém uma propriedade dataTransfer, um dataset comum a todos os eventos durante essa operação de drag.

### Revisão ortográfica e gramatical

Os agentes de usuário podem oferecer recursos de revisão ortográfica e gramatical, dependendo do que houver disponível em cada plataforma. Os desenvolvedores podem controlar o comportamento dessa ferramenta através do atributo spellcheck. Inserir spellcheck="true" num

elemento faz com que a revisão esteja habilitada para ele. Você também pode desabilitar a revisão para determinado elemento, inserindo spellcheck="false".

## Áudio

Para inserir áudio em uma página web, basta usar o elemento audio:

```html

<audio src="mus.oga" controls="true" autoplay="true" />

```

O valor de controls define se um controle de áudio, com botões de play, pause, volume, barra deprogresso, contador de tempo, etc. será exibido na tela. Se for setado como “false", será preciso controlar o player via javascript, com métodos como play() e pause(). O valor de autoplay definese o áudio vai começar a tocar assim que a página carregar.

### Origens alternativas de áudio

Todo agente de usuário deveria suportar o codec livre OggVorbis, mas, infelizmente, pode acontecer de seu arquivo oga não tocar no computador ou celular de alguém. Quem sabe do seu chefe ou seu cliente. Então é preciso saber como oferecer um formato alternativo de áudio. Fazemos assim:

```html

<audio controls="true" autoplay="true">

<source src="mus.oga" />

<source src="mus.mp3" />

<source src="mus.wma" />

</audio>

```

Claro, o agente de usuário pode ainda não saber tocar nenhum desses formatos, ou sequer ter suporte a áudio. Para esses casos, ofereça um conteúdo alternativo:

```html

<audio controls="true" autoplay="true">

<source src="mus.oga" />

<source src="mus.mp3" />

<source src="mus.wma" />

<p>Faça o <a href="mus.mp3">download da música</a>.</p>

</audio>

```

## Vídeo

O uso de vídeo é muito semelhante ao de áudio:

```html

<video src="u.ogv" width="400" height="300" />

```

E com vários elementos source:

```html

<video controls="true" autoplay="true" width="400" height="300">

<source src="u.ogv" />

<source src="u.mp4" />

<source src="u.wmv" />

<p>Faça o <a href="u.mp4">download do vídeo</a>.</p>

</video>

```

### Codecs

É muito importante que você inclua, nos seus elementos source de áudio e vídeo, informação a respeito do container e codecs utilizados. Isso vai evitar que o navegador tenha que baixar, pelo menos parcialmente, o arquivo de mídia para, depois, descobrir que não consegue tocá-lo. É importante lembrar que a extensão do arquivo não é informação relevante para isso, pelo contrário, não

significa nada. Uma URL pode não ter extensão de arquivo

e pode levar a um redirecionamento.

Para indicar ao navegador o container e codecs de determinado arquivo, usa-se o atributo type,

no formato:

```

type='MIME-type do container; codecs="codec de vídeo, codec de

áudio"'

```

Por exemplo, um vídeo em Ogg, usando os codecs Theora e Vorbis, terá seu source assim:

```html

<source src='video.ogv' type='video/ogg; codecs="theora, vorbis"'>

```

Com MPEG-4 a coisa é um pouco mais complicada, por que é preciso indicar ao navegador também

o profile do codec de vídeo utilizado. Veja um exemplo:

```html

<source src='video.mp4' type='video/mp4; codecs="mp4v.20.240, mp4a.40.2"'>

```

## Device

Você pode inserir em seu HTML um elemento de acesso

à webcam do usuário, assim:

```html

<device type="media">

```

Isso vai exibir uma interface solicitando ao usuário acesso a sua webcam. Se ele tiver mais de uma, também será permitido que ele escolha que webcam usar. O atributo media também pode conter o valor “fs", que vai abrir

uma caixa de seleção no sistema de arquivos, permitindo ao usuário escolher um arquivo para fazer stream.

O passo seguinte é conectar o stream desse seu elemento device a alguma coisa. Veja, por exemplo, como conectá-lo a um elemento video na própria página, fazendo com que o usuário possa ver a imagem de sua própria webcam:

```html

<!DOCTYPE html>

<html lang="en-US">

<head>

<meta charset="UTF-8" />

<title>Videochat, step 1</title>

<script>

function update(stream) {

document.getElementsByTagName('video')[0].src = stream.url;

}

</script>

</head>

<body>

<p>Para começar, selecione a camera: <device type="media" change="update(this.data)"></p>

<video autoplay />

</body>

</html>

```

## Streams

Você deve ter notado, no script acima, que a função de update recebe um parâmetro stream. Tratase de um objeto da classe Stream, que possui uma propriedade url, que já usamos acima, e um método record. O método record inicia a gravação do stream e retorna um objeto StreamRecorder.

Esse último possui um método stop, que retorna o arquivo que foi gravado.

## WebRTC

WebRTC, conhecido também como Web Real-Time Communications, é um projeto de software livre promovido pelo Google, Mozilla e outros e permite Comunicações em tempo real livre de plugin via API do Javascript. Ele facilita aplicativos entre navegadores para chamada de voz, bate-papo por vídeo e compartilhamento de arquivo.

## MathML

O HTML5 incorpora o padrão MathML. Trata-se de uma linguagem de marcação, baseada em XML, para representação de fórmulas matemáticas. Você pode ler mais sobre MathML em [http://www.w3.org/Math/](http://www.w3.org/Math/). Para incorporar código MathML em seu documento HTML5, não precisa fazer declarações especiais. Basta escrever normalmente o código, iniciando com um elemento

math. Veja este exemplo:

```html

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8" />

<title>MathML</title>

</head>

<body>

<math>

<mrow>

<mi>x</mi>

<mo>=</mo>

<mfrac>

<mrow>

<mo form="prefix">&minus;</mo>

<mi>b</mi>

<mo>&PlusMinus;</mo>

<msqrt>

<msup>

<mi>b</mi>

<mn>2</mn>

</msup>

<mo>&minus;</mo>

<mn>4</mn>

<mo>&InvisibleTimes;</mo>

<mi>a</mi>

<mo>&InvisibleTimes;</mo>

<mi>c</mi>

</msqrt>

</mrow>

<mrow>

<mn>2</mn>

<mo>&InvisibleTimes;</mo>

<mi>a</mi>

</mrow>

</mfrac>

</mrow>

</math>

</body>

</html>

```

ps: Testem no Firefox.

Mesmo que você nunca tenha visto MathML, e este código pareça um pouco assustador, dê uma olhada com calma no código, comparando com a imagem do resultado, e você vai perceber que é muito simples. Talvez algo que possa deixá-lo confuso é a entidade `&InvisibleTimes;`, que aparece algumas vezes no código. Ela está lá para separar os fatores 4ac, por exemplo. Esses valores

são multiplicados, é o que a fórmula representa, mas não queremos colocar um operador de multiplicação entre eles, porque por convenção se simplesmente escrevemos 4ac qualquer leitor saberá que isso é uma multiplicação.

Por que então se preocupar em inserir `&InvisibleTimes;`? Você vai notar que se remover a entidade e a tag mo correspondente o resultado visual será o mesmo. Colocamos `&InvisibleTimes;` porque MathML não é só visual, é semântica. Um outro agente de usuário pode ter recursos de importar essa fórmula para uma ferramenta de cálculo, por exemplo.

## SVG

Assim como MathML, SVG é uma outra linguagem XML que pode ser incorporada com facilidade em HTML5. Você pode ler mais sobre SVG em http://www.w3.org/Graphics/SVG/. SVG é uma linguagem para marcação de gráficos vetoriais. Vejamos um exemplo bem simples:

```html

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8" />

<title>SVG</title>

</head>

<body>

<svg width="400" height="400">

<!-- Um retângulo: -->

<rect x="10" y="10" width="150" height="50" stroke="#000000" stroke-width="5"ll="#FF0000" />

<!-- Um polígono: -->

<polygon fill="red" stroke="blue" stroke-width="10"

points="250,75 279,161 369,161 297,215

323,301 250,250 177,301 203,215

131,161 221,161" />

<!-- Um círculo -->

<circle cx="70" cy="240" r="60" stroke="#00FF00" stroke-width="5"

ll="#FFFFFF" />

</svg>

</body>

</html>

```

É possível fazer muito mais com SVG. A maioria dos editores de gráficos vetoriais hoje exporta e importa automaticamente SVG, permitindo a um designer construir um gráfico em seu editor vetorial predileto e exportá-lo diretamente. Em seguida, um programador pode construir javascript

que manipula esse SVG, usando os métodos do DOM. Com isso você pode ter gráficos dinâmicos, com animação, escaláveis e com excelente qualidade visual, programáveis em Javascript, sem tecnologias proprietárias e plugins.

## Canvas

A Canvas API permite a você desenhar na tela do navegador via Javascript. O único elemento HTML existente para isso é o elemento canvas, o resto todo é feito via Javascript. Veja como inserir o elemento canvas numa página:

```html

<canvas id="x" width="300" height="300"></canvas>

```

Isso vai exibir um retângulo vazio. Para desenhar nele, primeiro obtemos o contexto de desenho, com Javascript:

```js

context=document.getElementById('x').getContext('2d')

```

Agora que temos um contexto, podemos desenhar nele.

Vamos começar com um simples retângulo:

```js

context.fillRect(10, 10, 50, 150)

```

Simples, não? Que tal tentarmos algo um pouco mais

complexo? Dê uma olhada no exemplo:

```html

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8" />

<title>Canvas API</title>

</head>

<body>

<canvas id="x" width="300" height="300"></canvas>

<button onclick="desenhar()">desenhar</button>

<script>

function desenhar(){

// Obtemos o contexto

context=document.getElementById('x').getContext('2d')

//Iniciamos um novo desenho

context.beginPath()

//Movemos a caneta para o inicio do desenho

context.moveTo(150,50)

//Desenhamos as linhas

context.lineTo(220,250)

context.lineTo(50,125)

context.lineTo(250,125)

context.lineTo(80,250)

context.lineTo(150,50)

//O desenho não é de verdade enquanto você

//não mandar o contexto pintá-lo.

//Vamos pintar o interior de amarelo

context.fillStyle='#ff0'

context.fill()

//Vamos pintar as linhas de vermelho.

context.strokeStyle='#f00'

context.stroke()

}

</script>

</body>

</html>

```

E veja o que acontece quando se clica no botão.

Há muito mais para você estudar se quiser se aprofundar na Canvas API. Apenas para que você tenha uma idéia, é possível desenhar texto, sombras, gradientes, incluir imagens no canvas, manipular os pixels, rotacionar e transformar os objetos.

### Canvas e SVG

Uma dúvida muito comum é quando usar Canvas, quando usar SVG. Para saber escolher, é preciso entender as diferenças entre um e outro. SVG é vetorial, e baseado em XML, logo, acessível via DOM. Canvas é desenhado pixel a pixel, via Javascript.

Assim, as vantagens do SVG são:

- O conteúdo é acessível a leitores de tela

- O gráfico é escalável, não perde resolução ou serrilha ao redimensionar

- O conteúdo é acessível via DOM

E as vantagens do Canvas:

- A performance é muito superior ao SVG na maioria dos casos

- É fácil desenhar via Javascript. Em SVG, é preciso fazer seu script escrever XML para você.

Com Canvas você só manda desenhar, e pronto.

## Server-Sent Events

A Server-Sent Events API é uma maneira de inverter o fluxo das aplicações Ajax, fazendo com que o servidor possa disparar o envio de dados ao agente de usuário. Para isso, cria-se, no agente de usuário, um objeto EventSource:

```js

es=new EventSource('meu/servidor');

```

Isso vai abrir uma conexão HTTP para 'meu/servidor' e mantê-la escutando. Cada vez que o servidor enviar eventos para esse cliente, será disparado o evento message do objeto EventSource. Veja um exemplo:

```js

es.onmessage=function(e){

alert(“Chegaram dados: “+e.data)

}

```

Isso pode ser usado, por exemplo, para implementar uma interface de chat ou um monitor de status de alguma operação demorada ocorrendo no servidor.

### O protocolo de comunicação

Em nosso exemplo acima, a página comm.php envia eventos para o agente de usuário. Você não precisa se preocupar em saber como isso funciona do lado do cliente, uma vez que o agente de usuário faz todo o trabalho. Mas é importante que saiba como isso deve funcionar do lado do servidor. A URL de

comunicação deve devolver ao cliente um header `Content-type: text/event-stream`. Em seguida, envia as mensagens, que são blocos de texto separados um do outro por uma linha em branco:

## DOM

O Modelo de Objetos do Documento (DOM, na sigla em inglês) é a interface entre a linguagem Javascript e os objetos do HTML. DOM é o método padrão para construção de aplicações ricas com Javascript e é amplamente conhecido e utilizado. Neste capítulo, supondo que você já conhece DOM para HTML 4 ou XHTML, vamos nos focar na diferença entre as versões anteriores do DOM

e a do HTML 5.

Os primeiros navegadores a incorporar um motor de Javascript tinham alert, prompt, document. write e mais meia dúzia de maneiras de se interagir com o usuário. E só. A idéia de acessar a árvorede objetos do HTML trouxe poder às interfaces com o usuário na web. A idéia era tão boa que os fabricantes de navegadores não puderam esperar até que tivéssemos uma especificação padrão que atendesse suas necessidades, e criaram cada um seu próprio método de resolver o problema. Isso resultou em anos e anos de incompatibilidade, em que era preciso escrever uma versão de seus scripts para cada navegador.

Queremos, com certeza, evitar uma nova guerra de padrões. Por isso recomendamos a você, por mais sedutor que pareça utilizar um recurso proprietário Javascript, que se atenha ao DOM.

### getElementsByClassName

Esse é um sonho antigo de todo desenvolvedor Javascript. Com HTML5 você pode fazer:

```js

destaques = document.getElementsByClassName('destaque')

```

E isso retornará todos os elementos do HTML que possuem a classe “destaque".

### innerHTML

Outro sonho antigo que se torna realidade. A propriedade innerHTML é uma idéia tão boa que todos os navegadores atuais já a suportam há muito tempo e todo desenvolvedor web sabe usá-la.

Apesar disso, ela nunca havia sido descrita como um padrão.

Se porventura você nunca viu a propriedade innerHTML em ação (puxa, onde você estava nos últimos dez anos?) saiba que ela contém uma string, o conteúdo HTML da página. E você tem acesso

de leitura e escrita a essa propriedade.

Veja um exemplo de innerHTML:

```js

function adicionaItem(nome){

document.getElementById('lista').innerHTML += '<li>'+nome+'</li>';

}

```

### activeElement e hasFocus()

O documento HTML5 tem uma nova propriedade, activeElement, que contém o elemento que possui o foco no momento. O documento também possui o método hasFocus(), que retorna true se o documento contém o foco. Seu usuário pode estar trabalhando com múltiplas janelas, abas, frames, ou mesmo ter alternado para outro aplicativo deixando o navegador com sua aplicação Javascript rodando em segundo plano. O método hasFocus() é uma conveniente maneira de tratar ações que dependem do foco na aplicação atual.

Veja um exemplo de script dependente de foco:

```html

<!DOCTYPE html>

<html lang="pt-BR">

<head>

<meta charset="UTF-8" />

<title>Notifier</title>

<script>

function notify(text){

document.getElementById('msg').innerHTML+='<p>'+text+'</p>'

titleFlick()

}

function titleFlick(){

if(document.hasFocus()){

document.title='TEM FOCO'

return

}

document.title = ( document.title =='Notifier') ? '\* Notifier' : 'Notifier'

setTimeout('titleFlick()',500)

}

</script>

</head>

<body>

<input type="button" id="notify" value="Notify in 5 seconds"

onclick="notify('Irá notificar em 5 segundos...');setTimeout('notify(\'Eventoo!\')',5000)" />

<div id="msg"></div>

</body>

</html>

```