



HTML 5 e CSS 3

Moacyr Ferreira Minéro

[Formação Front-End – Módulo II]

[D R C Treinamentos]

Nota ao Aluno

Acompanhando a tendência do mercado de aplicações *web*, a DRC preparou uma formação técnica com o objetivo de trazer conhecimentos necessários ao desenvolvimento de um profissional Front-End.

Esse profissional é responsável pela construção de *interfaces* interativas para uma aplicação *web*, necessitando reunir as seguintes habilidades :

- ✓ Conceitos de Design para midias digitais;
- ✓ Marcação XHTML e CSS;
- ✓ Marcação HTML 5 e CSS 3;
- ✓ Lógica de programação;
- ✓ Programação de interface com a linguagem Javascript e jQuery;
- ✓ Gerenciamento de conteúdo via CMS (Wordpress, Joomla e outros);

Com os conhecimentos acima citados esse profissional está apto a integrar equipes de agências digitais, *software-houses* e departamento de TI em empresas envolvidas com aplicações *web*, sendo o responsável pela integração do projeto vindo do departamento de criação (*layout*) com a estrutura digital, e preparando-a para integração com a parte lógica chamada *back-end* (programação com o servidor e manipulação de banco de dados);

Um profissional Front-End pode elevar o seu nível técnico adquirindo conhecimentos necessários para integrar uma equipe *back-end*, e para isso necessita das seguintes habilidades técnicas:

- ✓ Conhecimento de programação servidor com linguagens do tipo: PHP, C Sharp, Java Server Faces, Phyton e Ruby on Rails.
- ✓ Conhecimento do conceito de Orientação a Objetos;
- ✓ Manipulação de informações via banco de dados como mySQL, SQL Server e Oracle;

A web, como um mercado extremamente dinâmico, exige que o profissional acompanhe seu ritmo evolutivo e responda com qualidade e eficiência aos desafios apresentados.

E para fazer parte desse mercado em franca expansão a DRC está junto com você, para prepará-lo com o que há de mais atual em desenvolvimento de aplicações para *web*.

Consulte nossa grade de cursos e fale com nossos Consultores de Negócios!

Atenciosamente,

DRC Treinamentos
Coordenação WEB

*** CONTEÚDO GERAL ***

■ Capítulo 01: Apresentação Geral do HTML 5	02
■ Capítulo 02: Elementos HTML 5.....	07
■ Capítulo 03: Formulário HTML 5.....	13
■ Capítulo 04: API de Áudio.....	29
■ Capítulo 05: API Vídeo	39
■ Capítulo 06: Elementos Validadores de Dados.....	50
■ Capítulo 07: API Canvas	60
■ Capítulo 08: API Arrastar e Soltar.....	85
■ Capítulo 09: API Geolocalização	92
■ Capítulo 10: API Armazenamento Local	104
■ Capítulo 11: CSS 3.....	113
■ Capítulo 12: Web Fonts	127
■ Capítulo 13: Textos.....	130
■ Capítulo 14: Bordas.....	136
■ Capítulo 15: Fundos	141
■ Capítulo 16: Sombras.....	146
■ Capítulo 17: Transformações	149
■ Capítulo 18: Transições	159
■ Capítulo 19: Media Queries.....	165
■ Capítulo 20: Template Layout	172
■ Capítulo 21: User Interface	180

Capítulo 01: Apresentação Geral do HTML 5

* A Evolução da Linguagem de Marcação HTML	7
* Padrões Web (Web Standards).....	8
* Ferramentas de Desenvolvimento Web.....	9
* Modelo de Documento HTML 5.....	19
* Navegadores <i>versus</i> HTML 5	20
* Elementos STYLE e SCRIPT	20

A Evolução da Linguagem HTML

Ao falar de internet é impossível não mencionar a principal linguagem responsável por sua evolução, o Hypertext Markup Language, também conhecido pelo acrônimo¹ HTML.

Seu criador foi o matemático Tim Berners Lee para suprir a necessidade de comunicação entre ele e um grupo de colegas pesquisadores. Com o crescimento da internet pública (modelo que serviria de base para a internet atual), o HTML tornou a base para o desenvolvimento de aplicações e recursos para a transmissão de informações na World Wide Web.

As primeiras especificações foram publicadas por Berners Lee no ano de 1993 com a ajuda de seu colega Dan Connolly na IETF – Internet Engineering Task Force.

A IETF criou um grupo de trabalho específico para o HTML, publicando a versão 2.0 no ano de 1995.

A partir do ano de 1996 o desenvolvimento de especificações foi atribuído e mantido por um grupo de empresas fabricantes de software denominada World Wide Consortium, ou simplesmente W3C.

A última versão oficial das recomendações do HTML foi lançada no ano de 1999, chamada de release 4.01.

A estrutura de um documento HTML possui os seguintes elementos:

```
<html>
  <head>
    <title>Título do Documento</title>
  </head>
  <body>
    Elementos de visualização
  </body>
</html>
```

Com a expansão frenética da internet e a crescente necessidade de padrões de desenvolvimento para aplicações, surge a primeira reformulação oficial desde a versão HTML 4.01, chamada de XHTML – EXtensible Hypertext Markup Language, baseada na linguagem XML.

A nova linguagem combina os elementos de marcação HTML com as regras do XML, visando a exibição num maior número de dispositivos (pc, celular, televisão, palm, impressora e etc).

As regras de marcação do XHTML são mais rígidas e consistentes se comparadas com o HTML, permitindo uma maior padronização de projetos através da diretiva² de DOCTYPE (Tipo de Documento) colocada no início da página XHTML.

As definições de DTD (Definição de Tipo de Documento) para o XHTML criaram a necessidade de uma estrutura de marcação mais limpa, tornando a separação do código e da estrutura de design nas aplicações um caminho inevitável.

A partir desse ponto começa uma ênfase nas técnicas de CSS e layouts sem o uso de tabelas (conhecido como tableless), buscando oferecer maior nível de acessibilidade ao usuário com uma marcação mais semântica³.

¹ Acrônimo: Termo ou palavra formada pelas letras ou sílabas iniciais de palavras sucessivas de uma locução, ou pela maioria delas.

² Diretiva: Conjunto de instruções padronizadas para processamento de informações.

³ Semântica: Estudo do significado de cada palavra em um determinado idioma.

A seguir a estrutura de um documento XHTML:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml11-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Título do Documento</title>
</head>
<body>
    Elementos de visualização
</body>
</html>
```

No ano de 2006 o W3C retoma os trabalhos sobre o HTML publicando o primeiro resumo de especificações em 2008 para o HTML 5, parando definitivamente o grupo de trabalho envolvido nas especificações para o XHTML release 2.

Padrões Web (Web Standards)

Primeiro, vamos entender a semântica da palavra Padrão:

"*Modelo reconhecido oficialmente e que serve como base para reprodução de novas unidades*".

Partindo deste ponto, voltamos ao começo da história da web onde não havia praticamente nenhum conhecimento sobre essa nova mídia de massa que começava a despontar.

Os profissionais envolvidos nessa onda garimpavam em várias partes os poucos materiais que estavam disponível, tentando entender como participar ativamente desse novo mercado criando suas primeiras aplicações web.

Nesse momento começava o movimento do "*eu_faço_do_meu_jeito.com*", onde cada grupo profissional e empresa envolvida com a web interpretava as informações de forma peculiar.

Não demorou muito para a instalação do caos no ambiente web, e buscando uma nova direção nesse processo criou-se um grupo chamado World Wide Web Consortium, mais conhecido como W3C, cuja missão desse grupo é colocar ordem no segmento buscando definir padrões para o desenvolvimento de aplicações para a internet.

Com isso surge o Web Standards Project com a proposta de criar um conjunto de linguagens estruturadas em regras que servissem de padrão no desenvolvimento de aplicações, e fossem interpretadas da mesma forma em qualquer lugar do mundo por fabricantes de *hardware*, *software* e profissionais de desenvolvimento.

Componentes que fazem parte dos Padrões Web (Web Standards) são:

- Documento de Modelo de Objetos (D.O.M.);
- Linguagem de marcação HTML;
- Linguagem de marcação XHTML;
- Linguagem de marcação HTML 5;
- Linguagem de marcação XML;
- Linguagem de script ECMAScript (Javascript);
- Linguagem de estilização/design CSS.

Para saber mais consulte:

<http://www.webstandards.org>

<http://www.w3.org>

Motores de Rederização (Browsers)

No início da *web* um dos problemas que mais atormentava os profissionais de desenvolvimento era chamado de "Compatibilidade". Num sentido figurado, a guerra dos *browsers* deixou muitos feridos nesses longos anos de *web*.

Tentar compatibilizar um layout nos principais navegadores da época era uma tarefa de super homem. Quem não se lembra da famosa frase no rodapé dos sites: "Melhor visualizado no browser X versão 3.0". Terrível não!

Com a chegada dos Padrões Web grande parte desse problema pôde ser controlado. Eu disse controlado e não resolvido! Lembre-se que temos o Internet Explorer.

Portanto, enquanto não conseguimos o controle absoluto temos que conhecer os principais motores de renderização usados pelos fabricantes de browser, buscando focar o desenvolvimento de aplicações *web* de forma a ser interpretada pela maior parte dos mesmos.

Assim, conseguimos atingir o maior número de dispositivos de acordo com o navegador que for utilizado por ele.

Exemplo: Dispositivos Apple utilizam o navegador Safari para acesso a *web*.

Abaixo segue a tabela dos motores de renderização e seus respectivos navegadores.

Tabela 1 - Motores de Renderização

Motor	Browser
<i>Webkit</i>	Chrome e Safari
<i>Gecko</i>	Firefox e Camino
<i>Trident</i>	Internet Explorer
<i>Presto</i>	Opera

Ferramentas de Desenvolvimento Web

Ao iniciar o desenvolvimento de aplicações para *web* o profissional deve reunir algumas ferramentas fundamentais para o sucesso de seu trabalho. Como a forma com que cada profissional vai trabalhar na criação de sua aplicação é peculiar, vou apresentar as ferramentas mais utilizadas pelo mercado e cada um escolhe a que mais lhe for conveniente.

Linguagem de Programação

As linguagens de programação são responsáveis pela interatividade da aplicação com o usuário. Elas estão divididas em dois segmentos:

- Client-Side
- Server-Side

As linguagens chamadas *Client-Side* são processadas pelo browser, sendo responsáveis por efeitos visuais, validação de informações e personalização de dados do usuário. As mais usadas são:

- Javascript
- jQuery

As linguagens chamadas *Server-Side* são processadas totalmente pelo servidor e devolvidas em código HTML para o browser, sendo responsáveis pela interação com base de dados e funções de nível servidor. As mais usadas são:

- ASP Net
- C# (lê C Sharp)
- JSF – Java Server Faces
- PHP – Pre Hypertext Processor

Editores HTML

Para a escrita do código de programação a princípio é suficiente apenas o famoso bloco de notas, mas por questões de praticidade e performance, a melhor opção é um editor HTML que permita a identificação de erros de digitação, janela para auto completar comandos, estruturação de pasta raiz e sub-pastas para a aplicação e outros recursos.

Sistema operacional Windows

- Dreamweaver
- Eclipse
- Notepad++

Sistema operacional Mac OS

- Eclipse
- Coda
- NVU

Browsers

Os *browsers* são as ferramentas mais importantes no desenvolvimento *web*, pois sem eles não teríamos uma visão abrangente do comportamento da aplicação, tanto em termos de programação, como em termos de design.

Os mais usados são:

- Internet Explorer (versões 7, 8 e 9)
- Firefox
- Google Chrome
- Opera
- Safari

Ferramentas Auxiliares

Existem no mercado um conjunto de ferramentas que podem ser agregadas a estrutura dos *browsers*, para apoiar e otimizar o desenvolvimento de aplicações *web*.

Debugadores de código HTML e CSS

- Firebug (<https://addons.mozilla.org/pt-br/firefox/addon/firebug/>)

Biblioteca de identificação de características HTML 5

- Modernizr (<http://www.modernizr.com/>)

Capítulo 02: Elementos HTML 5

* Modelo de Documento HTML 5	08
* Tipos de Conteúdo	10
* Novos Elementos de Marcação.....	10

Modelo de Documento HTML 5

DOCTYPE

É o acrônimo da palavra Document Type Definition, que significa Definição do Tipo de Documento, que é o elemento responsável pelo controle da forma de renderização de um documento HTML de acordo com os padrões estabelecidos pelo W3C.

A nova estrutura da linguagem HTML 5 possui uma marcação de código mais flexível e enxuta, não necessitando mais do link para o arquivo do tipo de documento (DTD), conforme o código abaixo:

Exemplo:

```
<!DOCTYPE html>
<html>
<head>
<title>HTML 5 - A Nova Onda</title>
</head>
<body>
</body>
</html>
```

* A diretiva de DOCTYPE deve ser colocada na primeira linha do documento HTML.

CHARSET

Outro elemento importante para a correta renderização do documento é a codificação de caracteres do idioma.

A codificação pode ser feita com o uso do comando META em conjunto com o atributo CHARSET, e que foi simplificada também no HTML 5.

Exemplo:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>HTML 5 - A Nova Onda</title>
</head>
<body>
</body>
</html>
```

LANG

O atributo LANG define o idioma do documento HTML, complementando a codificação dos caracteres pelo atributo CHARSET.

Exemplo:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<meta lang="pt-br">
<title>HTML 5 - A Nova Onda</title>
</head>
<body>
</body>
</html>
```

SINTAXE

O HTML 5 trouxe uma sintaxe de marcação que permite uma compatibilidade com os modelos HTML e XHTML.

Portanto, podemos escrever o código de marcação com as seguintes variações:

Modelo HTML

```

```

Modelo XHTML

```

```

Modelo HTML 5

```
  
ou  

```

Como boa prática de programação, adote apenas uma sintaxe como modelo de marcação.

STYLE

É o elemento "container" para os códigos de estilização CSS. Sua sintaxe de marcação foi simplificada na versão HTML 5, não havendo mais a necessidade do atributo *type* como definição do tipo de linguagem utilizada.

Exemplo:

```
...  
<style>  
    p {font: normal 13px Arial;}  
</style>  
...
```

SCRIPT

Outro elemento "container" que também sofreu uma simplificação em sua sintaxe foi o SCRIPT.

Foram retirados os atributos *type* e *language* como obrigatórios para definição do tipo de linguagem do "container".

Exemplo:

```
...  
<script>  
    alert("Olá Mundo!")  
</script>  
...
```

A partir da versão do HTML 5, a identificação do valor do atributo *type* fica sob a responsabilidade do agente (browser).

Tipos de Conteúdo

O HTML 5 trouxe uma nova proposta de marcação semântica de código, dividida em sete grupos diferentes de conteúdo.

Grupo 1 – Conteúdo Embutido

Todo o conteúdo que é importado de outras fontes para dentro do documento.

Exemplo: Arquivos de Áudio, Árquivos de Vídeo, Imagens, iFrame.

Grupo 2 – Conteúdo de Fluxo

São os elementos inseridos no corpo do documento através do elemento *body*.

Exemplo: Cabeçalhos tipo h1, formulário, span, div entre outros.

Grupo 3 – Conteúdo de Cabeçalhos

São os elementos que definem cabeçalhos para as seções do documento.

Exemplo: h1, h2..h6, hgroup.

Grupo 4 – Conteúdo Interativo

São os elementos que produzem interação com o usuário.

Exemplo: Textarea, buttons, input entre outros.

Grupo 5 – Conteúdo de Metadado

Metadados são elementos que servem para configuração de características e comportamentos, e são localizados na seção *head* do documento HTML.

Exemplo: Charset, Script, Style, Keywords entre outros.

Grupo 6 – Conteúdo de Expressão

São os elementos de texto e marcação de texto no documento.

Exemplo: Span, cite, abbr, link, em entre outros.

Grupo 7 – Conteúdo de Seção

São os elementos que definem uma seção específica do documento.

Exemplo: Section, Article, Header, Footer, Aside, Nav.

Novos Elementos de Marcação

Um dos principais pontos de mudança na versão HTML 5 foi seu núcleo de marcação, que trouxe novos elementos mais semânticos quanto as suas definições.

Com a tendência mundial de uso cada vez maior de dispositivos móveis, fazer a informação chegar ao usuário é necessário romper barreiras de acessibilidade e disponibilidade, e para isso é preciso uma marcação que permita a interpretação da maioria dos dispositivos existentes.

Enquanto essa apostila era escrita, a Adobe anunciava o fim de investimentos no aplicativo Flash para o segmento de *mobile*.

Dessa forma, o html 5 já nasce "vitaminado" para a Nova Onda de aplicações Web.

SECTION

O Elemento SECTION tem como foco descrever o conteúdo de um documento HTML. Podemos pensar no elemento SECTION como um capítulo de um livro, por exemplo.

Exemplo:

```
<section>
  <article>
    <p>Mauris ultricies urna nec ante imperdiet at.</p>
  </article>
</section>
```

ARTICLE

Elemento que define blocos de informações independentes dentro do documento HTML.

Exemplo:

```
<section>
  <article>
    <p>Mauris ultricies urna nec ante imperdiet at.</p>
  </article>
  <article>
    <p>Mauris ultricies urna nec ante imperdiet at.</p>
  </article>
</section>
```

HEADER

Define um bloco de marcação com cabeçalhos de formatação h1, h2 etc.

Exemplo:

```
<section>
  <header>
    <h1>Mauris ultricies urna nec ante imperdiet at.</h1>
  </header>
  <article>
    <p>Mauris ultricies urna nec ante imperdiet at.</p>
  </article>
</section>
```

FOOTER

Elemento que define um bloco de marcação com características de rodapé, seja para uma seção, artigo ou documento HTML.

Exemplo:

```
<section>
  <article>
    <p>Mauris ultricies urna nec ante imperdiet at.</p>
  </article>
  <footer>
    <p>Mauris ultricies urna nec ante imperdiet at.</p>
  </footer>
</section>
```

NAV

Elemento que define um bloco de marcação com elementos de navegação como menus, barra de navegação, "breadcrumb's" e etc.

Exemplo:

```
<section>
  <article>
    <p>Mauris ultricies urna nec ante imperdiet at.</p>
  </article>
  <nav>
    <ul>
      <li>Quem Somos</li>
      <li>Produtos</li>
      <li>Localização</li>
      <li>Contato</li>
    </ul>
  </nav>
</section>
```

ASIDE

Elemento usado para marcar um bloco de texto relacionado ao conteúdo principal de uma seção ou artigo do documento HTML. Exemplo: um banner, uma barra lateral de informações (cross-content), etc.

Exemplo:

```
<section>
  <article>
    <p>Mauris ultricies urna nec ante imperdiet at.</p>
    <aside>
      <h3>What is a Lorem Ipsum?</h3>
      <p>Mauris ultricies urna nec ante imperdiet at.</p>
    </aside>
  </article>
</section>
```

FIGURE

Esse elemento define a marcação de conteúdos do tipo imagens, ilustrações, fotos, gráficos, fluxogramas e etc.

Exemplo:

```
<figure>
  <legend>
    
  </legend>
</figure>
```

MARK

Esse elemento marca uma palavra ou trecho de um conteúdo de texto que deva ser destaque, ou servir de referência.

Exemplo:

```
<p><mark>HTML 5</mark>é a nova proposta de marcação para a Web mais semântica.</p>
```

Capítulo 03: Formulário HTML 5

* Visão Geral.....	14
* Suporte dos Browsers	14
* Novos Elementos de Formulário	14
* Novos Atributos de Formulário.....	26

Visão Geral

Quando descrevemos uma aplicação web é impossível não falarmos de uma rotina que envolva o elemento formulário. É o elemento do documento HTML que permite maior interatividade na troca de informações entre o usuário e a aplicação web.

E por esta razão, o HTML 5 reformulou um conjuntos de elementos e atributos para o elemento *input*, que permitem otimizar o trabalho de construção de formulários de agora em diante.

Suporte dos Browsers

O suporte atual para os elementos de formulário HTML 5 ainda não é total, mas digamos que se caminha para isso, pois não há como voltar aos métodos antigos com uma proposta de web mais semântica.

O browser que melhor trabalho de renderização fez com os novos elementos de formulário HTML 5 visualmente é o Opera 10. Em seguida vem o pessoal do Chrome e do Safari.

Os outros (Firefox e IE) aceitam a marcação dos elementos, mas não os renderizam corretamente em termos visuais, o que prejudica algumas funcionalidades dos mesmos.

Um exemplo é o elemento *color*, que mostra visualmente uma paleta de cores quando renderizado pelo Opera 10.

Novos Elementos de Formulário

A maior parte das inovações em termos de funcionalidades para o elemento formulário se deu através do campo *input*, como veremos a seguir:

tel

O tipo *tel* prepara o campo input para receber um valor que represente um número de telefone no seu formato mais genérico. Ele não faz qualquer tipo de validação para os dados de entrada, sendo essa responsabilidade da programação *Javascript* aplicada ao elemento.

Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <form>
    <fieldset>
      <legend>Novos Elementos de Formulário</legend>
      <label for='tel'>Digite seu telefone:</label><br>
      <input type='tel' name='tel' id='tel'>
      <span>Ex.: (xx) xxxx-xxxx</span><br><br>
      <input id='btn' type="submit" value=" [ enviar ] ">
    </fieldset>
  </form>
</section>
...
```

STYLE

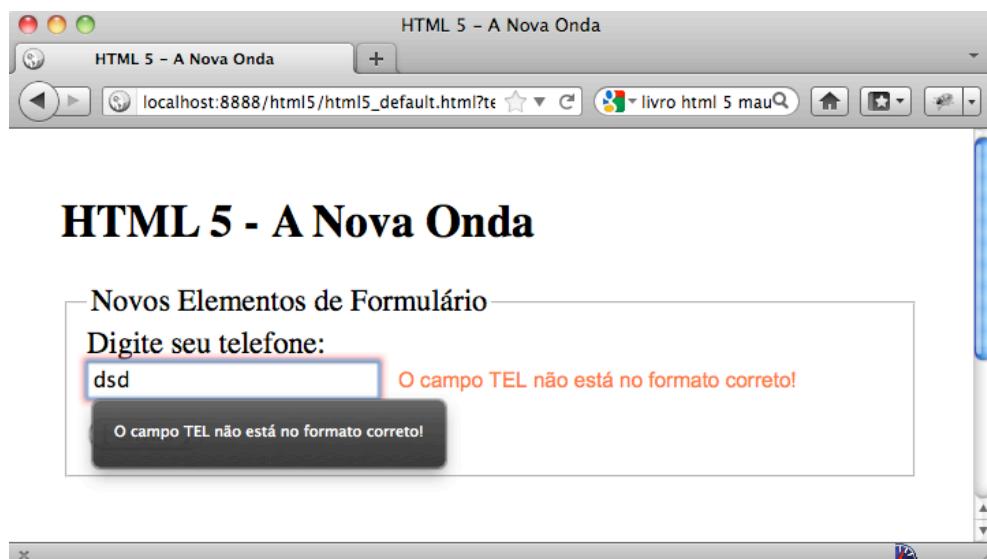
```
...
#tudo {
    width:450px;
    height:300px;
    margin: 0 auto;
    padding: 10px;
}
span{
    font: normal 11px Arial;
    color: #F63;
    margin-left: 10px;
}
input[type='submit']{
    cursor: pointer;
    padding: 1px;
    margin-top: 10px;
}
...
...
```

SCRIPT

```
...
window.onload = function(){
    var expVal = /^[0-9]{2}\)[0-9]{4}-[0-9]{4}$/;
    var campo = document.querySelector('#tel');
    var span = document.getElementsByTagName('span')[0];
    campo.addEventListener('blur', validaFone, false);

    function validaFone(){
        if(!expVal.test(this.value)){
            this.setCustomValidity('O campo '+this.name.toUpperCase()+' não
está no formato correto!');
        }
        else{
            this.setCustomValidity('');
        }
        span.innerHTML = this.validationMessage;
    };
};

...
...
```



search

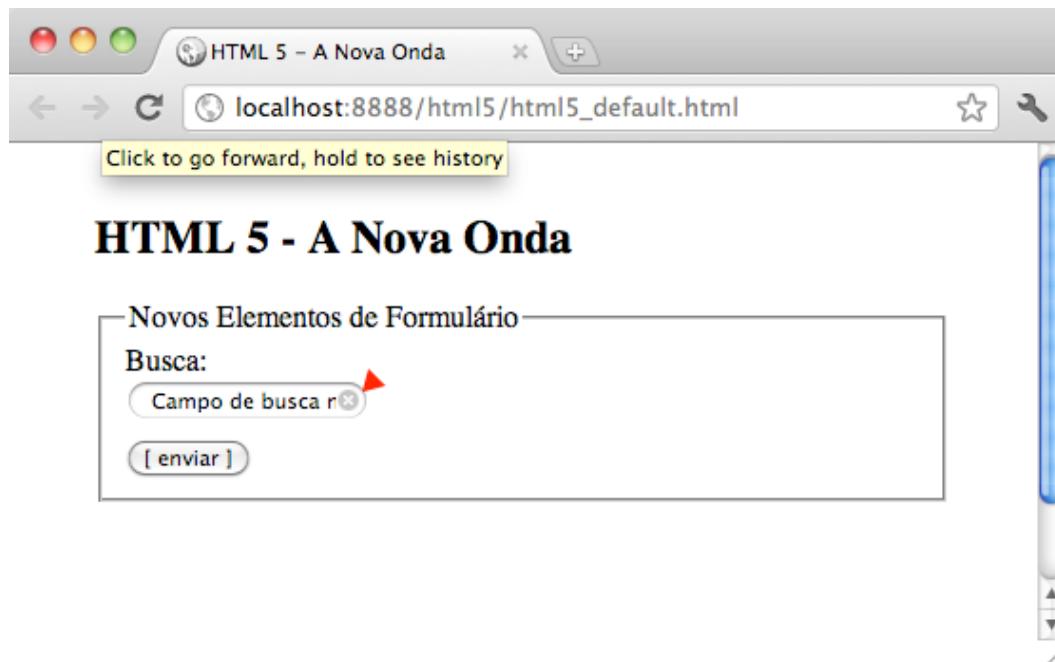
O elemento *search* modifica o comportamento do campo *input* para um tipo busca, alterando seu padrão visual para diferenciá-lo dos outros campos. Esse padrão é controlado pelo navegador utilizado.

Como exemplo, o Google Chrome ao renderizar esse elemento, coloca uma letra "X" no canto direito da área de digitação. O objetivo é fazer desaparecer o texto de marcação (*placeholder*) colocado no atributo *value* do campo *input*, no momento do foco.

Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <form>
    <fieldset>
      <legend>Novos Elementos de Formulário</legend>
      <label for='busca'>Busca:</label><br>
      <input type='search' name='busca' id='busca'><br><br>
      <input id='btn' type="submit" value="[ enviar ]">
    </fieldset>
  </form>
</section>
...
```



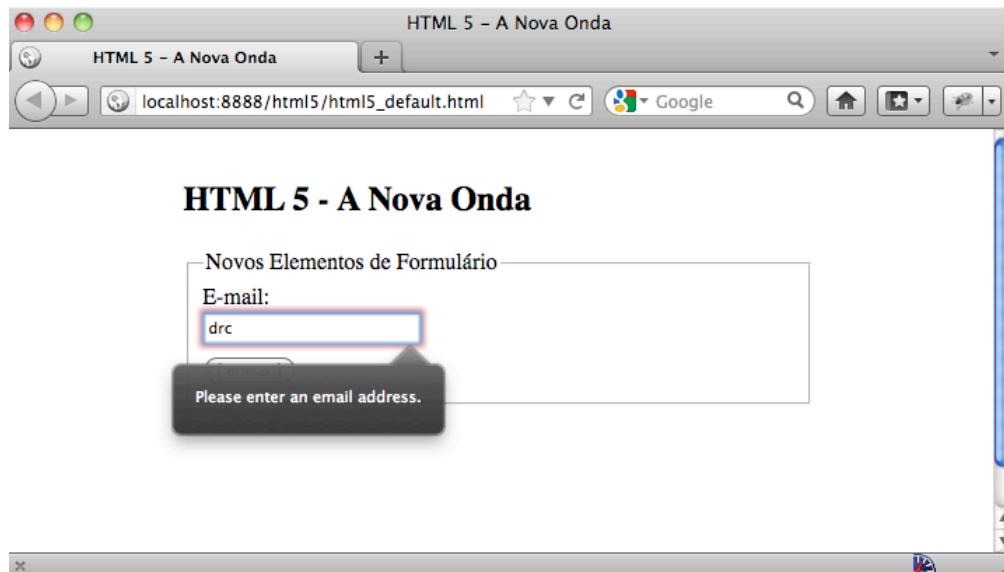
email

Esse elemento trouxe uma configuração básica para a identificação do conteúdo como um endereço de *e-mail*. O campo não valida o e-mail digitado, sendo essa rotina de atribuição da programação *Javascript*. Ele apenas procura um identificador, no caso o "@" como parte do conteúdo digitado.

Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <form>
    <fieldset>
      <legend>Novos Elementos de Formulário</legend>
      <label for='email'>E-mail:</label><br>
      <input type='email' name='email' id='email'><br>
      <input id='btn' type="submit" value="[ enviar ]">
    </fieldset>
  </form>
</section>
...
```



Como dito anteriormente, para promover uma validação mais consistente do conteúdo do campo *e-mail* use a programação *Javascript*, em conjunto com a função *setCustomValidity*.

Dessa maneira podemos ampliar a funcionalidade do elemento mantendo o padrão web de desenvolvimento não obstrutivo.

url

Esse tipo de elemento define o campo *input* para receber um URL absoluto, validando o conteúdo com base no protocolo *http*.

Exemplo:

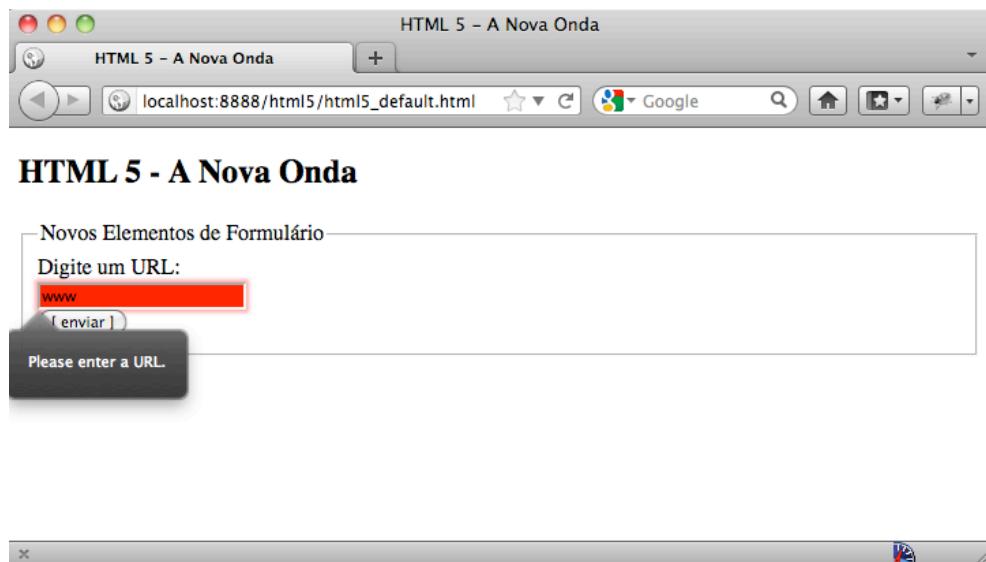
HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <form>
    <fieldset>
      <legend>Novos Elementos de Formulário</legend>
      <label for='url'>Digite um URL:</label><br>
      <input type='url' name='url' id='url' class='valida'><br><br>
      <input id='btn' type="submit" value="[ enviar ]">
    </fieldset>
  </form>
</section>
...
```

Para ver o efeito visual da validação foi usado a pseudo-classe CSS3 *valid*.

STYLE

```
...
.valida:valid {
  background-color: #0F0;
  color: #000;
}
.valida:invalid{
  background-color: #F00;
  color: #F63;
}
...
```



date

O campo do tipo *date* está preparado para receber uma *string* que representa uma data válida. A renderização do aspecto visual depende do navegador usado, e enquanto essa apostila era escrita o único que permite visualizar um calendário padrão é o Opera 10.

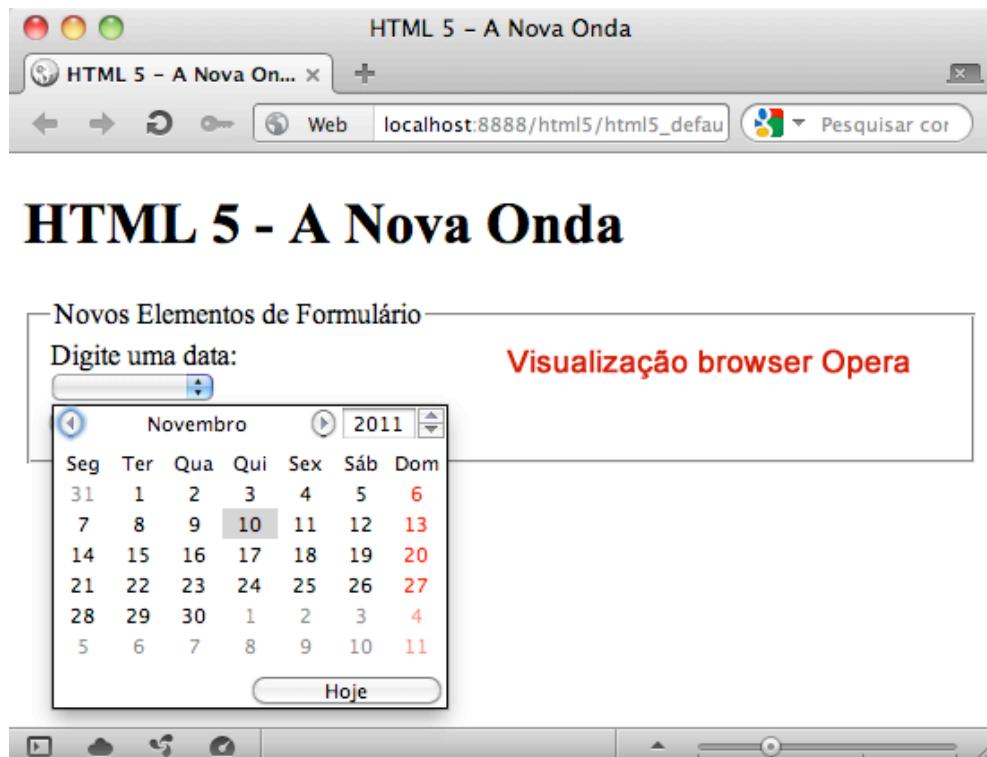
O Google Chrome mostrará esse elemento com o formato de campo tipo *step by step*.

Os outros mostram um campo do tipo *text* padrão.

Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <form>
    <fieldset>
      <legend>Novos Elementos de Formulário</legend>
      <label for='data'>Digite uma data:</label><br>
      <input type='date' name='data' id='data'><br><br>
      <input id='btn' type="submit" value="[ enviar ]">
    </fieldset>
  </form>
</section>
...
```



time

O campo do tipo *time* está preparado para receber como conteúdo uma *string* que representa uma hora válida.

Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <form>
    <fieldset>
      <legend>Novos Elementos de Formulário</legend>
      <label for='hora'>Digite uma hora:</label><br>
      <input type='time' name='hora' id='hora'><br>
    </fieldset>
  </form>
</section>
...
```

Novamente, a renderização visual desse elemento depende do tipo de browser do usuário. O exemplo acima foi renderizado no Opera 10, conforme a imagem a seguir:



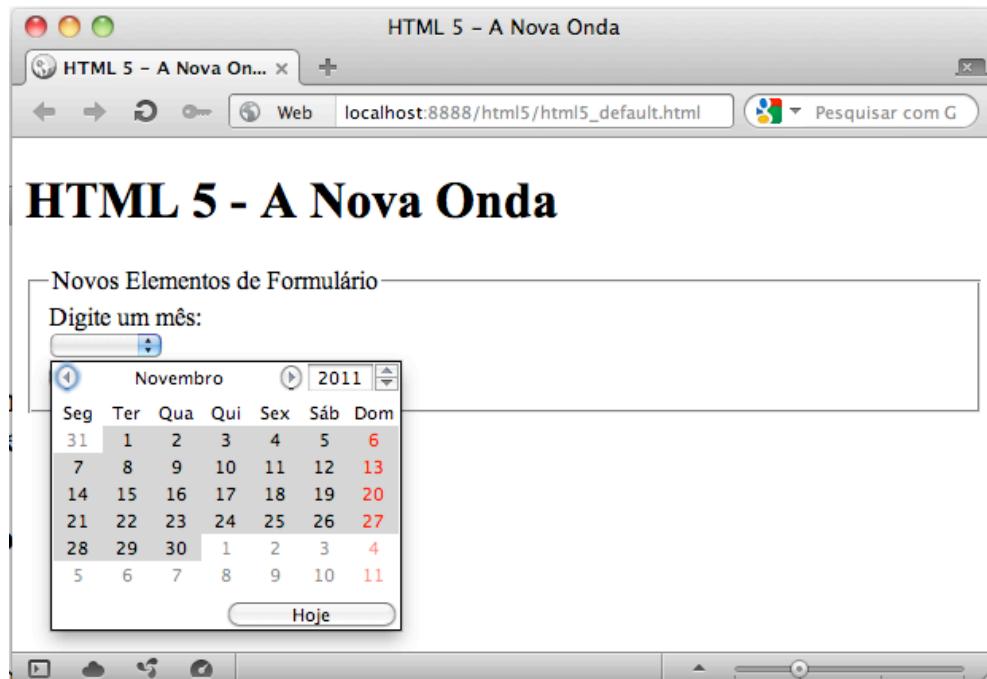
month

O elemento *month* define um mês do ano baseado no calendário Gregoriano. Sua renderização visual no browser Opera 10 é um objeto do tipo *calendar*.

Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <form>
    <fieldset>
      <legend>Novos Elementos de Formulário</legend>
      <label for='mes'>Digite um mês:</label><br>
      <input type='month' name='mes' id='mes'>
    </fieldset>
  </form>
</section>
...
```



A renderização visual do elemento do tipo *month* no navegador Google Chrome é do tipo *step by step*.

Nos outros navegadores a renderização visual desse elemento para o campo *input* é do tipo *text*.

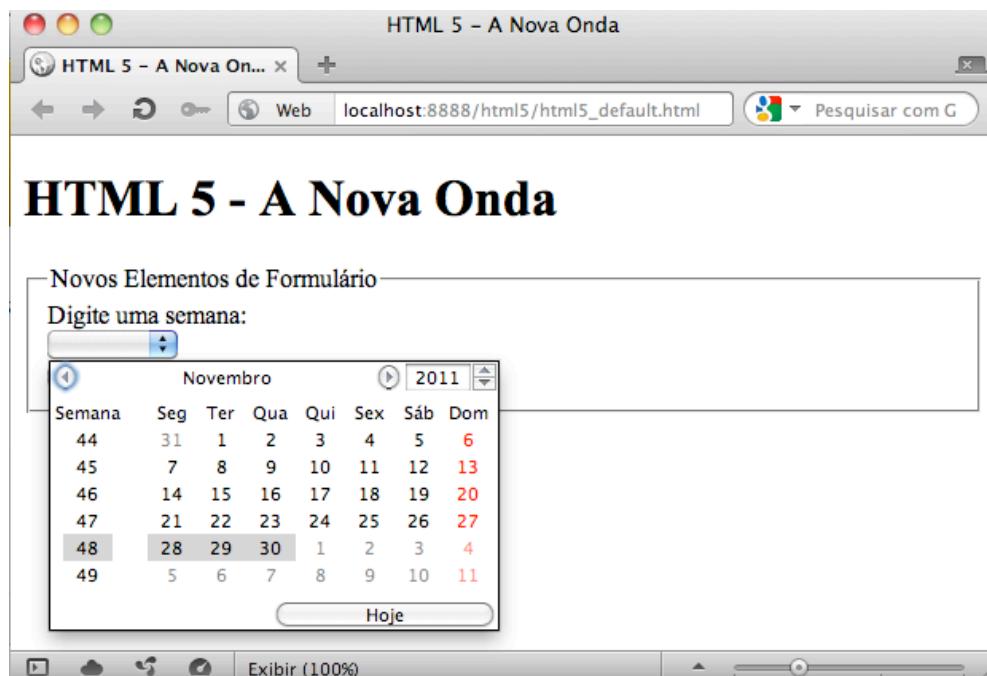
week

O elemento *week* define um número da semana do ano baseado no calendário Gregoriano. Sua renderização visual no browser Opera 10 é um objeto do tipo *calendar*.

Exemplo:

HTML

```
...
<section>
    <h1>HTML 5 - A Nova Onda</h1>
    <form>
        <fieldset>
            <legend>Novos Elementos de Formulário</legend>
            <label for='semana'>Digite uma semana:</label><br>
            <input type='week' name='semana' id='semana'>
        </fieldset>
    </form>
</section>
...
```



A renderização visual do elemento do tipo month no navegador Google Chrome é do tipo *step by step*.

Nos outros navegadores a renderização visual desse elemento para o campo *input* é do tipo *text*.

number

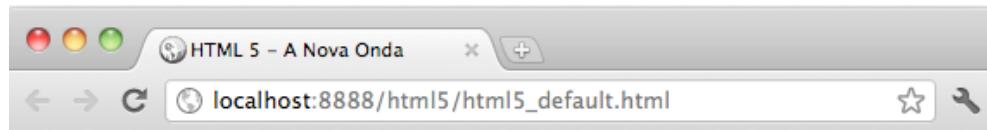
O elemento *number* define o elemento *input* para receber um valor numérico.

Em geral, sua renderização no navegador do usuário é do tipo *slider* com um conjunto de setas direcionais para cima e para baixo, permitindo o incremento do valor numérico do conteúdo.

Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <form>
    <fieldset>
      <legend>Novos Elementos de Formulário</legend>
      <label for='nr'>Digite um valor numérico:</label><br>
      <input type='number' name='nr' id='nr'>
    </fieldset>
  </form>
</section>
...
```



HTML 5 - A Nova Onda

Novos Elementos de Formulário

Digite um valor numérico:

[enviar]

No navegador Firefox a renderização visual desse elemento para o campo *input* é do tipo *text*.

range

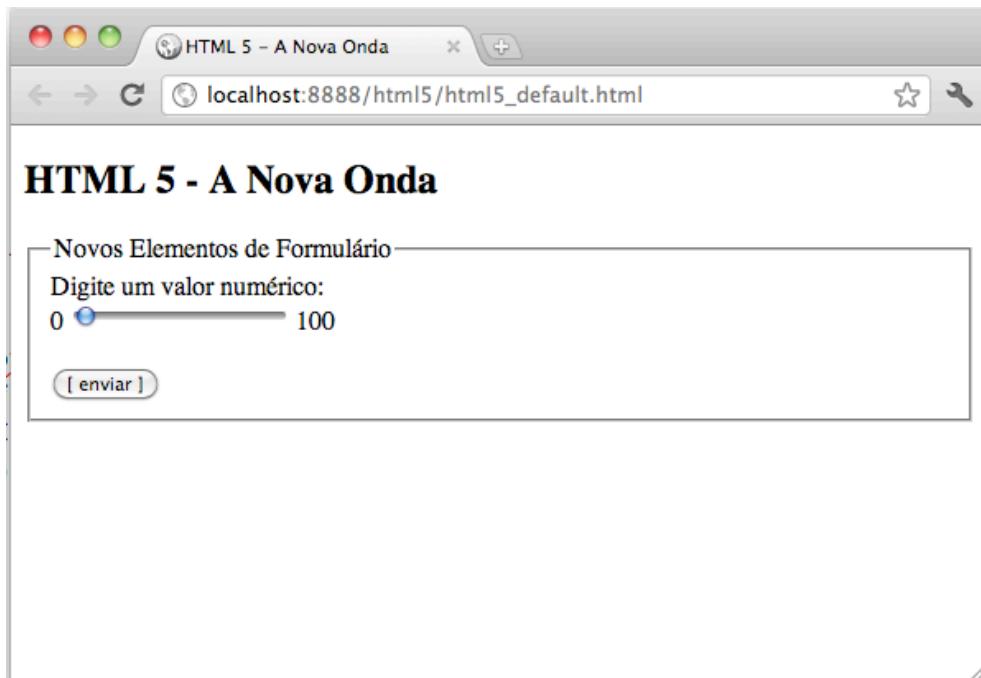
O elemento *range* prepara o campo *input* para receber um valor numérico, dentro de um intervalo específico.

Esse intervalo é constituído de um valor mínimo, um valor máximo, um valor de incremento e um valor padrão.

Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <form>
    <fieldset>
      <legend>Novos Elementos de Formulário</legend>
      <label for='nr'>Digite um valor numérico:</label><br>
      0<br><input type='range' name='nr' id='nr' min='0' max='100'
      step='5' value='0'>100
    </fieldset>
  </form>
</section>
...
```



A renderização visual desse elemento adquire o formato de um *slider* nos browsers Google Chrome, Safari e Opera.

Nos outros browsers o elemento *range* é renderizado como um campo do tipo *text*.

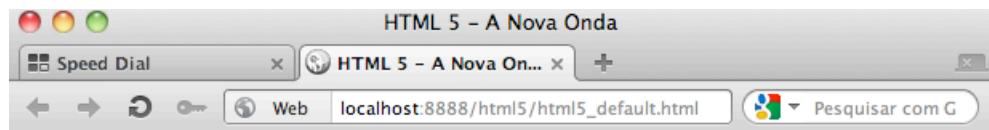
color

O elemento *color* define para o campo *input* uma paleta de cores RGB.

Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <form>
    <fieldset>
      <legend>Novos Elementos de Formulário</legend>
      <label for='cor'>Escolha uma cor:</label><br>
      <input type='color' name='cor' id='cor'>
    </fieldset>
  </form>
</section>
...
```



HTML 5 - A Nova Onda



A renderização visual desse elemento adquire o formato de um *color-picker* no Opera. Nos outros browsers o elemento *color* é renderizado como um campo do tipo *text*.

Novos Atributos de Formulário

Um novo conjunto de atributos para o elemento formulário veio com a versão HTML 5, buscando otimizar as ações existentes em versões anteriores.

Esses atributos não alteram a forma anterior de trabalho do container *form*, pelo contrário, ampliam e flexibilizam o processo de interação com o usuário final.

formaction

Esse atributo possui funcionalidade similar ao atributo *action*, definindo um URL para envio das informações a serem processadas pelo formulário.

No caso do atributo *formaction*, o mesmo pode ser aplicado para os elementos *button* e *input*, enquanto o atributo *action* era declarado somente no cabeçalho do container *form*.

Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <form>
    <fieldset>
      <legend>Novos Atributos de Formulário</legend>
      <p>
        <label for='email'>Digite seu e-mail:</label><br>
        <input type='email' name='email' id='email'>
      </p>
      <p>
        <input type='submit' value='Enviar'
          formaction='newsletter.php'

```

formenctype

Esse atributo possui funcionalidade similar ao atributo *enctype*, definindo um tipo de conteúdo (MIME – Extensões Multi função para Mensagens de Internet) a ser enviado na postagem do formulário. Por exemplo, se for usado o campo *input* do tipo *file* para enviar arquivos de imagem via *form*, temos que informar essa configuração pelo *enctype*.

Esse atributo pode ser aplicado para os elementos *button* e *input*, enquanto o atributo *enctype* era declarado somente no cabeçalho do container *form*.

Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <form>
    <fieldset>
      <legend>Novos Atributos de Formulário</legend>
      <p>
        <label for='imagem'>Envie sua foto:</label><br>
        <input type='file' name='imagem'>
      </p>
      <p>
        <input type='submit' value='Enviar'
          formenctype='image/jpg'>
      </p>
    </fieldset>
  </form>
</section>
...
```

formmethod

Esse atributo possui a mesma função do atributo *method*, definindo um método de envio das informações do formulário através de um protocolo: *GET* ou *POST*.

Esse atributo pode ser aplicado para os elementos *button* e *input*, enquanto o atributo *enctype* era declarado somente no cabeçalho do container *form*.

Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <form>
    <fieldset>
      <legend>Novos Atributos de Formulário</legend>
      <p>
        <label for='email'>Digite seu e-mail:</label><br>
        <input type='email' name='email' id='email'>
      </p>
      <p>
        <input type='submit' value='Enviar'
          formmethod='post'>
      </p>
    </fieldset>
  </form>
</section>
...
```

formtarget

Esse atributo define o alvo de destino para as informações enviadas pelo formulário. Sua funcionalidade é similar ao do atributo *target* usado para definir container de carregamento. Seus parâmetros possíveis são: *_blank*, *_top*, *_self* e *_parent* ou um nome específico.

Esse atributo pode ser aplicado para os elementos *button* e *input*, enquanto o atributo *enctype* era declarado somente no cabeçalho do container *form*.

Exemplo:

HTML

```
...
<section>
    <h1>HTML 5 - A Nova Onda</h1>
    <form>
        <fieldset>
            <legend>Novos Atributos de Formulário</legend>
            <p>
                <label for='email'>Digite seu e-mail:</label><br>
                <input type='email' name='email' id='email'>
            </p>
            <p>
                <input type='submit' value='Enviar'
                    formtarget='_blank'>
            </p>
        </fieldset>
    </form>
</section>
...
```

form

Ao contrário do elemento container *form*, o atributo *form* tem como função associar um elemento de formulário ao seu controle específico. Esse atributo permite que associar controles inseridos fora dos limites da tag *form* ou dentro de outros formulários. Para efetuar a associação, o valor do atributo *form* deve ser o mesmo do *id* do formulário a ser associado.

Exemplo:

HTML

```
...
<section>
    <h1>HTML 5 - A Nova Onda</h1>
    <form id='frm'>
        <fieldset>
            <legend>Novos Atributos de Formulário</legend>
            <p>
                <label for='email'>Digite seu e-mail:</label><br>
                <input type='email' name='email' id='email'>
            </p>
            <p>
                <input type='submit' value='Enviar'
                    formtarget='_blank'>
            </p>
        </fieldset>
    </form>
    <p><label>Nome</label><input type='text' name='nome'
        form='frm'></p>
</section>
...
```

Capítulo 04: API de Áudio

* Visão Geral.....	30
* Suporte dos Browsers	30
* Tipos de Arquivos de Áudio	30
* Tipos de CODECS de Áudio.....	31
* Elemento Áudio.....	32
* Elemento Source.....	33
* Atributos do Elemento Áudio.....	33
* Métodos do Elemento Áudio.....	37

Visão Geral

Em tempos de internet 2.0, os elementos áudio-visuais ganharam relevância na transmissão de informações ao usuário final de uma aplicação web.

Hoje em dia a maioria dos dispositivos que acessam a internet tem que estar preparados para executar elementos de *media*, seja de áudio, vídeo ou os dois.

O HTML 5 traz uma API (*Application Programming Interface*) que possibilita a integração e execução de arquivos de áudio com simplicidade, evitando os famosos *plug-ins* inseridos nos agentes de usuário (navegadores) para correta interpretação dos mesmos.

Suporte dos Browsers

Hoje, a maior parte dos navegadores executa um arquivo de áudio de acordo com seu motor de renderização (*engine*). Assim sendo, podemos tocar arquivos de áudio usando a API do HTML 5 e especificando o tipo de áudio para os principais navegadores usados.

Espera-se daqui para frente, que o W3C busque uma padronização para tornar o desenvolvimento de aplicações com API de Áudio/Vídeo realmente simples e desburocratizadas. Para um novo começo, a atual API já nos dá um grande alívio em termos de compatibilidade.

Tipos de Arquivos de Áudio

Um arquivo de áudio nada mais é que um repositório (*container*) de informações sobre seu conteúdo. Em sua estrutura interna encontram-se informações sobre as trilhas de áudio, e metadados como título, autor, nome da música, tempo e etc.

Veja a figura abaixo sobre um *container* de áudio:



Para manipularmos um *container* de áudio usamos a linguagem de programação *Javascript*, em conjunto com as propriedades e métodos da API.

Os tipos mais populares de *containers* de áudio são:

- AAC (.aac)
- MPEG 3 (.mp3)
- MIDI (.mid)
- Ogg (.ogg)
- Windows Media Player (.wma)

Para trabalhar com a API de áudio do HTML 5 usaremos os containers mp3, wma e ogg.

Tipos de CODECS de Áudio

Quando o assunto é áudio, alguns termos precisam ser conhecidos para se obter uma melhor performance na construção de aplicações web que utilizem esse recurso.

Um deles é o termo CODEC (Codificador/Decodificador) que significa um dispositivo de *hardware* ou um elemento de *software*, capaz de comprimir um arquivo de áudio a uma determinada taxa de *bits*. A compressão de um arquivo de áudio se torna necessária, principalmente para ambientes *web*, devido ao peso (Kb) que esse recurso representa no contexto da aplicação *on-line*.

Um arquivo de áudio com uma boa taxa de compressão tende a ser carregado mais rapidamente, proporcionando ao usuário final uma experiência mais rápida e otimizada com sons.

Os tipos de CODECS utilizados na API de Áudio do HTML 5 são:

- WMA
- Ogg Vorbis
- MP3

Usando a referência de qualidade *versus* taxa de compressão, os melhores CODECS de áudio para trabalho em ambiente *web* são o Ogg Vorbis e o MP3.

Um fator que ainda não está resolvido com relação a API de áudio HTML 5 é qual arquivo/*codec* será usado como padrão. Como o MP3 é um padrão proprietário, dependendo de *royalties* para seu uso, o grupo de trabalho do W3C busca uma alternativa *open source* com o uso do Ogg Vorbis.

Porém, ainda não há um consenso sobre um padrão, e enquanto isso, temos que servir em uma aplicação web formatos alternativos para que o agente de usuário (navegador) possa tocar arquivos de áudio sem problemas.

Desse maneira, no trabalho com a API de áudio HTML 5 usaremos os formatos:

- Ogg
- MP3
- WMA

Elemento Áudio

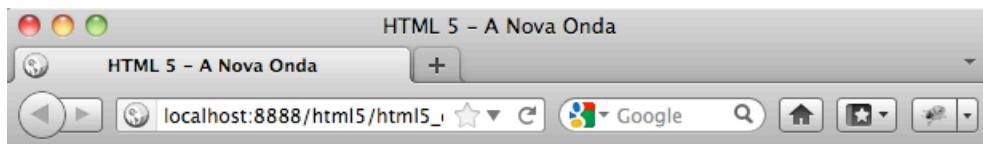
Para poder utilizar a API de áudio do HTML 5 temos o elemento de marcação <audio>, que possui todas as propriedades e métodos necessários para manipulação da estrutura de som.

Esse recursos são manipuláveis através da linguagem *Javascript*.

Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <audio controls="controls"></audio>
</section>
...
```



HTML 5 - A Nova Onda



Com a marcação acima criamos um container de áudio que permite a recepção de arquivos de som, com recursos de uma barra de controle de propriedades com o atributo *controls*. Simples assim!

Não há mais a necessidade de *plug-ins* incorporados na página *web*, para que o arquivo de som seja interpretado corretamente. Também não há necessidade de construção dos famosos *players* de controle de propriedades de som, geralmente, feitos em *Flash*.

Devido a essas características de simplicidade, a API de áudio do HTML 5 se mostrou a alternativa para incorporação de elementos de som em dispositivos móveis, de maneira não obstrutiva, o que levou o aplicativo *Flash* a ser descartado como opção para esse processo.

Até o momento em que esse capítulo estava sendo escrito, a ADOBE divulgou nota oficial comunicando o fim de investimentos para adaptação do *Flash* a área de *mobile*.

Elemento Source

Como dito no início do capítulo, a falta de padronização de um formato de áudio traz a necessidade de usar opções para que cada agente de usuário (navegador) possa tocar o arquivo de som proposto. Com isso, foi criado na API um elemento filho da marcação `<audio>` que é o `<source>`.

Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <audio controls="controls">
    <source src="som.ogg"></source>
    <source src="som.mp3"></source>
    <source src="som.wma"></source>
  </audio>
</section>
...
```

Veja que com o acréscimo do elemento `<source>` podemos oferecer formatos alternativos de *containers* de som, permitindo que o agente de usuário (navegador) identifique em sua *engine* qual *container* usar.

Atributos do Elemento Áudio

Os elementos `<audio>` e `<source>` possuem um conjunto de atributos que facilitam a manipulação e controle dos recursos de áudio.

autoplay

Esse atributo permite que o arquivo de som seja carregado e sua reprodução seja imediata. É especificado no elemento `<audio>` da API.

Como padrão de marcação, colocamos em seu valor o mesmo nome do atributo representando o *booleano* de confirmação do recurso que é *true*. Também é permitido marcar somente o atributo sem o seu valor. O agente de usuário interpreta o valor *booleano* da mesma maneira.

Porém, por questões de boas práticas de programação usamos o atributo e seu valor.

Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <audio autoplay="autoplay"></audio>
</section>
...
```

controls

Esse atributo permite que o arquivo de som seja carregado e junto com ele uma barra de controles para manipulação de suas propriedades básicas.

É especificado no elemento <audio> da API.

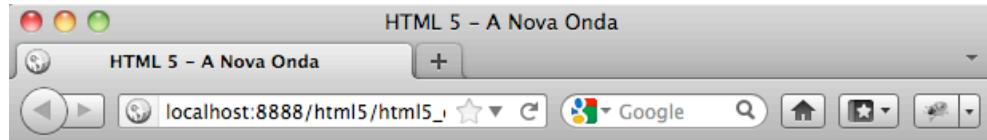
Como padrão de marcação, colocamos em seu valor o mesmo nome do atributo representando o *booleano* de confirmação do recurso que é *true*. Também é permitido marcar somente o atributo sem o seu valor. O agente de usuário interpreta o valor *booleano* da mesma maneira.

Porém, por questões de boas práticas de programação usamos o atributo e seu valor.

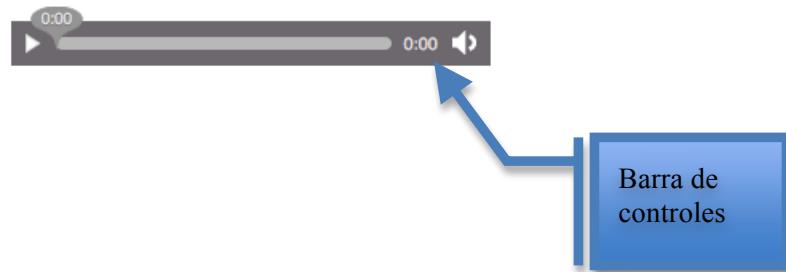
Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <audio controls="controls"></audio>
</section>
...
```



HTML 5 - A Nova Onda



Obs.: Se for omitido o atributo controls do elemento <audio>, o arquivo de som tocará normalmente, porém o usuário não terá controle sobre os recursos básicos do som como parar, retroceder, avançar e etc.

Recomenda-se por questões de acessibilidade e boas práticas, sempre fornecer ao usuário opções de controle dos recursos de áudio, sejam eles nativos da API ou via *Javascript*.

src

O atributo *src* permite definir a origem e/ou caminho do recurso de áudio que está sendo carregado.

É especificado no elemento <source> da API.

Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <audio>
    <source src="som.ogg">
    <source src="som.mp3">
    <source src="som.wma">
  </audio>
</section>
...
ou
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <audio>
    <source src="http://www.exemplo.com.br/sons/som.ogg">
    <source src="http://www.exemplo.com.br/sons/som.mp3">
    <source src="http://www.exemplo.com.br/sons/som.wma">
  </audio>
</section>
...
```

type

O atributo *type* define ao agente de usuário (navegador) o tipo de recurso que será incorporada ao documento, realizado através de um *MIME Type* (Extensão Multi-função para Mensagens de Internet).

É especificado no elemento <source> da API.

Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <audio>
    <source src="som.ogg" type="audio/ogg">
  </audio>
</section>
...
```

OBS.: A marcação do atributo *type* é importante, pois, ele define qual o *container* de áudio o agente de usuário (navegador) deve utilizar para carregar o recurso.

codec

O atributo *codec* define para o agente de usuário (navegador) como o arquivo de áudio carregado foi codificado, e com quais parâmetros. Enquanto não houver uma padronização sobre o formato de áudio utilizado na API, é recomendado que se defina o *codec* do arquivo carregado.

É especificado no elemento <**source**> da API.

Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <audio>
    <source src="som.ogg" type="audio/ogg" codec="vorbis"

```

loop

Como o próprio nome sugere, esse atributo permite uma reprodução contínua do arquivo de som após seu término.

É especificado no elemento <**audio**> da API.

Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <audio autoplay="autoplay" controls="controls" loop="loop"

```

Para uma prática de acessibilidade e padronização web, é importante oferecer ao usuário alternativas as novas características da linguagem HTML 5.

Uma forma bem simples, no caso da API de áudio, é disponibilizar um *link de download* dos arquivos de áudio, caso o agente de usuário (navegador) não esteja preparado para as novas características HTML 5.

Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <audio autoplay="autoplay" controls="controls" loop="loop"

```

Métodos do Elemento Áudio

A API de áudio do HTML 5 possui alguns métodos para controle de funções via *Javascript*. O controle de funções via linguagem de programação oferece ao desenvolvedor uma maneira de personalizar a entrega de áudio ao usuário final, porém, não se pode esquecer que o mesmo tem o controle sobre o *browser*, podendo desabilitar o *Javascript* no momento que desejar.

Por isso, a melhor opção é oferecer uma alternativa para o controle via programação com elementos de marcação HTML.

play

Esse método controla o processo de execução do arquivo de som.

Exemplo:

```
...
  som.play();
...

```

volume

Esse método controla o volume do arquivo de som tocado. Seus parâmetros são 0 para o volume mais baixo, e 1 para o volume mais alto do som. Para manipular a faixa intermediária de volume do som deve-se usar frações do inteiro 1.

Exemplo:

```
...
  som.volume += 0.1;
...

```

pause

No conjunto de métodos da API de áudio não existe o método *stop*. O processo de parada do som tocado é realizado pelo método *pause*.

Exemplo:

```
...
  som.pause();
...

```

canPlayType

Esse método permite indentificar se o agente de usuário (navegador), está preparado para receber as características da API de áudio HTML 5.

Exemplo:

```
...
  if(som.canPlayType) {
    código...
  }
  else{
    código...
  }
...

```

Abaixo segue um modelo de código de controle da API de áudio do HTML 5 com *Javascript*.

Exemplo:

HTML

```
...
<section>
    <h1>HTML 5 - A Nova Onda</h1>
    <audio id="trilha">
        <source src="som.ogg">
        <source src="som.mp3">
        <source src="som.wma">
        <a href="trilha.zip">Baixe os arquivos de som</a>
    </audio>
    <div id="botoes">
        <button>play</button>&nbsp;&nbsp;&nbsp;
        <button>pause</button>&nbsp;&nbsp;&nbsp;
        <button>volume [+]</button>&nbsp;&nbsp;&nbsp;
        <button>volume [-]</button>&nbsp;&nbsp;&nbsp;
    </div>
</section>
...

```

Javascript

```
...
<script>
    window.onload = function(){
        var bt = document.getElementsByTagName('button');
        var trilha = document.querySelector('#trilha');
        bt[0].addEventListener('click',tocaSom,false);
        bt[1].addEventListener('click',pausaSom,false);
        bt[2].addEventListener('click',volumeMais,false);
        bt[3].addEventListener('click',volumeMenos,false);

        function tocaSom(){
            trilha.volume = 1;
            trilha.play();
        }

        function pausaSom(){
            trilha.pause();
        }

        function volumeMais(){
            trilha.play();
            trilha.volume += 0.1;
        }

        function volumeMenos(){
            trilha.play();
            trilha.volume -= 0.1;
        }
    }; //Função global
</script>
...

```

Capítulo 05: API de Vídeo

* Visão Geral.....	40
* Suporte dos Browsers.....	40
* Tipos de Arquivos de Vídeo.....	40
* Tipos de CODECS de Vídeo.....	41
* Elemento Vídeo.....	41
* Elemento Source	42
* Atributos do Elemento Vídeo.....	43
* Métodos do Elemento Vídeo.....	47

Visão Geral

Nos dias de hoje é quase impossível que nenhum usuário de *internet* não tenha assistido um vídeo na *web*. A maioria dos sites possui algum arquivo de vídeo em seu conteúdo, e a tendência é aumentar esse número cada vez mais.

Está ai o *Youtube* para não deixar dúvidas!

O HTML 5 traz uma API (*Application Programming Interface*) que possibilita a integração e execução de arquivos de vídeo com simplicidade, evitando os famosos *plug-ins* inseridos nos agentes de usuário (navegadores) para correta interpretação dos mesmos.

Suporte dos Browsers

Hoje, a maior parte dos navegadores executa um arquivo de vídeo de acordo com seu motor de renderização (*engine*). Assim sendo, podemos tocar arquivos de vídeo usando a API do HTML 5 e especificando o tipo de vídeo para os principais navegadores usados.

Como dito no capítulo anterior, aguardamos uma breve padronização para que o trabalho de desenvolvimento com arquivos de áudio/vídeo seja menos burocratizado.

Tipos de Arquivos de Vídeo

Um arquivo de vídeo nada mais é que um repositório (*container*) de informações sobre seu conteúdo. Em sua estrutura interna encontram-se informações sobre as trilhas de vídeo, e metadados como título, autor, nome do vídeo, tempo e etc.

Veja a figura abaixo sobre um *container* de vídeo:



Para manipularmos um *container* de áudio usamos a linguagem de programação *Javascript*, em conjunto com as propriedades e métodos da API.

Os tipos mais populares de *containers* de vídeo são:

- Ogg (.ogv)
- MPEG 4 (.mp4 e .m4v)
- WebM (.webm)
- Flash Vídeo (.flv)

Para trabalhar com a API de vídeo do HTML 5 usaremos os containers mp4, webm e ogv.

Tipos de CODECS de Vídeo

Um arquivo de vídeo com uma boa taxa de compressão tende a ser carregado mais rapidamente, proporcionando ao usuário final uma experiência mais rápida e otimizada com imagens.

Os tipos de CODECS utilizados na API de Vídeo do HTML 5 são:

- MPEG4 para .mp4
- Theora Vorbis para .ogv
- H.264 para MPEG
- VP8 para .webm

Um fator que ainda não está resolvido com relação a API de vídeo HTML 5 é qual arquivo/*codec* será usado como padrão. Como o MPEG 4 é um padrão proprietário, dependendo de *royalties* para seu uso, o grupo de trabalho do W3C busca uma alternativa *open source* com o uso do Theora Vorbis.

Já o Google produz um forte investimento no padrão WebM, também *open source*.

Porém, ainda não há um consenso sobre um padrão, e enquanto isso, temos que servir em uma aplicação web formatos alternativos para que o agente de usuário (navegador) possa tocar arquivos de vídeo sem problemas.

Desse maneira, no trabalho com a API de vídeo HTML 5 usaremos os formatos:

- Ogv
- MP4
- WebM

Elemento Vídeo

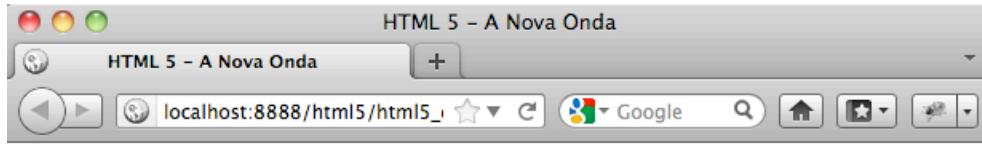
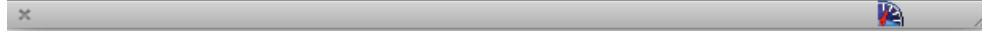
Para poder utilizar a API de vídeo do HTML 5 temos o elemento de marcação <video>, que possui todas as propriedades e métodos necessários para manipulação da estrutura de vídeo/imagens.

Esse recursos são manipuláveis através da linguagem *Javascript*.

Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <video controls="controls"></video>
</section>
...
```

**HTML 5 - A Nova Onda****Elemento Source**

A falta de padronização de um formato de vídeo traz a necessidade de usar opções para que cada agente de usuário (navegador) possa tocar o arquivo proposto. Com isso, foi criado na API um elemento filho da marcação `<video>` que é o `<source>`.

Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <video controls="controls">
    <source src="video.ogv"></source>
    <source src="video.mp4"></source>
    <source src="video.webm"></source>
  </video>
</section>
...
```

Com o acréscimo do elemento `<source>` podemos oferecer formatos alternativos de *containers* de vídeo, permitindo que o browser identifique em sua *engine* qual *container* usar.

Atributos do Elemento Vídeo

Os elementos `<video>` e `<source>` possuem um conjunto de atributos que facilitam a manipulação e controle dos recursos de vídeo.

autoplay

Esse atributo permite que o arquivo de vídeo seja carregado e sua reprodução seja imediata. É especificado no elemento `<video>` da API.

Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <video autoplay="autoplay"></video>
</section>
...
```

controls

Esse atributo permite que o arquivo de vídeo seja carregado e junto com ele uma barra de controles para manipulação de suas propriedades básicas.

É especificado no elemento `<video>` da API.

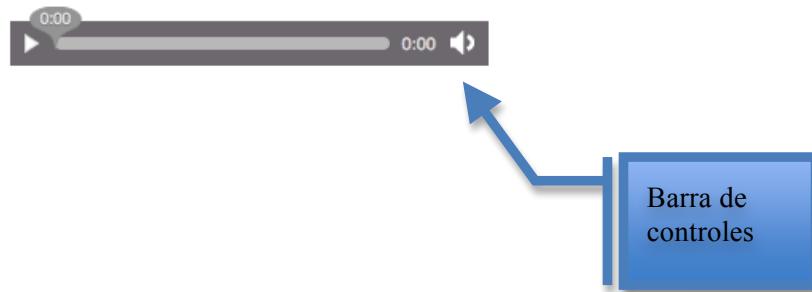
Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <video controls="controls"></video>
</section>
...
```



HTML 5 - A Nova Onda



src

O atributo *src* permite definir a origem e/ou caminho do recurso de vídeo que está sendo carregado.

É especificado no elemento <source> da API.

Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <video>
    <source src="video.ogv">
    <source src="video.mp4">
    <source src="video.webm">
  </video>
</section>
...
ou
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <video>
    <source src="http://www.exemplo.com.br/videos/video.ogv">
    <source src="http://www.exemplo.com.br/videos/video.mp4">
    <source src="http://www.exemplo.com.br/videos/video.webm">
  </video>
</section>
...

```

type

O atributo *type* define ao agente de usuário (navegador) o tipo de recurso que será incorporada ao documento, realizado através de um *MIME Type* (Extensão Multi-função para Mensagens de Internet).

É especificado no elemento <source> da API.

Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <video>
    <source src="video.ogv" type="video/ogg">
  </video>
</section>
...

```

OBS.: A marcação do atributo *type* é importante, pois, ele define qual o *container* de vídeo o agente de usuário (navegador) deve utilizar para carregar o recurso.

codec

O atributo *codec* define para o agente de usuário (navegador) como o arquivo de vídeo carregado foi codificado, e com quais parâmetros. Enquanto não houver uma padronização sobre o formato de vídeo utilizado na API, é recomendado que se defina o *codec* do arquivo carregado.

É especificado no elemento <**source**> da API.

Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <video>
    <source src="video.ogv" type="video/ogg" codec="theora">
  </video>
</section>
...
...
```

loop

Como o próprio nome sugere, esse atributo permite uma reprodução contínua do arquivo de vídeo após seu término.

É especificado no elemento <**video**> da API.

Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <video autoplay="autoplay" controls="controls" loop="loop"

```

Para uma prática de acessibilidade e padronização web, é importante oferecer ao usuário alternativas as novas características da linguagem HTML 5.

Uma forma bem simples, no caso da API de vídeo, é disponibilizar um *link de download* dos arquivos de vídeo, caso o agente de usuário (navegador) não esteja preparado para as novas características HTML 5.

Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <video autoplay="autoplay" controls="controls" loop="loop">
    <source src="video.ogv" type="video/ogg" codec="theora">
    <a href="videos.zip">Baixe os arquivos de vídeo</a>
  </video>
</section>
...
...
```

width

O atributo *width* define a largura em *pixels* do arquivo de vídeo carregado.

Exemplo:

HTML

```
...
<section>
    <h1>HTML 5 - A Nova Onda</h1>
    <video autoplay="autoplay" controls="controls" width="300">
        <source src="video.ogv" type="video/ogg" codec="theora">
        <a href="videos.zip">Baixe os arquivos de video</a>
    </video>
</section>
...
```

height

O atributo *height* define a altura em *pixels* do arquivo de vídeo carregado.

Exemplo:

HTML

```
...
<section>
    <h1>HTML 5 - A Nova Onda</h1>
    <video controls="controls" width="300" height="200">
        <source src="video.ogv" type="video/ogg" codec="theora">
        <a href="videos.zip">Baixe os arquivos de video</a>
    </video>
</section>
...
```

poster

O atributo *poster* é exclusivo do elemento *video*, e define uma imagem no lugar do arquivo de vídeo se ele não for carregado.

Exemplo:

HTML

```
...
<section>
    <h1>HTML 5 - A Nova Onda</h1>
    <video controls="controls" poster="imagem/capa.gif">
        <source src="video.ogv" type="video/ogg" codec="theora">
        <a href="videos.zip">Baixe os arquivos de video</a>
    </video>
</section>
...
```

Com o atributo *poster* é possível fazer um pré-carregamento das informações do vídeo, informando o usuário sobre o seu conteúdo.

preload

Esse atributo define que o arquivo de vídeo será carregado, quando a página do documento está sendo renderizada pelo agente do usuário (navegador). Esse atributo é ignorado caso o atributo *autoplay* esteja configurado.

O atributo *preload* admite três tipos de valores: *none*, *auto* e *metadata*.

Item	Descrição
<i>none</i>	Nenhum quadro do arquivo de vídeo é pré-carregado enquanto a página está sendo renderizada pelo browser.
<i>auto</i>	O navegador gerencia o carregamento do vídeo para uma melhor experiência do usuário.
<i>metadata</i>	O navegador deve baixar a quantidade necessária de quadros do arquivo para começar a reprodução do vídeo.

Exemplo:

HTML

```
...
<section>
    <h1>HTML 5 - A Nova Onda</h1>
    <video controls="controls" preload="auto">
        <source src="video.ogv" type="video/ogg" codec="theora">
        <a href="videos.zip">Baixe os arquivos de video</a>
    </video>
</section>
...

```

Métodos do Elemento Vídeo

A API de vídeo do HTML 5 possui alguns métodos para controle de funções via *Javascript*. O controle de funções via linguagem de programação oferece ao desenvolvedor uma maneira de personalizar a entrega de vídeo ao usuário final, porém, não se pode esquecer que o mesmo tem o controle sobre o *browser*, podendo desabilitar o *Javascript* no momento que desejar.

Por isso, a melhor opção é oferecer uma alternativa para o controle via programação com elementos de marcação HTML.

play

Esse método controla o processo de execução do arquivo de vídeo.

Exemplo:

```
...
    video.play();
...

```

pause

No conjunto de métodos da API de vídeo não existe o método *stop*. O processo de parada do vídeo tocado é realizado pelo método *pause*.

Exemplo:

```
...
video.pause();
...
```

canPlayType

Esse método permite indentificar se o agente de usuário (navegador), está preparado para receber as características da API de vídeo HTML 5.

Exemplo:

```
...
if(video.canPlayType) {
    código...
}
else{
    código...
}
...
...
```

load

Esse método carrega o arquivo de vídeo e prepara para a execução através do método *play*.

Exemplo:

```
...
video.load();
...
```

Abaixo segue um modelo de código de controle da API de vídeo do HTML 5 com *Javascript*.

Exemplo:

HTML

```
...
<section>
    <h1>HTML 5 - A Nova Onda</h1>
    <video id="filme">
        <source src="bigBuck.ogv" type="video/ogg">
        <source src=" bigBuck.m4v" type="video/mp4">
        <source src=" bigBuck.webm" type="video/webm">
        <a href="filmes.zip">Baixe os arquivos de vídeo</a>
    </video>
    <div id="botoes">
        <button>play</button>&nbsp;&nbsp;&nbsp;
        <button>pause</button>&nbsp;&nbsp;&nbsp;
        <button>capa</button>&nbsp;&nbsp;&nbsp;
    </div>
</section>
...
```

Javascript

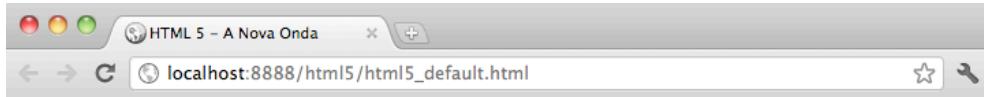
```
...
<script>
window.onload = function(){
    var bt = document.getElementsByTagName('button');
    var filme = document.querySelector('#filme');
    bt[0].addEventListener('click',tocaVideo,false);
    bt[1].addEventListener('click',pausaVideo,false);
    bt[2].addEventListener('click',capaVideo,false);

    function tocaVideo(){
        filme.play();
    }

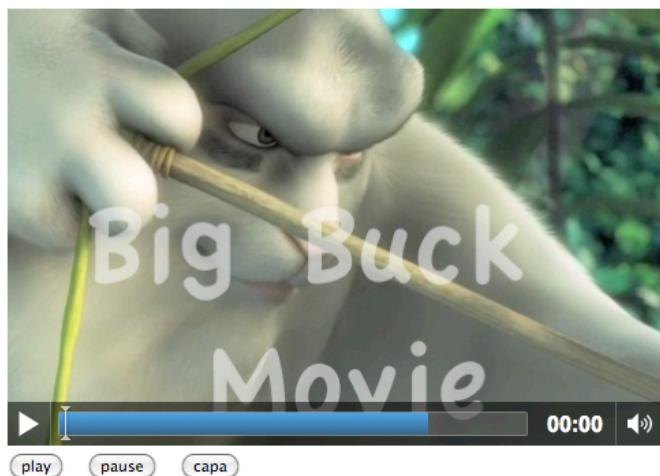
    function pausaVideo(){
        filme.pause();
    }

    function capaVideo(){
        var capa = filme.setAttribute('poster','capa.png');
        filme.appendChild(capa);
    }
};//Função global
</script>
...

```



HTML 5 - A Nova Onda



Capítulo 06: Elementos Validadores de Dados

* Visão Geral	51
* Suporte dos Browsers	51
* Elementos Validadores	51
* Atributos Validadores	57

Visão Geral

Dentro do modelo DOM chamamos de elementos e/ou atributos validadores, as estruturas que manipulam o conteúdo existente para produzir um resultado de acordo com uma condição específica de resposta. Um exemplo é um campo de formulário que não pode ser enviado, se o seu conteúdo for nulo ou vazio.

No HTML 5 existe um conjunto de elementos e atributos que validam dados dentro do conceito acima, sendo alguns novos e outros já conhecidos das versões anteriores (HTML e XHTML).

Suporte dos Browsers

Alguns dos elementos e atributos validadores vem das versões HTML anteriores, e portanto, já estão homologados nas principais versões dos agentes de usuários (navegador) atuais.

Os novos elementos e atributos criados na versão HTML 5 ainda dependem de uma padronização de renderização por parte dos *browsers* atuais, porém, as versões mais recentes estão praticamente compatíveis com 97% deles.

É uma boa prática usar o processo de testes antes de confirmar o uso de algum elemento ou atributo, de acordo com o agente de usuário (navegador).

Elementos Validadores

dataList

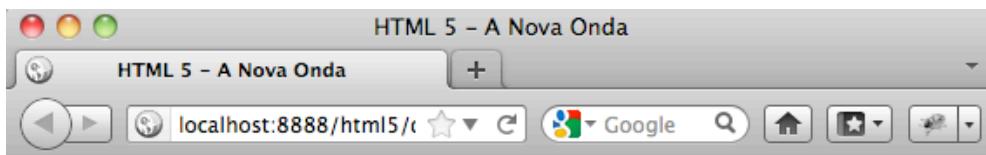
O elemento *dataList* define uma lista de sugestão de dados configurada pelo desenvolvedor, permitindo que o usuário escolha a opção pré-definida ou entre com uma nova opção.

Esse elemento é usado em conjunto com o elemento *input* de formulário, e para essa conexão é necessário que seja colocado no campo *input* o atributo *list*. O valor do atributo *list* deve ser o *id* de identificação do elemento *dataList*.

Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <label>Digite o estado de nascimento:</label><br>
  <input type="text" list="estados" maxlength="15">
  <datalist id="estados">
    <option value="SP">
    <option value="RJ">
    <option value="MG">
    <option value="RS">
    <option value="SC">
  </datalist>
</section>
...
***
```



HTML 5 - A Nova Onda

Digite o estado de nascimento:

SP
RJ
MG
RS
SC



progress

Esse elemento veio de encontro a uma necessidade antiga dos desenvolvedores de ter um objeto que representasse a progressão de dados de uma aplicação.

Ele define o andamento de uma tarefa em relação a sua conclusão, e é um elemento estático. O que quer dizer que não basta marcá-lo no documento para que haja uma funcionalidade dinâmica do processo. Ele tem que ser integrado com a linguagem de programação *Javascript* para que isso aconteça.

Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <label>Carregando...</label><br>
  <progress id="load" max="100" value="0"></progress>&nbsp
  <span>0</span>%
</section>
...
```

Javascript

```
...
<script>
  var valor = 0;
  var timer;
  var section;
  var span;
  var progress;
```

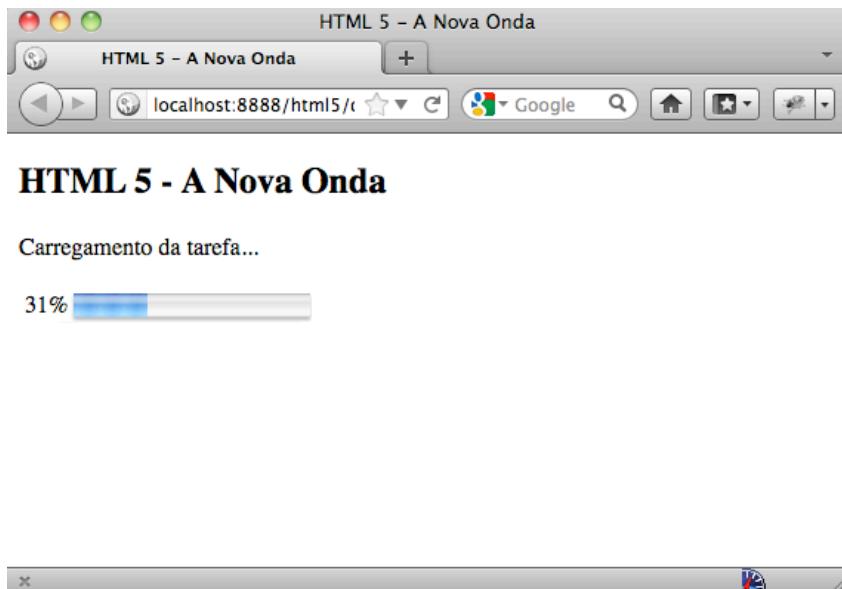
```

function carregaDados(){
    section = document.getElementsByTagName('section')[0];
    span = document.getElementsByTagName('span')[0];
    progress = document.querySelector('#load');
    progress.setAttribute('value', valor);
    section.appendChild(progress);
    span.innerHTML = valor;

    if(valor != 100){
        valor += 1;
        timer = setInterval(carregaDados,1000);
    }
    else{clearInterval(timer);}
}

window.onload = carregaDados;
</script>
...

```



output

O elemento *output* define um *container* para receber um resultado de um cálculo matemático.

Exemplo:

HTML

```

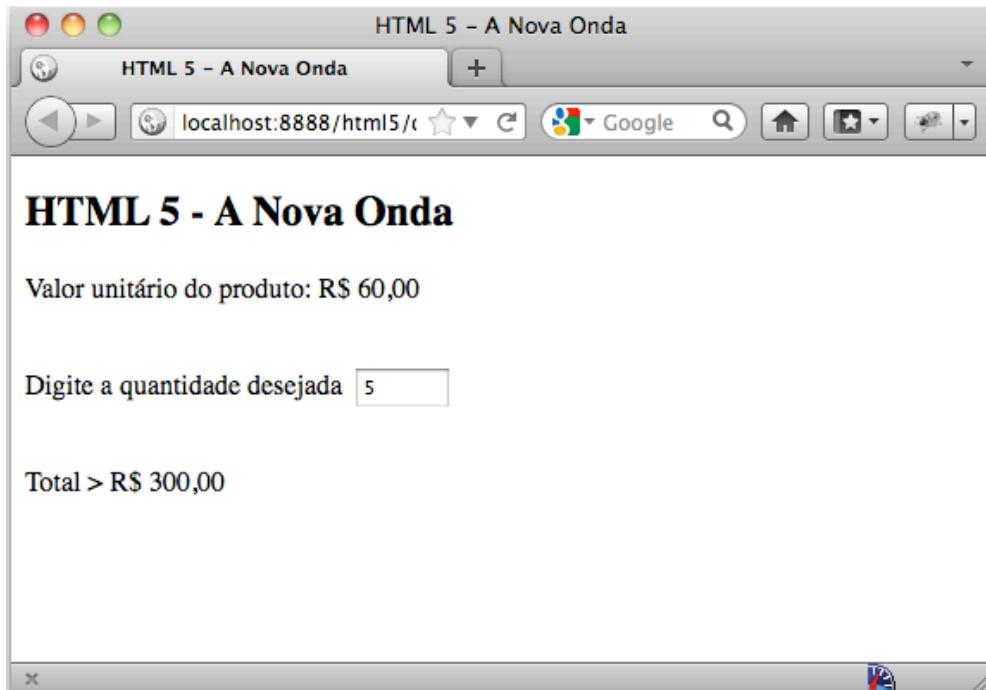
...
<section>
    <h1>HTML 5 - A Nova Onda</h1>
    <p>Valor unitário do produto: R$ 60,00</p><br>
    <label for="qtd">Digite a quantidade desejada</label>&nbsp;
    <input type="text" id="qtd" maxlength="5" size="5"><br><br>
    <p id="resposta">Total > R$&nbsp;<output id="total">0</output></p>
</section>
...

```

Javascript

```
...
<script>
    function calculaTotal(){
        var qtd = document.querySelector('#qtd').value;
        var total = document.querySelector('#total');
        var calculo = parseFloat((qtd*60)).toFixed(2);
        calculo = calculo.replace('.',',');
        total.innerHTML = calculo;
    }

    window.onload = function(){
        var campo = document.querySelector("#qtd");
        campo.addEventListener('blur',calculaTotal,false);
    }
</script>
...
```



time

O elemento *time* define uma marcação para um padrão de hora no formato 0 a 24h, ou uma data baseada no calendário Gregoriano.

O objetivo desse elemento é tornar o mais semântico possível a especificação de um horário e data, para que os dispositivos de acesso a *internet* tenham facilidade de leitura, bem como, os sistemas de buscas.

Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <p>Valor unitário do produto: R$ 60,00</p><br>
  <label for="qtd">Digite a quantidade desejada</label>&nbsp;
  <input type="text" id="qtd" maxlength="5" size="5"><br><br>
  <p id="resposta">Total > R$&nbsp;<output id="total">0</output></p>
</section>
...

```

Javascript

```
...
<script>
  function data(){
    var data = new Date();
    var dia = data.getDate();
    var mes = data.getMonth()+1;
    var ano = data.getFullYear();
    var time = document.getElementsByTagName('time')[0];

    if(dia < 10){
      dia = '0'+dia;
    }

    if(mes < 10){
      mes = '0'+mes;
    }

    var dataCompleta = dia+'/'+mes+'/'+ano;
    time.innerHTML = dataCompleta;
  }

  window.onload = data;
</script>
...

```

O elemento *time* possui dois atributos globais: *datetime* e o *pubdate*.

O atributo *datetime* define um horário e/ou data marcado. Se esse atributo for omitido, o valor de hora e/ou data deve ser colocado como conteúdo *html* do elemento *time*.

Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <h3>Demonstração do elemento time</h3><br>
  <p>Evento realizado <time datetime="2011-11-19">no
    sábado</time> </p>
</section>
...

```

O atributo *pubdate* quando especificado, define um valor booleano que representa a data e/ou hora no instante que foi marcada.

Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <h3>Demonstração do elemento time</h3><br>
  <p>Evento realizado <time pubdate="2011-11-19">hoje</time></p>
</section>
...
```

OBS.: Em uma página *html* padrão poderá ter somente um elemento *time* com o atributo *pubdate*, a não ser que, essa página contenha blocos marcados com o elemento *article*.

Dentro de blocos *article* diferentes é permitido marcar mais de um elemento *time* com o atributo *pubdate*.

A janela de demonstração abaixo refere-se ao código do elemento *timena* página 54.



Atributos Validadores

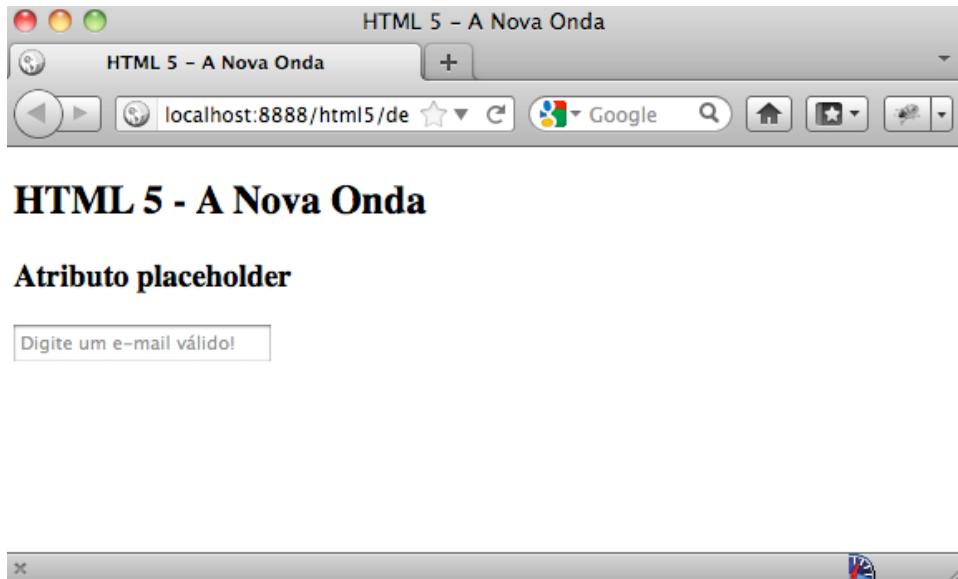
placeholder

O atributo *placeholder* vem acrescentar um rótulo interno ao campo *input* ou *textarea*. Antes do HTML 5 essa funcionalidade era obtida através da linguagem *Javascript*, o que demandava o acréscimo de código para uma simples rotina.

Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <h3>Atributo placeholder</h3>
  <input type="email" placeholder="Digite um e-mail válido!">
</section>
...
```



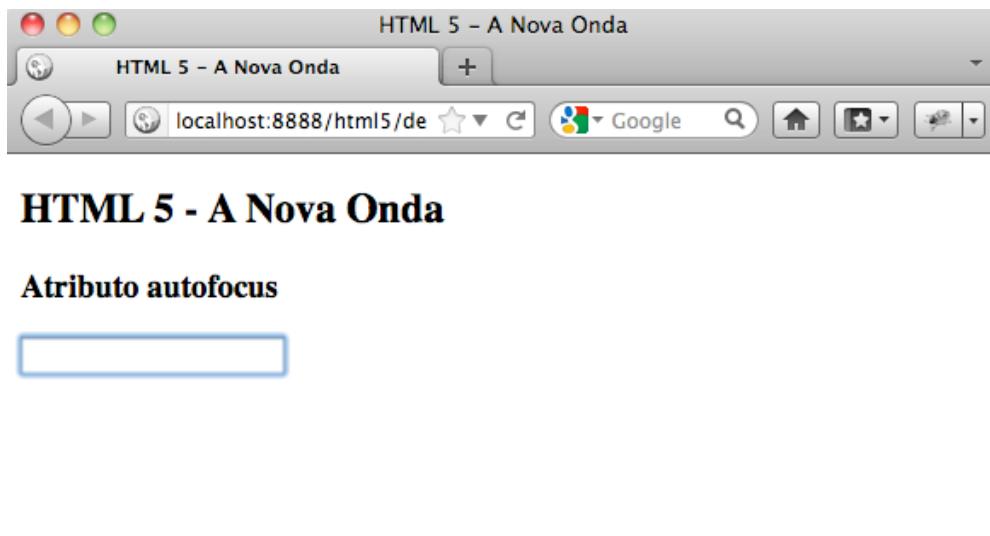
autofocus

O atributo `autofocus` permite estabelecer o foco automático em um campo do tipo `input` ou `textare`a, no carregamento da página.

Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <h3>Atributo autofocus</h3>
  <input type="text" placeholder="Digite seu nome" autofocus>
</section>
...
```



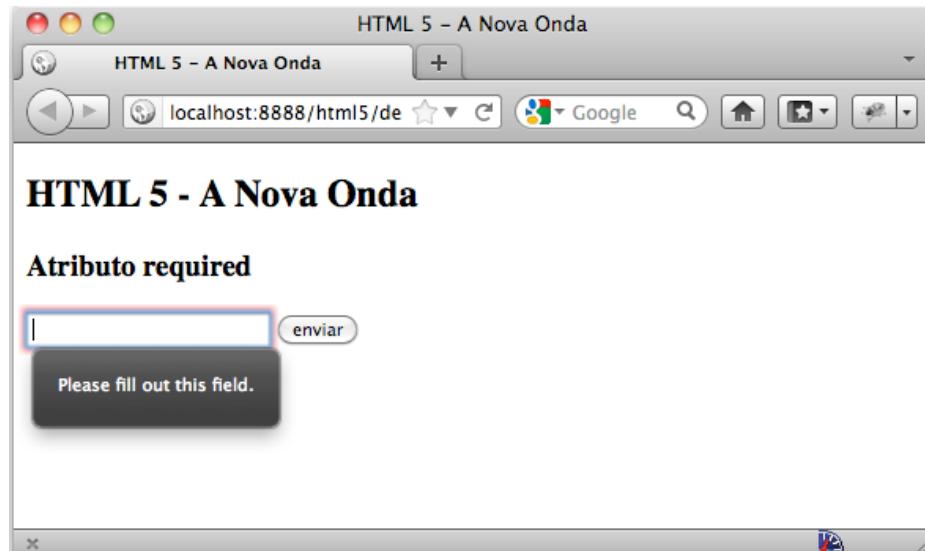
required

Esse atributo define que um campo é de preenchimento obrigatório ou requerido, e dispara uma mensagem de alerta informando que o campo está vazio e necessita ser preenchido. A mensagem do campo requerido é controlado pelo agente de usuário (navegador).

Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <h3>Atributo required</h3>
  <form>
    <input type="text" placeholder="Digite seu nome" autofocus required>
    <input type="submit" value="enviar">
  </form>
</section>
...
```



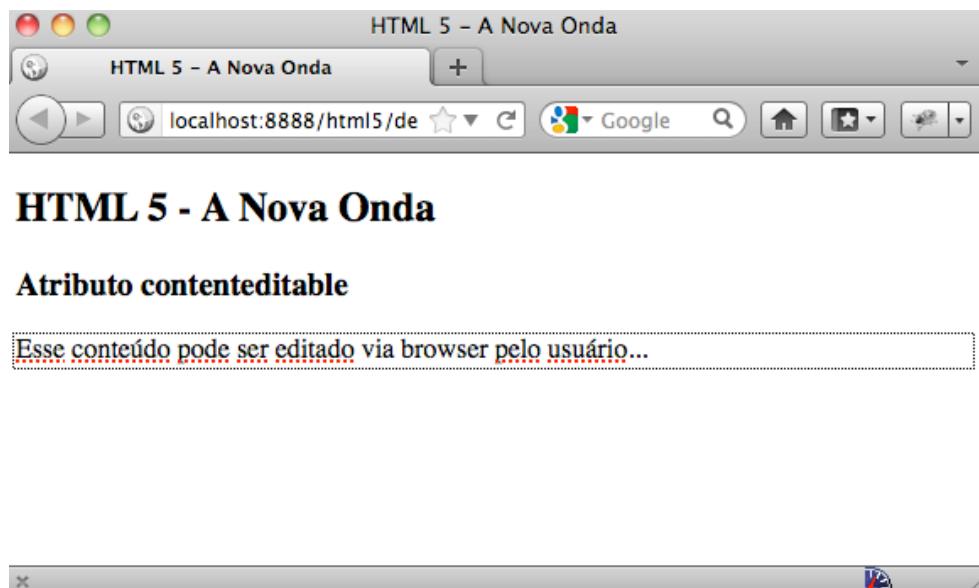
contenteditable

Esse atributo define um conteúdo como editável pelo usuário no navegador.

Exemplo:

HTML

```
...
<section>
    <h1>HTML 5 - A Nova Onda</h1>
    <h3>Atributo contenteditable</h3>
    <article contenteditable="true">
        <p>Esse conteúdo pode ser editado via browser</p>
    </article>
</section>
...
```



draggable

A configuração desse atributo no elemento apenas define que este é arrastável no documento HTML. Para executar o processo de arrastar e soltar específico, é necessário usar os recursos da API Drag and Drop em conjunto com a linguagem Javascript .

Exemplo:

HTML

```
...
<section>
    <h1>HTML 5 - A Nova Onda</h1>
    <h3>Atributo draggable</h3>
    <div draggable="true">
        <p> Essa DIV é arrastável...</p>
    </div>
    </article>
</section>
...
```

OBS.: As funcionalidades de arrastar e soltar serão mostradas no capítulo da API Drag and Drop.

Capítulo 07: API Canvas

* Visão Geral	61
* Suporte dos Browsers	61
* Elemento Canvas	61
* Contexto de Desenho	63
* Elementos de Desenho	63
* Contorno	64
* Preenchimento	65
* Retângulo	66
* Linha	67
* Geométricos	68
* Caminhos	70
* Sombra	74
* Transparência	75
* Rotação	78
* Gradiente	79
* Imagem	80
* Texto	81
* Padrão	83

Visão Geral

Quando falamos de *Canvas*, quem está acostumado com os softwares de tratamento de imagem automaticamente pensa na tela de desenho. E o *Canvas* é isto, uma tela para desenho, e no caso da *web*, a tela do agente de usuário (navegador).

Porém, existe uma certa dúvida quando se deve usar *Canvas* ou *SVG* (*Scalable Vector Graphics*), e para saná-la temos que entender o que é um e outro.

O elemento *Canvas* permite que seja feito o processo de desenho via linguagem de programação *Javascript*, *pixel a pixel*.

O *SVG* é uma linguagem de marcação baseada no XML para a construção de imagens gráficas vetoriais, portanto, manipulável via DOM.

Vantagens Canvas versus SVG

SVG

- ✓ Elemento acessível por leitores de tela.
- ✓ Elemento não perde a resolução se for redimensionado.
- ✓ Elemento acessível via DOM.

Canvas

- ✓ Na maioria das vezes, a performance de renderização do canvas é superior ao SVG.
- ✓ Elemento não perde a resolução se for redimensionado.
- ✓ Elemento acessível via DOM.

Suporte dos Browsers

Os principais *browsers* em suas versões mais recentes, já possuem suporte para desenho de gráficos com o elemento *Canvas*.

Porém, não podemos considerar como um padrão estabelecido, e todo desenvolvimento que necessite desse elemento deve ser testado.

Elemento Canvas

Esse elemento define um *container* para desenho de gráficos vetoriais, dentro de um documento HTML. Por padrão, suas medidas são 300 x 150 *pixels* se não for especificada uma medida personalizada. Essas medidas podem ser colocadas com os atributos *width* e *height*, ou via CSS.

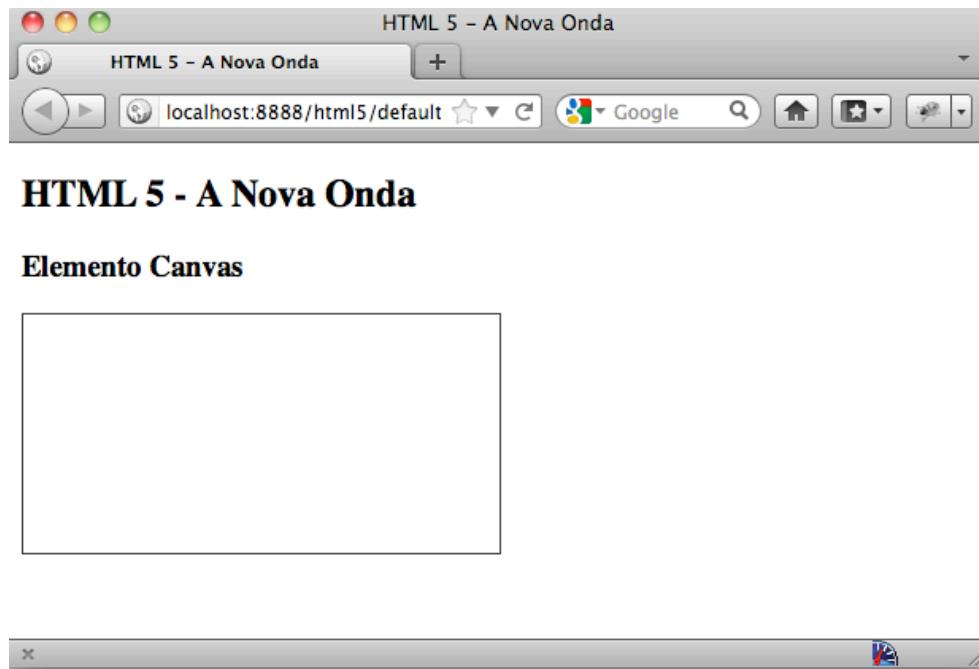
Exemplo:

HTML

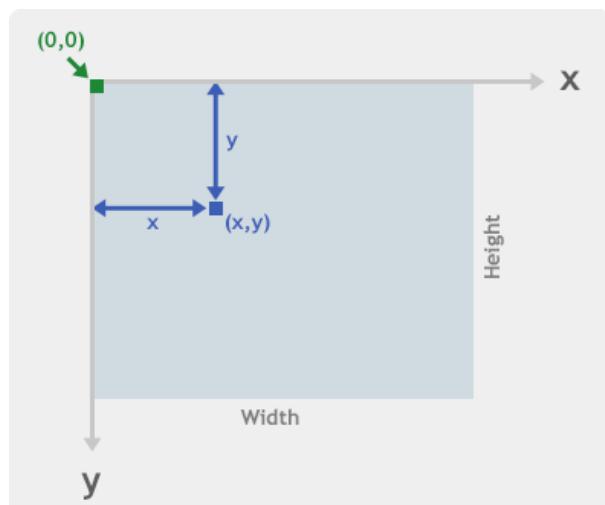
```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <h3>Elemento Canvas</h3>
  <canvas></canvas>
</section>
...
```

CSS

```
...
<style>
  canvas{
    border: 1px solid #333;
  }
</style>
...
```



A criação dos gráficos e textos são realizados dentro desse container *canvas*, assim como, a aplicação dos métodos da *API Canvas*. A definição dos gráficos dentro da área do *canvas* é feita através de coordenadas cartesianas *x* e *y*.



É tomado como ponto de referência 0 no plano cartesiano do *canvas*, o canto esquerdo superior da área definida pelas medidas padrão ou personalizadas.

A proporção visual dos elementos desenhados no *canvas* está diretamente relacionado com as medidas de largura e altura do mesmo.

Sendo assim, um gráfico desenhado em um *canvas* com uma largura de 150 x 75 *pixels* terá sua imagem proporcional, em relação a outro com 150 x 105 *pixels*.

Contexto de Desenho

Para que o processo de desenho possa ser efetivado dentro do elemento *canvas*, é necessário especificar um padrão ou contexto em que esse desenho será realizado.

Esse padrão ou contexto pode ser de duas formas: 2D ou 3D.

Até o presente momento em que essa apostila está sendo escrita, nenhum dos *browsers* em suas mais recentes versões está preparado para renderização de gráficos em 3D. Portanto, nosso contexto de desenho no *canvas* será em 2D por enquanto.

Para representar esse contexto na interação com o elemento *canvas* usamos a referência *Javascript* a seguir:

Exemplo:

Javascript

```
...
<script>
  var canvas = document.querySelector('#cv');
  var contexto = canvas.getContext('2d');
</script>
...
```

Definido o contexto de desenho, o elemento *canvas* está preparado para receber qualquer gráfico ou imagem em seu conteúdo.

Para simplificar o processo de codificação dos exemplos mostrados nessa apostila, será usado a grafia *ctx* para representar no código *Javascript* o contexto de desenho.

Exemplo:

Javascript

```
...
<script>
  var canvas = document.querySelector('#cv');
  var ctx = canvas.getContext('2d');
</script>
...
```

Elementos de Desenho

A API *Canvas* possui um conjunto de elementos para a produção de gráficos, imagens e textos.

Cada elemento possui uma função específica no processo de construção dos gráficos e imagens, e serão abordados os recursos fundamentais de cada um, deixando a cargo de cada desenvolvedor o aprimoramento desses recursos em função de sua necessidade e criatividade.

Vale ressaltar que o desenvolvimento de gráficos e imagens com o elemento *canvas* requer conhecimento de coordenadas cartesianas *x* e *y* e posicionamento de objetos nesse plano, pois todo o processo é realizado com base nos mesmos.

Não há, até o presente momento, nenhuma ferramenta que crie gráficos e imagens automaticamente para o elemento *canvas*.

Todo o processo é feito através de código de programação *Javascript* puro.

Contorno

Dentro do processo de produção de gráficos e imagens um dos elementos fundamentais é o contorno (*stroke* em inglês). Esse recurso define os limites de um gráfico ou imagem e pode ser usado, ou não, na composição visual do elemento. Dentro dos elementos de desenho da *API Canvas* existe o elemento *stroke* para esse fim.

Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <h3>Elemento strokeStyle</h3>
  <canvas id="cv"></canvas>
</section>
...

```

CSS

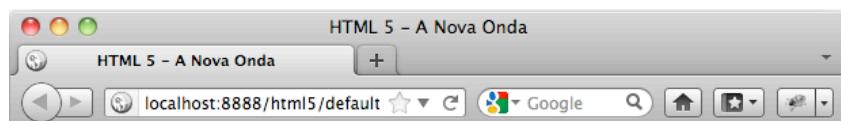
```
...
<style>
  canvas{border: 1px solid #333;}
</style>
...

```

Javascript

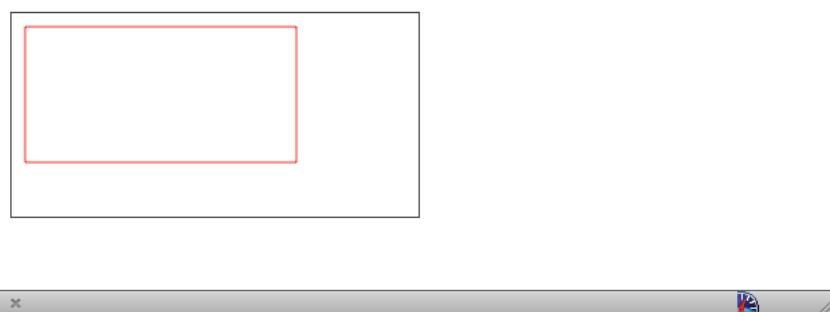
```
...
<script>
  function desenhar() {
    var canvas = document.querySelector('#cv');
    var ctx = canvas.getContext('2d');
    ctx.strokeStyle = "#F00";
    ctx.strokeRect(10,10,200,100);
  }
  window.onload = desenhar;
</script>
...

```



HTML 5 - A Nova Onda

Elemento strokeStyle



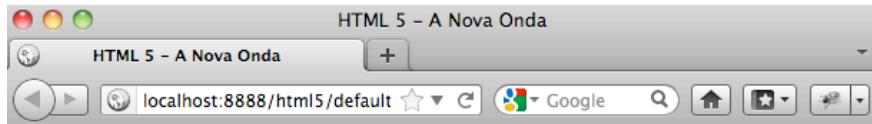
Preenchimento

Outro elemento fundamental no processo de desenho de gráficos e imagens no *canvas* é o preenchimento (*fill* em inglês). Ele define os limites de um gráfico ou imagem através de uma área coberta por uma cor, um padrão (*pattern*) ou textura. Na *API Canvas* existe um elemento que cumpre esse papel: o *fill*.

Exemplo:

Javascript

```
...
<script>
    function desenhar() {
        var canvas = document.querySelector('#cv');
        var ctx = canvas.getContext('2d');
        ctx.fillStyle = "#F00";
        ctx.fillRect(10,10,200,100);
    }
    window.onload = desenhar;
</script>
...
```



HTML 5 - A Nova Onda

Elemento fillStyle



Retângulo

Como uma das formas básicas para desenho temos as funções *strokeRect* e *fillRect*, que desenham a primitiva retângulo dentro da faixa de coordenadas *x* e *y* do *canvas*.

A sintaxe da função é *strokeRect(x,y,width, height)* ou *fillRect(x,y,width, height)*.

Exemplo:

Javascript

```
...
<script>
    function desenhar() {
        var canvas = document.querySelector('#cv');
        var ctx = canvas.getContext('2d');
```

```

ctx.strokeStyle = "#000";
ctx.fillStyle = "#F00";
ctx.strokeRect(10,10,200,100);
ctx.fillRect(10,10,200,100);
}
window.onload = desenhar;
</script>
...

```

Quando é feito um desenho no *canvas* ele permanece nessa área, e se outro desenho for feito nas mesmas coordenadas haverá sobreposição. Para evitar que isso aconteça é necessário limpar a área do desenho anterior, para que se possa incluir um novo no local. A função *clearRect* define uma posição nas coordenadas x,y e uma largura e altura para limpeza do(s) elemento(s) desenhado(s).

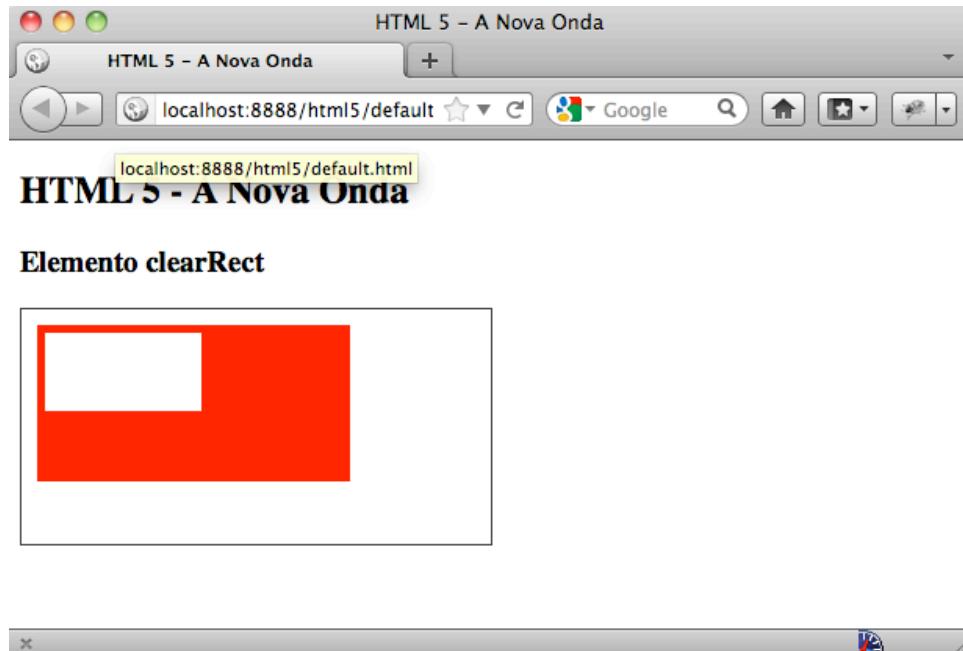
Exemplo:

Javascript

```

...
<script>
function desenhar() {
    var canvas = document.querySelector('#cv');
    var ctx = canvas.getContext('2d');
    ctx.fillStyle = "#F00";
    ctx.fillRect(10,10,200,100);
    ctx.clearRect(15,15,100,50);
}
window.onload = desenhar;
</script>
...

```



OBS.: A função *clearRect* é usada para fazer a limpeza de uma área de desenho, independente do tipo que foi desenhado.

Linha

Um elemento fundamental no processo de desenho é a linha, que permite definir limites visuais de contorno e realizar traçados de várias formas geométricas.

Dentro da *API Canvas* esse elemento possui o atributo *lineWidth* que determina a espessura da linha desenhada, e o *lineTo* que indica a posição do traçado da linha. Esse último será descrito no tópico sobre Caminhos.

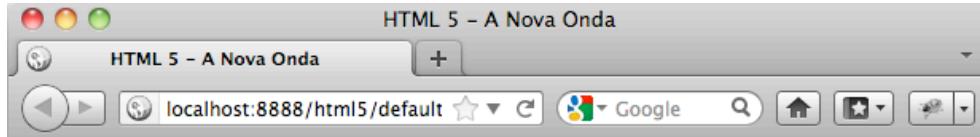
Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <h3>Elemento lineWidth</h3>
  <canvas id="cv"></canvas>
</section>
...
```

Javascript

```
...
<script>
  function desenhar() {
    var canvas = document.querySelector('#cv');
    var ctx = canvas.getContext('2d');
    ctx.strokeStyle = "#000";
    ctx.lineWidth = 2;
    ctx.strokeRect(10,10,200,100);
  }
  window.onload = desenhar;
</script>
...
```



HTML 5 - A Nova Onda

Elemento lineWidth



Geométricos

Dentro da *API Canvas* existem as principais formas primitivas geométricas: retângulo, círculo, elipse e polígono. A diferença está em como cada uma delas pode ser desenhada dentro do *canvas*.

Uma figura geométrica usada nos tópicos anteriores é a retângulo, e para ela existe uma função própria que permite seu desenho imediato: *rect()*.

A mesma coisa acontece com a primitiva círculo/elipse que possui a função própria *arc()*.

Porém, para se desenhar uma primitiva poligonal dentro do *canvas* temos que utilizar uma forma personalizada de desenho, que é chamada de *paths* (caminhos). Veremos essa forma mais adiante.

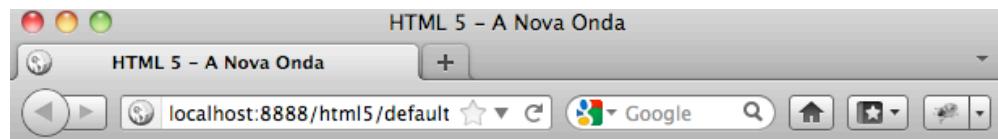
1. rect

A função *rect* define um elemento retângulo com os seguintes parâmetros: posição x, posição y, largura e altura.

Exemplo:

Javascript

```
...
<script>
    function desenhar() {
        var canvas = document.querySelector('#cv');
        var ctx = canvas.getContext('2d');
        ctx.strokeStyle = "#000";
        ctx.rect(10,10,200,100);
        ctx.stroke();
    }
    window.onload = desenhar;
</script>
...
```



HTML 5 - A Nova Onda

Elemento rect



1. arc

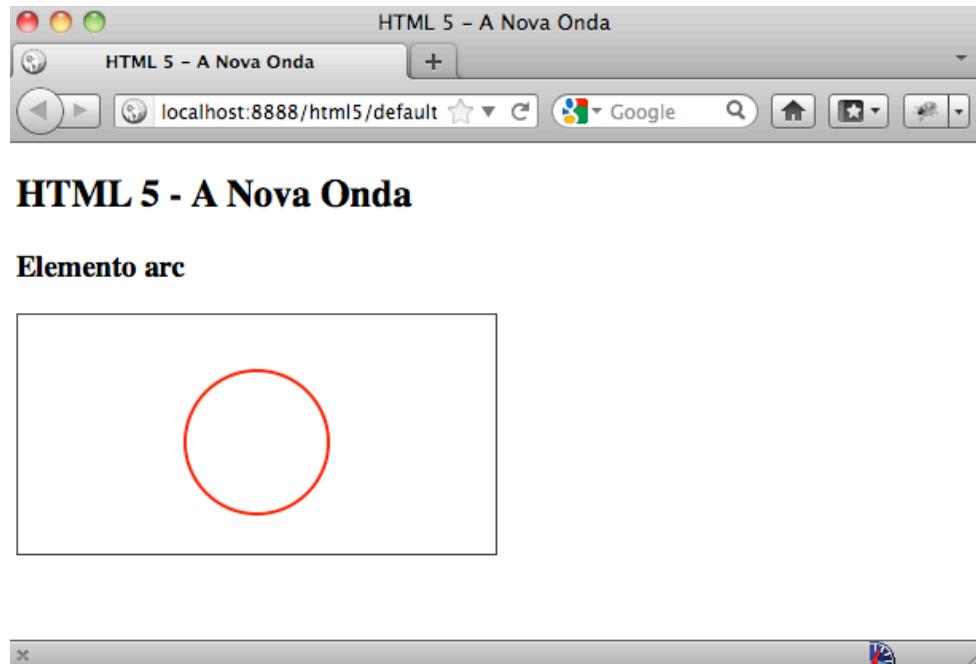
A função *arc* define um elemento que pode ser do formato de um círculo, e o desenha na área de *canvas* conforme suas coordenadas.

A sintaxe da função é: *arc(x,y,raio, ângulo inicial, ângulo final, sentido do desenho)*.

Exemplo:

Javascript

```
...
<script>
    function desenhar() {
        var canvas = document.querySelector('#cv');
        var ctx = canvas.getContext('2d');
        ctx.lineWidth = 2;
        ctx.strokeStyle = "#F00";
        ctx.arc(150,80,45,0,Math.PI*2,true);
        ctx.stroke();
    }
    window.onload = desenhar;
</script>
...
```



OBS.: Uma variação do elemento *arc* é o *arcTo* que desenha um círculo com origem nas coordenadas x_1, y_1 e fim nas coordenadas x_2, y_2 e um raio.

A sintaxe é *arcTo(x1,y1,x2,y2,raio)*.

Caminhos

Dentro da *API Canvas* para se desenhar um gráfico ou imagem com formato personalizado, é necessário o uso da função *beginPath()*. Ela permite a construção de traços sobre o *canvas* definindo a forma de desenho desejada. Podem ser usadas as funções *rect* e *arc* para formas pré-definidas.

Exemplo:

HTML

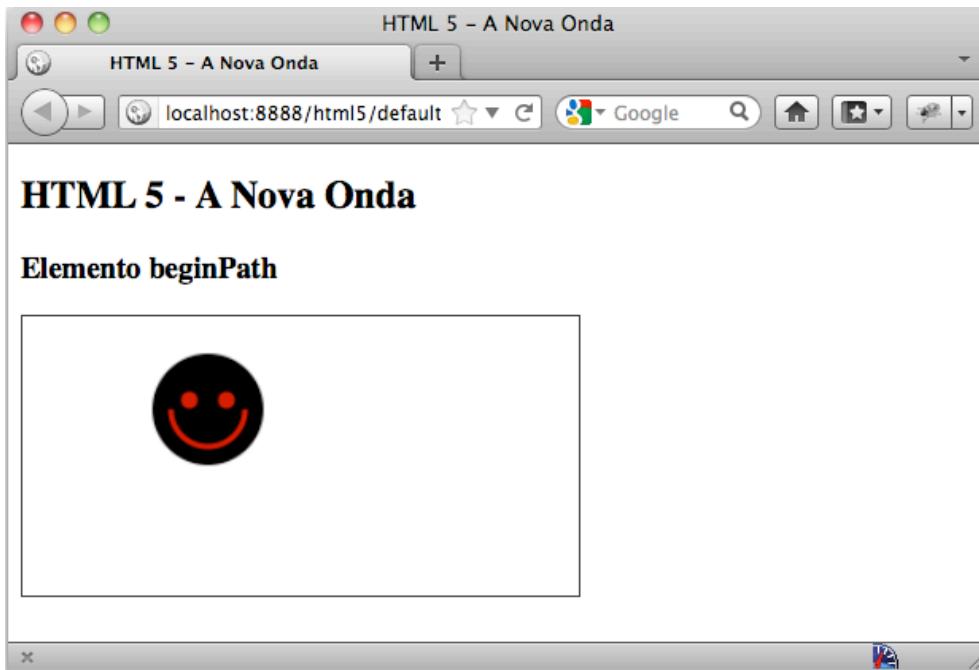
```
...
<section>
    <h1>HTML 5 – A Nova Onda</h1>
    <h3>Elemento beginPath</h3>
    <canvas id="cv"></canvas>
</section>
...
...
```

CSS

```
...
<style>
    canvas{
        border: 1px solid #333;
        width: 350px;
        height: auto;
    }
</style>
...
...
```

Javascript

```
...
<script>
    function desenhar() {
        var canvas = document.querySelector('#cv');
        var ctx = canvas.getContext('2d');
        //Desenhando a cabeça do boneco
        ctx.beginPath();
        ctx.arc(100, 50, 30, 0, Math.PI*2, true);
        ctx.fill();
        //Desenhando o sorriso do boneco
        ctx.beginPath();
        ctx.strokeStyle = '#c00';
        ctx.lineWidth = 3;
        ctx.arc(100, 50, 20, 0, Math.PI, false);
        ctx.stroke();
        //Desenhando os olhos do boneco
        ctx.beginPath();
        ctx.fillStyle = '#c00';
        ctx.arc(90, 45, 3, 0, Math.PI*2, true);
        ctx.fill(); ctx.moveTo(113, 45);
        ctx.arc(110, 45, 3, 0, Math.PI*2, true);
        ctx.fill();
        ctx.stroke();
    }
    window.onload = desenhar;
</script>
...
```



Dentro do processo de construção de desenhos com *beginPath* pode haver a necessidade de fechar os pontos de uma forma aberta, e para isso existe a função *closePath*. Ela define o fechamento de um caminho ligando o ponto inicial e final de um desenho, por uma linha reta.

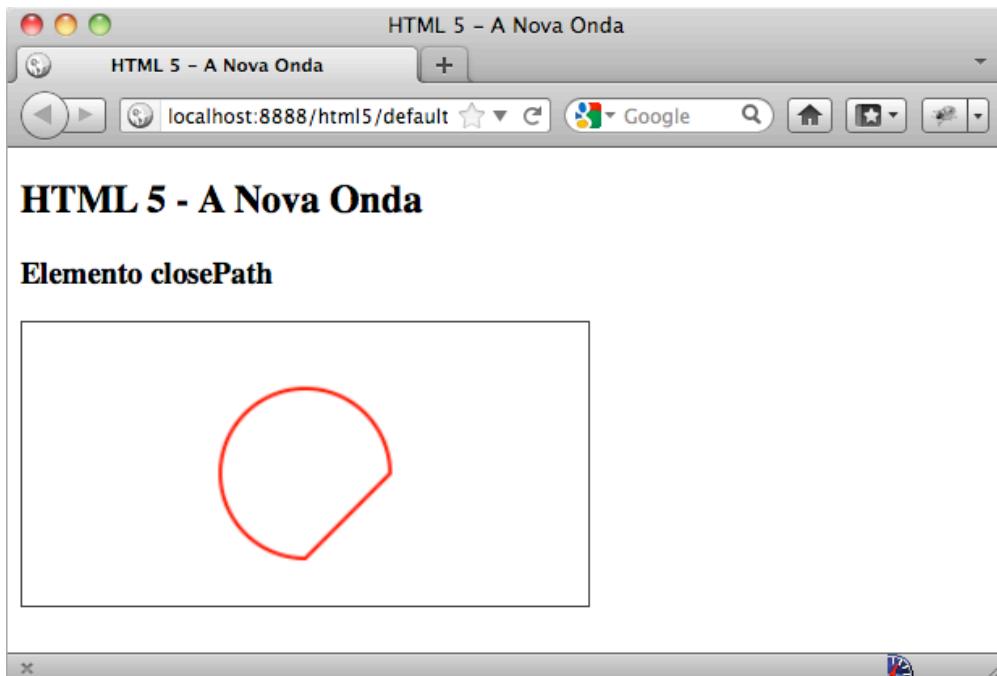
Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <h3>Elemento closePath</h3>
  <canvas id="cv"></canvas>
</section>
...
```

Javascript

```
...
<script>
  function desenhar() {
    var canvas = document.querySelector('#cv');
    var ctx = canvas.getContext('2d');
    ctx.strokeStyle = "#F00";
    ctx.arc(150, 80, 45, 0, -3/2*Math.PI, true);
    ctx.closePath();
    ctx.stroke();
  }
  window.onload = desenhar;
</script>
...
```

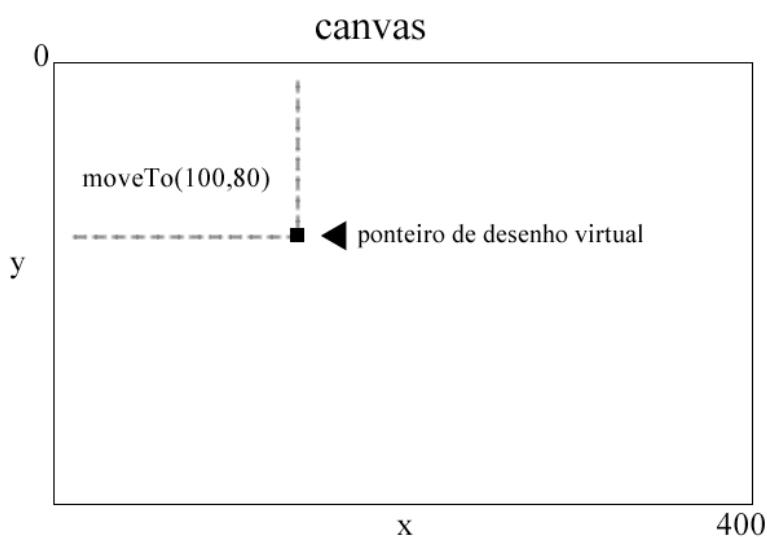


Dentro da construção de formas poligonais com o elemento *beginPath* existem métodos que ajudam no desenho das mesmas.

São elas:

1. **moveTo**

O método *moveTo* posiciona o ponteiro de desenho nas coordenadas *x* e *y*, e permite o traçado em diversos pontos do elemento *canvas*.



OBS.: O ponteiro de desenho controlado pela API *Canvas* é um elemento virtual, portanto, não visível para o desenvolvedor.

2. lineTo

O método *moveTo* posiciona o ponteiro de desenho nas coordenadas *x* e *y*, e permite o traçado em diversas direções do elemento *canvas* para a construção de formas poligonais.

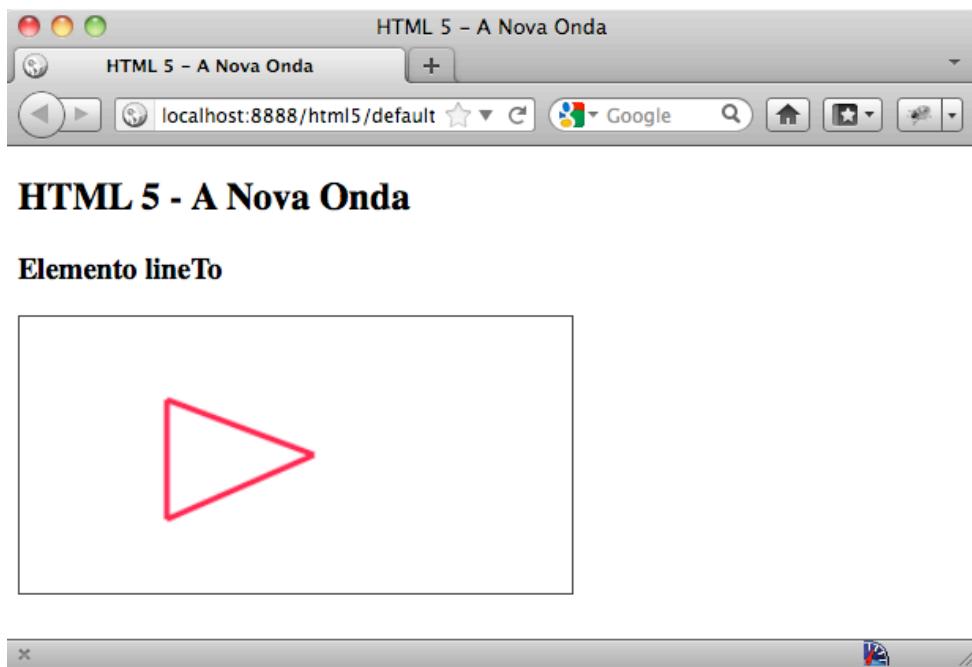
Exemplo:

HTML

```
...
<section>
    <h1>HTML 5 - A Nova Onda</h1>
    <h3>Elemento lineTo</h3>
    <canvas id="cv"></canvas>
</section>
...
```

Javascript

```
...
<script>
    function desenhar(){
        var canvas = document.querySelector('canvas');
        var ctx = canvas.getContext('2d');
        ctx.lineWidth = 3;
        ctx.strokeStyle = "rgb(255,0,63)";
        ctx.beginPath();
        ctx.moveTo(80,110);
        ctx.lineTo(80,45);
        ctx.lineTo(160,75);
        ctx.lineTo(160,75);
        ctx.stroke();
        ctx.closePath();
    }
    window.onload = desenhar;
</script>
...
```



Sombra

Um dos efeitos mais utilizados no processo de desenho de imagens e formas geométricas é a sombra. Sua aplicação no objeto de desenho gera um plano de profundidade, dando uma idéia de perspectiva. É possível aplicar sombra nos objetos desenhados no *canvas* através da propriedade *shadow*.

A propriedade *shadow* possui 4 opções de configuração que devem ser definidas para que o efeito seja produzido. São elas:

shadowColor

Define a cor da sombra sobre o objeto de desenho. Seu valor pode ser configurado por notação hexadecimal ou *rgb*.

Exemplo: *shadowColor* = "#FF0" ou *shadowColor* = "rgb(255,255,0)"

shadowOffsetX

Essa propriedade define o deslocamento da sombra no eixo x, em relação ao objeto no qual ela está aplicada. O valor dessa propriedade pode ser um número inteiro positivo ou negativo.

Exemplo: *shadowOffsetX* = 15

shadowOffsetY

Essa propriedade define o deslocamento da sombra no eixo y, em relação ao objeto no qual ela está aplicada. O valor dessa propriedade pode ser um número inteiro positivo ou negativo.

Exemplo: *shadowOffsetY* = 5

shadowBlur

Essa propriedade define a intensidade do desfoque da sombra sobre o objeto no qual foi aplicada. O valor dessa propriedade deve ser um número inteiro positivo, e quanto maior os valores mais transparente será o efeito de desfoque.

Exemplo: *shadowBlur* = 15

Abaixo segue um exemplo completo de aplicação da propriedade *shadow*:

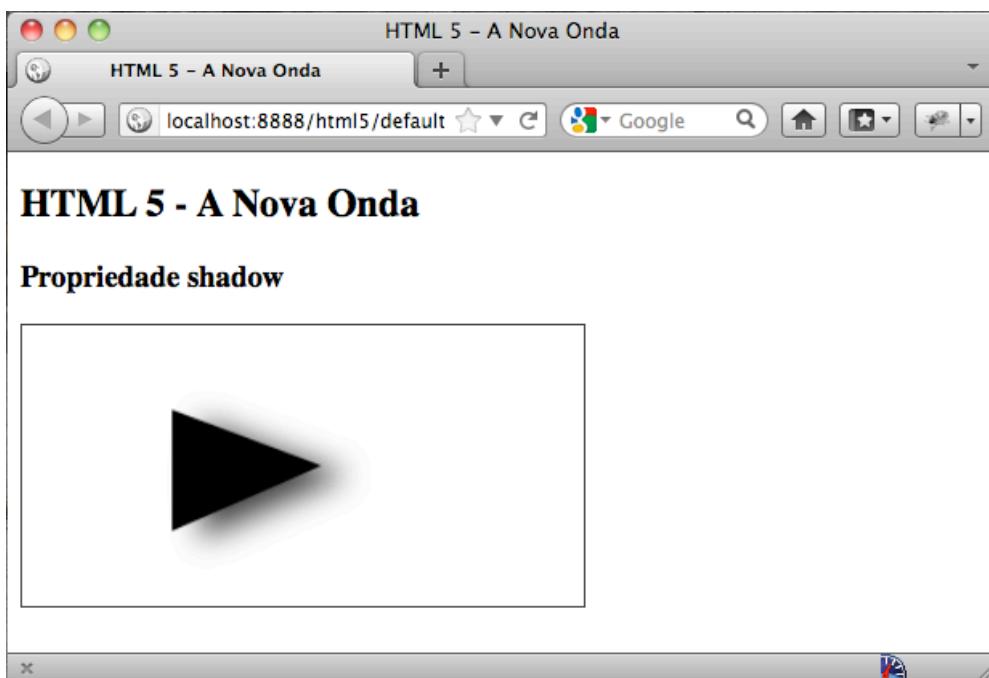
Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <h3>Elemento shadow</h3>
  <canvas id="cv"></canvas>
</section>
...
...
```

Javascript

```
...
<script>
    function desenhar(){
        var canvas = document.querySelector('canvas');
        var ctx = canvas.getContext('2d');
        ctx.fillStyle = "#FF0";
        ctx.shadowColor = "#333";
        ctx.shadowOffsetX = 15;
        ctx.shadowOffsetY = 5;
        ctx.shadowBlur = 15;
        ctx.beginPath();
        ctx.moveTo(80,110);
        ctx.lineTo(80,45);
        ctx.lineTo(160,75);
        ctx.lineTo(160,75);
        ctx.fill();
        ctx.closePath();
    }
    window.onload = desenhar;
</script>
...
```



Transparência

Outro efeito muito utilizado é o de transparência, também conhecido como alfa. Esse efeito modifica a estrutura de visualização de um objeto, através do processo de opacidade entre os valores de 0 e 1. A proporção entre essa faixa de valores são suas frações. Dentro do objeto *canvas* podemos aplicar transparência nos objetos de duas maneiras:

globalAlpha

Define uma transparência para todos os objetos desenhados no elemento *canvas*.

rgba

Através da propriedade de cor `rgb` podemos definir um canal alfa, que será aplicado ao elemento específico que recebe essa configuração.

Exemplo:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <h3>Elemento alpha</h3>
  <canvas id="cv"></canvas>
</section>
...
```

Javascript

```
...
<script>
  function desenhar(){
    var canvas = document.querySelector('canvas');
    var ctx = canvas.getContext('2d');
    ctx.fillStyle = "#333";
    ctx.globalAlpha = 0.25;
    ctx.beginPath();
    ctx.moveTo(80,110);
    ctx.lineTo(80,45);
    ctx.lineTo(160,75);
    ctx.lineTo(160,75);
    ctx.fill();
    ctx.closePath();
  }
  window.onload = desenhar;
</script>
...
```



Exemplo 2:

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <h3>Elemento alpha</h3>
  <canvas id="cv"></canvas>
</section>
...
...
```

Javascript

```
...
<script>
  function desenhar(){
    var canvas = document.querySelector('canvas');
    var ctx = canvas.getContext('2d');
    ctx.fillStyle = rgba(153,153,153,0.25);
    ctx.beginPath();
    ctx.moveTo(80,110);
    ctx.lineTo(80,45);
    ctx.lineTo(160,75);
    ctx.lineTo(160,75);
    ctx.fill();
    ctx.closePath();
    ctx.fillStyle = "#F00";
    ctx.fillRect(180,25,80,80);
  }
  window.onload = desenhar;
</script>
...
```



Rotate

A propriedade *rotate* rotaciona o objeto desenhado usando a estrutura do *canvas*, e tomando como base a posição inicial 0 dos eixos x e y. **O objeto não gira em torno do seu próprio eixo, e sim, o elemento canvas.** A rotação é feita com um valor definido em radianos ($\pi/180$), e pode ser positivo (sentido horário) ou negativo (sentido anti-horário).

Exemplo :

HTML

```
...
<section>
    <h1>HTML 5 – A Nova Onda</h1>
    <h3>Elemento rotate</h3>
    <canvas id="cv"></canvas>
</section>
...
...
```

Javascript

```
...
<script>
    function desenhar() {
        var canvas = document.querySelector('canvas');
        var ctx = canvas.getContext('2d');
        ctx.fillStyle = "rgba(153,153,153,0.25)";
        ctx.beginPath();
        ctx.moveTo(80,110);
        ctx.lineTo(80,45);
        ctx.lineTo(160,75);
        ctx.lineTo(160,75);
        ctx.fill();
        ctx.closePath();
        ctx.rotate(-0.2115);
        ctx.fillStyle = "#F00";
        ctx.fillRect(180,25,80,80);
    }
    window.onload = desenhar;
</script>
...
```



Gradiente

Dentro do processo de desenho o elemento cor é de fundamental importância para destacar um objeto de outro. A API Canvas do HTML 5 possui a propriedade gradiente, que permite aplicar uma sequência de cores a um objeto desenhado.

Existem três métodos para a criação de gradiente no *canvas*:

createLinearGradient

Define um modelo de gradiente linear sobre o objeto desenhado. Sua configuração necessita da especificação das coordenadas x,y e x1,y1, representando os pontos de início e término do modelo de gradiente.

createRadialGradient

Esse método define um modelo de gradiente radial sobre o objeto desenhado. Sua configuração necessita da especificação das coordenadas x,y e x1,y1, representando os pontos de início e término do modelo de gradiente. Além das coordenadas x e y, são necessárias as definições de r e r1, que representam os raios do modelo de gradiente.

addColorStop

Esse método configura como as cores do modelo de gradiente serão distribuídas pelo objeto desenhado. Possui dois parâmetros: *offset* e *cor*.

O parâmetro *offset* define as cores do gradiente sobre a área aplicada entre os valores de 0 e 1. O valor 0 representa a cor de início e o valor 1 a cor final do gradiente. O parâmetro *cor* define as cores aplicadas no gradiente.

Como o processo de criação de um gradiente requer duas cores, ou mais, o método *addColorStop* deve ser aplicado o número de vezes necessários para criar a configuração desejada.

Exemplo :

HTML

```
...
<section>
    <h1>HTML 5 - A Nova Onda</h1>
    <h3>Elemento gradiente</h3>
    <canvas id="cv"></canvas>
</section>
...
```

Javascript

```
...
<script>
    function desenhar() {
        var canvas = document.querySelector('canvas');
        var ctx = canvas.getContext('2d');
        var gradiente = ctx.createLinearGradient(0,15,250,100);
        gradiente.addColorStop(0,'#036');
        gradiente.addColorStop(1,'#D9DDE1');
        ctx.fillStyle = gradiente;
        ctx.fillRect(10,15,250,125);
        ctx.fill();
    }
    window.onload = desenhar;
</script>
...
```



Imagen

Além dos elementos geométricos de desenho, é possível trabalhar com imagens dentro do elemento *canvas*.

Para esse fim existe na API o método *drawImage*, que permite inserir uma imagem de um elemento *img*, *canvas* ou vídeo. Os parâmetros do método *drawImage* são: imagem a ser inserida, coordenada x e coordenada y.

A imagem para aparecer no *canvas* deve estar completamente carregada.

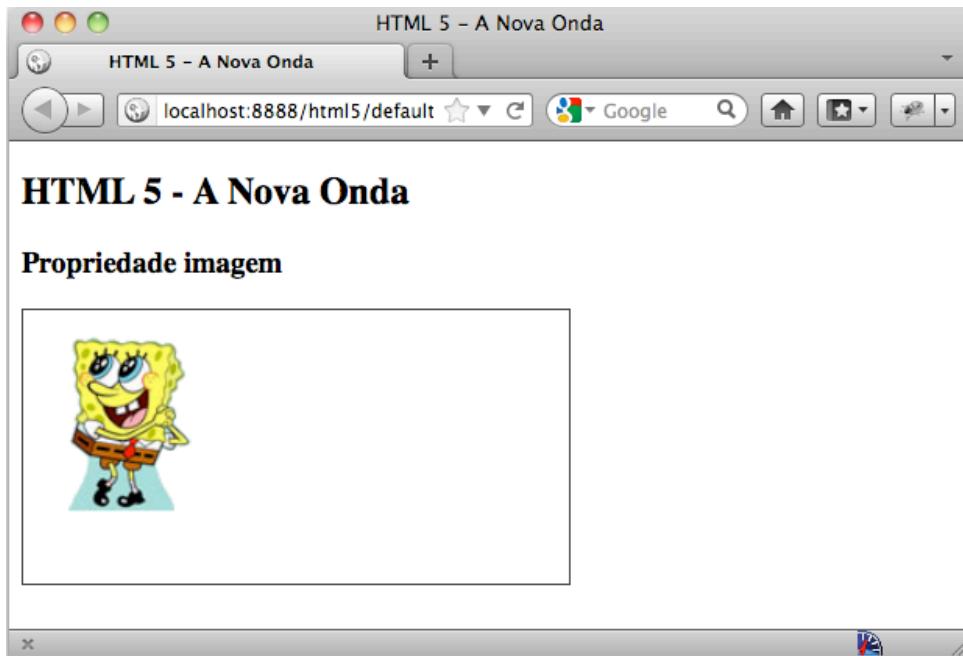
Exemplo :

HTML

```
...
<section>
    <h1>HTML 5 - A Nova Onda</h1>
    <h3>Elemento imagem</h3>
    <canvas id="cv"></canvas>
</section>
```

Javascript

```
...
<script>
    function desenhar() {
        var canvas = document.querySelector('canvas');
        var ctx = canvas.getContext('2d');
        var imagem = new Image();
        var imagem.src = 'bob.png';
        imagem.onload = function() {
            ctx.drawImage(imagem, 25, 15);
        }
    }
    window.onload = desenhar;
</script>
```



Texto

Um elemento importante no processo de desenho é o texto.

Para a manipulação de textos, a API *canvas* oferece cinco métodos mostrados a seguir:

font

Esse método define o tamanho da fonte e sua família tipográfica.

fillText

Define o tipo de preenchimento do texto a ser inserido no *canvas*. Esse método admite quatro parâmetros, sendo três obrigatórios e um opcional.

O primeiro parâmetro é o texto a ser inserido.

O segundo parâmetro é a coordenada x em que o texto será posicionado horizontalmente.

O terceiro parâmetro é a coordenada y em que o texto será posicionado verticalmente.

O quarto parâmetro é facultativo, e define o comprimento máximo que o texto ocupará na área em que foi posicionado.

strokeText

Define o tipo de preenchimento do texto a ser inserido no *canvas*. Esse método admite quatro parâmetros, sendo três obrigatórios e um opcional.

O primeiro parâmetro é o texto a ser inserido.

O segundo parâmetro é a coordenada x em que o texto será posicionado horizontalmente.

O terceiro parâmetro é a coordenada y em que o texto será posicionado verticalmente.

O quarto parâmetro é facultativo, e define o comprimento máximo que o texto ocupará na área em que foi posicionado.

textAlign

Esse método como o próprio nome sugere, alinha o texto inserido no *canvas* em um dos cinco valores de alinhamento: *start*, *end*, *left*, *right* e *center*.

O ponto de referência para alinhamento é uma linha vertical imaginária, definida na posição x do texto desenhado com os métodos *fillText* ou *strokeText*.

textBaseline

O método *textBaseline* define o posicionamento do texto no *canvas*, tomando como ponto de referência a linha-base em que o texto é desenhado.

Esse método *textBaseline* possui como parâmetros os valores: *top*, *hanging*, *ideographic*, *middle*, *alphabetic* e *bottom*.

O Valor padrão é *alphabetic*.

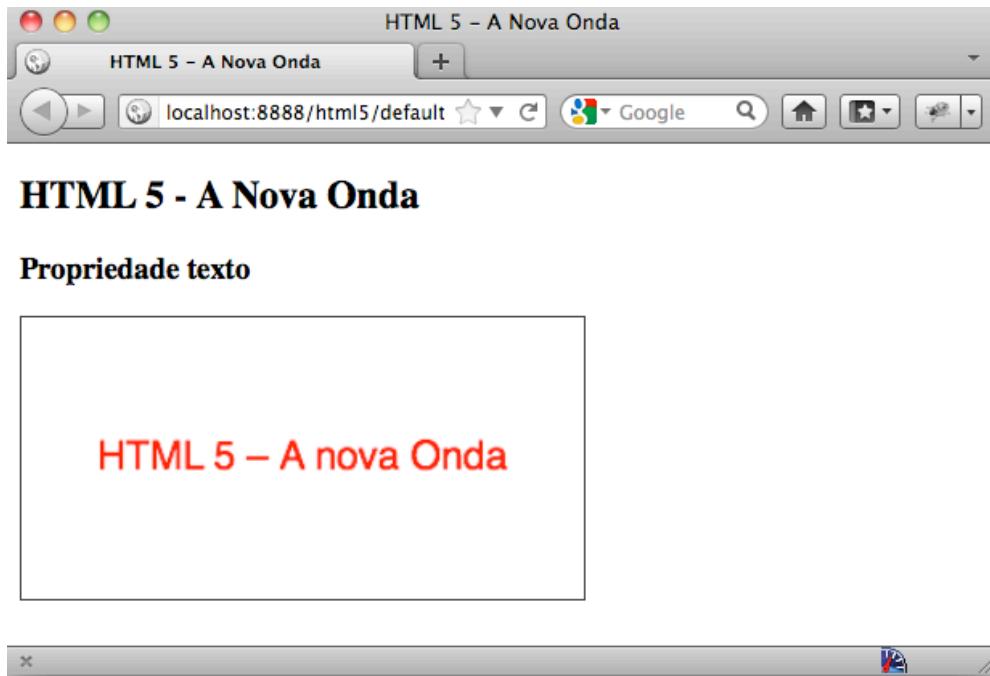
Exemplo :

HTML

```
...
<section>
    <h1>HTML 5 - A Nova Onda</h1>
    <h3>Elemento Texto</h3>
    <canvas id="cv"></canvas>
</section>
...
```

Javascript

```
...
<script>
    function desenhar() {
        var canvas = document.querySelector('canvas');
        var ctx = canvas.getContext('2d');
        var txt = "HTML 5 - A nova Onda";
        var wdt = canvas.width/2;
        var hgt = canvas.height/2;
        ctx.font = "21px Helvetica";
        ctx.fillStyle = "#F00";
        ctx.textAlign = "center";
        ctx.textBaseline = "middle";
        ctx.fillText(txt, wdt, hgt);
    }
    window.onload = desenhar;
</script>
...
```



Padrão

A API de *canvas* permite a criação de padrões (*pattern*) para ser aplicado no objeto de desenho. Um padrão só poderá ser aplicado em um contorno ou preenchimento de um objeto desenhado no *canvas*.

Para criar um padrão no *canvas* usamos o método *createPattern*, que possui dois parâmetros de configuração: imagem, opção de repetição do *pattern*.

Exemplo :

HTML

```
...
<section>
    <h1>HTML 5 - A Nova Onda</h1>
    <h3>Elemento Padrão</h3>
    <canvas id="cv"></canvas>
</section>
...
```

Javascript

```
...
<script>
    function desenhar(){
        var canvas = document.querySelector('canvas');
        var ctx = canvas.getContext('2d');
        var offsetX = canvas.width - 20;
        var offsetY = canvas.height - 20;
        var imagem = new Image();
        imagem.src = 'api canvas/bob.png';
```

```

        imagem.onload = function() {
            var pattern = ctx.createPattern(imagem, 'repeat-x');
            ctx.rect(10, 10, offsetX, offsetY);
            ctx.fillStyle = pattern;
            ctx.fill();
        }
        window.onload = desenhar;
    </script>
    ...

```



Para aplicar um padrão (*pattern*) em um contorno lembre-se de configurar uma largura adequada, que permita a visualização do mesmo.

Exemplo :

Javascript

```

...
<script>
    function desenhar() {
        var canvas = document.querySelector('canvas');
        var ctx = canvas.getContext('2d');
        var offsetX = canvas.width - 20;
        var offsetY = canvas.height - 20;
        var imagem = new Image();
        imagem.src = 'api canvas/bob.png';
        ctx.lineWidth = 15;
        imagem.onload = function() {
            var pattern = ctx.createPattern(imagem, 'repeat-x');
            ctx.rect(10, 10, offsetX, offsetY);
            ctx.strokeStyle = pattern;
            ctx.stroke();
        }
        window.onload = desenhar;
    </script>
    ...

```

Capítulo 08: API Drag and Drop

* Visão Geral	86
* Suporte dos Browsers	86
* Atributo draggable	86
* Método dataTransfer	87
* Propriedades	87
* Eventos	88

Visão Geral

Um dos objetivos das aplicações denominadas Web 2.0, é tornar a interação do usuário tão dinâmica quanto uma aplicação *desktop* padrão. E uma das funcionalidades que permite uma interação lúdica do usuário com a aplicação é o famoso "arrastar e soltar", ou em inglês *drag and drop*.

O HTML 5 trouxe um atributo que permite configurar qualquer elemento HTML como arrastável, e um conjunto de métodos e eventos que definem a funcionalidade de todo esse processo de arrastar e soltar.

Suporte dos Browser

Até o presente momento em que este material está sendo escrito, os únicos *browsers* que não interpretam corretamente a API *Drag and Drop* são o Opera e o Firefox.

Atributo Draggable

Para poder trabalhar com os métodos e eventos da API *Drag and Drop*, é necessário tornar o elemento HTML desejado arrastável. Isso é feito definindo-se o atributo *draggable* com o valor *true* no elemento.

Exemplo :

HTML

```
...
<section>
  <h1>HTML 5 - A Nova Onda</h1>
  <h3>Elemento draggable</h3>
  <div id="drag" draggable="true"></div>
</section>
...

```

CSS

```
...
<style>
  #drag{
    width: 150px;
    height: 150px;
    background: #CCC;
  }
</style>
...

```



Método dataTransfer

Para trabalhar com a API *Drag and Drop* é necessário usar o método *dataTransfer*, que define como os dados do(s) elemento(s) serão interpretados no momento em que ocorre o arraste.

Esse método deve ser definido no evento que inicia o arraste do(s) elemento(s), caso contrário, não serão interpretados pela API os resultados programados em cada estágio do processo de arrastar e soltar.

Propriedades

Como complemento do método *dataTransfer*, existe um conjunto de propriedades para a API *Drag and Drop* que definem o formato e os efeitos permitidos dos dados que estão sendo arrastados.

effectAllowed

Essa propriedade define qual é o efeito permitido para o objeto que está sendo arrastado. As configurações permitidas são:

- ✓ *None*
- ✓ *Copy*
- ✓ *Link*
- ✓ *Move*

setData

Propriedade que configura o formato de dado que será transferido no processo de arraste, usando o conceito de chave/valor.

getData

A propriedade *getData* retorna o valor do dado definido com *setData*.

setDragImage

Essa propriedade retorna os dados de posicionamento atualizados do objeto arrastado, em relação a sua posição inicial. Possui os seguintes parâmetros: elemento arrastado, posição x e posição y.

Eventos

Em complemento aos métodos e propriedades, a API *Drag and Drop* possui um conjunto de manipuladores de eventos específicos para esse processo.

dragStart

Evento disparado no momento em que inicia o arraste do objeto configurado.

dragEnter

Esse evento é disparado quando o mouse toca a área limite do objeto-alvo, enquanto ocorre o arraste.

dragOver

Esse evento é disparado quando o mouse está sobre o objeto-alvo, enquanto ocorre o arraste.

dragLeave

O evento *dragLeave* é disparado quando o mouse sai da área limite do objeto-alvo, enquanto ocorre o arraste.

drag

Esse evento é disparado quando o mouse é movido com o objeto de arraste.

drop

Esse evento é disparado no momento em que é finalizado o arraste do objeto, sobre o objeto-alvo.

dragEnd

Esse evento é disparado quando o usuário libera o botão do mouse, enquanto ocorre o processo de arraste. Ele finaliza o processo de arraste instantaneamente.

Exemplo :**HTML**

```
...
<div id="principal">
  <section id="home" ondragenter="return dragEnter(event)"
  ondragover="return dragOver(event)" ondrop="return dragDrop(event)">
    <header>
      <h1>Exemplo de Drag and Drop HTML 5</h1>
      <h3>Arraste o Bob Esponja para os boxes abaixo e de volta para o
      box principal.</h3>
    </header>
    
    <br><br>
    <p></p>
  </section>
<br><br>
<div id="drop1" ondragenter="return dragEnter(event)" ondragover="
  return dragOver(event)" ondrop="return dragDrop(event)"></div>
<div id="drop2" ondragenter="return dragEnter(event)" ondragover="
  return dragOver(event)" ondrop="return dragDrop(event)"></div>
</div>...
```

CSS

```
...
<style>
  body {background:#b4d7fa;}
  #drop1, #drop2 {
    background:#ffffff;
    float:left;
    margin-left:150px;
    padding:15px;
    height:200px;
    width:200px;
    border:solid #40464d 3px;
    color:#69C;
    border-radius: 10px;
    box-shadow: 10px 5px 10px #666;
  }
  div {
    width: 800px;
    margin: 0 auto;
    text-align: center;
  }
  img{
    margin: 0 auto;
    width: 136px;
  }
  section{
    border: 1px dashed #336;
    padding: 25px;
    position: relative;
  }
</style>
```

```

h1{
    width: 400px;
    margin: 0 auto;
}

p{
    margin: 0 auto;
    font: normal 15px Arial, Helvetica;
    width: 400px;
}
</style>
...

```

Javascript

```

...
<script>
    function dragStart(ev) {
        //Definindo que tipo de efeito será permitido no evento de arraste
        ev.dataTransfer.effectAllowed='copy';
        //Configurando o tipo de dado arrastável e pegando seu ID
        ev.dataTransfer.setData("Text", ev.target.getAttribute('id'));
        return true;
    }

    function dragEnter(ev) {
        //Evitando a ação de link sobre a DIV
        ev.preventDefault();
        return true;
    }

    function dragOver(ev) {
        ev.preventDefault();
        return false;
    }

    function dragEnd(ev) {
        ev.preventDefault();
        return false;
    }

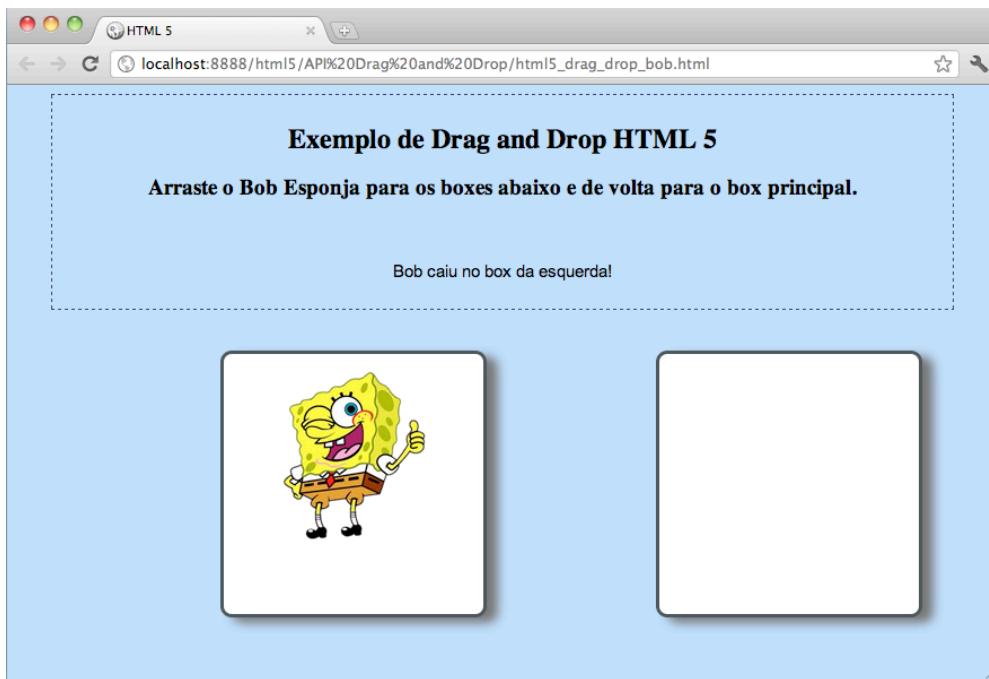
    function dragDrop(ev) {
        ev.preventDefault();
        //Pegando o elemento definido como arrastável na função ondragstart
        var elemento = ev.dataTransfer.getData("Text");
        //Anexando o elemento arrastável como conteúdo da DIV sobre o qual ele foi solto
    }

```

```

ev.target.appendChild(document.getElementById(elemento));
//Captura o parágrafo que leva a mensagem
var p = document.getElementsByTagName('p')[0];
var msg;
//Verifica qual foi o container que recebeu o elemento
arrastável
switch(ev.target.id){
    case 'drop1':
        msg = 'Bob caiu no box da esquerda!';
        break;
    case 'drop2':
        msg = 'Bob caiu no box da direita!';
        break;
    case 'home':
        msg = 'Bob voltou!';
        break;
}
//Escreve a mensagem no parágrafo
p.innerHTML = msg;
return true;
}
</script>
...

```



É possível substituir a chamada dos eventos diretamente colocados no objeto HTML, usando a assinatura de *listener*.

Capítulo 09: API Geolocalização

* Visão Geral.....	93
* Suporte dos Browsers	93
* Métodos	93
* Propriedades de posicionamento	94
* Tratamento de Erros	96
* API Google Maps	98

Visão Geral

Com a Web 2.0 e os dispositivos móveis, um dos recursos que ganhou popularidade foi a Geolocalização. Esse recurso consiste em usar coordenadas de latitude e longitude para compartilhar o posicionamento geográfico de pessoas e objetos, usando como recursos endereços de IP, redes sem fio (*wireless*) e GPS (Global Positioning System).

Atualmente, o recurso mais utilizado é o GPS, que calcula as coordenadas de latitude e longitude via atualização por satélite, e está presente no cotidiano da maioria das pessoas que vivem em grandes metrópois.

O HTML 5 traz em sua configuração uma API de Geolocalização, que permite a identificação das coordenadas de latitude e longitude, efetuando o compartilhamento do posicionamento geográfico do usuário, desde que, o mesmo, aceite esse compartilhamento dando sua permissão.

Suporte dos Browsers

Maior parte dos navegadores estão preparados para renderizar nativamente a API de Geolocalização HTML 5. As versões que aceitam essa característica são:

- ✓ IE 9.0
- ✓ Firefox 3.5+
- ✓ Chrome 5.0+
- ✓ Safari 5.0+
- ✓ Opera 10.6+

Métodos

A API traz um conjunto de métodos que permite a manipulação das principais propriedades e eventos, relacionados ao compartilhamento de posicionamento do usuário. Esses métodos são:

getCurrentPosition

Esse método realiza a captura do posicionamento atual do usuário, retornando um objeto chamado *position* com as propriedades de latitude e longitude, caso haja sucesso, e um objeto chamado *err* com os tipos de erros ocorridos, caso não seja possível a identificação correta das coordenadas. Esses retornos são feitos por funções chamadas de *callback*.

watchPosition

A diferença do método anterior para o *watchPosition*, é que este realiza a captura das coordenadas de posicionamento do usuário de modo continuo. Portanto, se o foco da aplicação de geoposicionamento for um dispositivo móvel, esse é o melhor método a ser utilizado.

clearWatch

Esse método permite que pare a atualização continua feita pelo método *watchPosition*. Para o funcionamento desse método é necessário que se passe o número retornado pelo método *watchPosition*. É possivel também trabalhar com a função *setInterval* do Javascript.

Propriedades de posicionamento

O objeto *position* retornado pelos métodos da API, trabalha com um conjunto de propriedades que permite aprimorar o processo de posicionamento do usuário. As principais propriedades são:

latitude

Essa propriedade retorna um valor em graus decimais relativos ao posicionamento sobre um eixo x imaginário.

longitude

Essa propriedade retorna um valor em graus decimais relativos ao posicionamento sobre um eixo y imaginário.

altitude

Retorna um valor referente a altura do posicionamento medido em metros.

altitudeAccuracy

Fornece um valor em metros, com precisão, da coordenada geográfica de altitude da API. Se não for possível determinar a precisão da altitude será retornado um valor *null*.

accuracy

Essa propriedade refere-se a precisão. É medida em metros, e está relacionada com as coodenadas de latitude e longitude de posicionamento.

heading

Essa propriedade define a direção do deslocamento realizado pelo usuário. Retorna o valor de um ângulo que varia de 0 a 360 graus no sentido horário. Se não for possível detectar a direção, o valor retornado será *null*.

speed

Propriedade que fornece a velocidade de deslocamento do usuário, medida em metros por segundo. Caso não seja possível realizar essa medição, o retorno é um valor *null*.

timestamp

Essa propriedade retorna um valor numérico em milisegundos, que representa o tempo exato em que foi capturado o posicionamento do usuário.

Obs.: Tanto as coordenadas de latitude e longitude, como as propriedades citadas acima trazem valores aproximados e com precisão acima de 95%, mas não de 100%.

Portanto, como a própria descrição técnica da API informa, o resultado não é totalmente preciso.

Exemplo :**HTML**

```
...
<section>
    <h3>Elemento Geolocation</h3>
    <button>pegando coordenadas do browser</button>
    <br>
    <p id='log'></p>
</section>
...
```

Javascript

```
...
<script src="modernizr.js"></script>
<script>
    window.onload = function(){
        var bt = document.getElementsByTagName('button')[0];
        var box = document.getElementById('log');
        var msg;
        if(!Modernizr.geolocation){
            alert('Seu browser não aceita Geolocalização!');
        }
        else{
            bt.addEventListener('click',pegaCoordenadas,false);
        }
        function pegaCoordenadas(){
            navigator.geolocation.getCurrentPosition(sucesso);
            function sucesso(position){
                var lat = position.coords.latitude;
                var long = position.coords.longitude;
                var alt = position.coords.altitude;
                var altPrec = position.coords.altitudeAccuracy;
                var prec = position.coords.accuracy;
                var dir = position.coords.heading;
                var veloc = position.coords.speed;
                var temp = position.coords.timestamp;
                var msg = 'Suas coordenadas são:<br><br>';
                msg += '<strong>Latitude</strong>: '+lat+'<br>';
                msg += '<strong>Longitude</strong>: '+long+'<br>';
                msg += '<strong>Altitude</strong>: '+alt+'<br>';
                msg += '<strong>Altitude precisa</strong>: '+altPrec+'<br>';
                msg += '<strong>Precisão</strong>: '+prec+'<br>';
                msg += '<strong>Direção</strong>: '+dir+'<br>';
                msg += '<strong>Velocidade</strong>: '+veloc+'<br>';
                msg += '<strong>Tempo</strong>: '+temp+'<br>';
                box.innerHTML = msg;
            };
        };
    };
</script>
...
```



Elemento Geolocation HTML 5

`pegando coordenadas do browser`

Suas coordenadas são:

Latitude: -23.548943
Longitude: -46.638818
Altitude: null
Altitude precisa: null
Precisão: 140000
Direção: null
Velocidade: null
Tempo: undefined

Tratamento de erros

É importante no desenvolvimento de uma aplicação *web* o tratamento de erros, possibilitando um retorno para o usuário em virtude de alguma instabilidade ocorrida.

A API de geolocalização possui um conjunto de códigos para tratamento dos principais erros ocorridos na captura de posicionamento do usuário.

A função *callback* de erro retorna um código referente ao problema ocorrido, e que pode ser interpretado através do atributo *code*.

Os tipos de erros e seus respectivos códigos são:

permission_denied (1)

Esse erro ocorre quando o usuário não permite que seja compartilhado seu posicionamento pelo dispositivo de captura.

position_unavailable (2)

Erro ocorrido quando não é possível determinar o posicionamento do usuário, ou por problemas no processo de conexão, ou pela rede estar fora do ar.

timeout (3)

Esse erro ocorre quando o tempo para localização da posição do usuário excede o tempo limite estabelecido pela conexão.

Exemplo :

HTML

```
...
<section>
    <h3>Elemento Geolocation</h3>
    <button>pegando coordenadas do browser</button>
    <br>
    <p id='log'></p>
</section>
...
```

Javascript

```
...
<script src="modernizr.js"></script>
<script>
    window.onload = function(){
        var bt = document.getElementsByTagName('button')[0];
        var box = document.getElementById('log');
        var msg;
        if(!Modernizr.geolocation){
            alert('Seu browser não aceita Geolocalização!');
        }
        else{
            bt.addEventListener('click',pegaCoordenadas,false);
        }
        function pegaCoordenadas(){
            navigator.geolocation.getCurrentPosition(erro);
            function erro(err){
                switch(err.code){
                    case 1:
                        msg = 'Permissão para obter posição negada!';
                        break;
                    case 2:
                        msg = 'Erro de conexão!';
                        break;
                    case 3:
                        msg = 'Tempo de conexão esgotado!';
                        break;
                    default:
                        msg = 'Não foi possível obter os dados!';
                        break;
                }
                box.innerHTML = '<strong>' +msg+ '</strong>';
            }
        };
    };
</script>
...
```

Para testar o efeito do tratamento de erro da API de geolocalização negue a permissão de compartilhamento de posição na solicitação do navegador.

Os outros códigos de erros dependem de recursos da rede de conexão que não são administráveis para teste.

API Google Maps

Apesar da API de geolocalização do HTML 5 permitir uma facilidade na obtenção do posicionamento geográfico do usuário, o maior atrativo está na visualização gráfica desses resultados.

Para isso, o Google desenvolveu uma API chamada de Maps que retorna mapas visuais estáticos e dinâmicos, relacionados as coordenadas de posicionamento do usuário, e que podem ser integradas com a API de geolocalização do HTML 5.

Vamos estudar dois exemplos de integração, um estático e outro dinâmico, pois, a API Maps do Google é extensa e com vários de recursos.

Para quem desejar um maior conhecimento sobre a API Maps segue o link oficial abaixo:
<http://code.google.com/intl/pt-BR/apis/maps/>

Mapas estáticos

Os mapas resultantes dessa integração não possuem controles dinâmicos de *zoom*, redimensionamento e etc. A visualização do mapa de posicionamento é totalmente estática.

Exemplo :

HTML

```
...
<section>
    <h3>Elemento Geolocation</h3>
    <button>pegando coordenadas do browser</button>
    <br>
    <p id='log'></p>
    <br>
    <img id="mapa" >
</section>
...

```

Javascript

```
...
<script src="modernizr.js"></script>
<script>
    window.onload = function(){
        var bt = document.getElementsByTagName('button')[0];
        var box = document.getElementById('log');
        var msg;
        if(!Modernizr.geolocation){
            alert('Seu browser não aceita Geolocalização!');
        }
        else{
            bt.addEventListener('click', pegaCoordenadas, false);
        }
        function pegaCoordenadas(){
            navigator.geolocation.getCurrentPosition(erro, sucesso);
        }
    }

```

```

function erro(err) {
    switch(err.code) {
        case 1:
            msg = 'Permissão para obter posição negada!';
            break;
        case 2:
            msg = 'Erro de conexão!';
            break;
        case 3:
            msg = 'Tempo de conexão esgotado!';
            break;
        default:
            msg = 'Não foi possível obter os dados!';
            break;
    }
    box.innerHTML = '<strong>' + msg + '</strong>';
}
};//Fim da func erro

function sucesso(position) {
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;
    var mapa = document.getElementById('mapa');
    msg = 'Suas coordenadas são:<br /><br />';
    msg += '<strong>Latitude</strong>: ' + latitude + '<br />';
    msg += '<strong>Longitude</strong>: ' + longitude + '<br />';
    box.innerHTML = msg;
    //Definições da API Google Maps
    var mapa = document.getElementById('mapa');
    var url = "http://maps.google.com/maps/api/staticmap?center=";
    url += latitude + "," + longitude;
    url += "&zoom=15&size=520x520";
    url += "&markers=color:orange|" + latitude + "," + longitude;
    url += "&sensor=true"
    mapa.src = url;
    mapa.style.display = "block";
};//fim da func sucesso
};

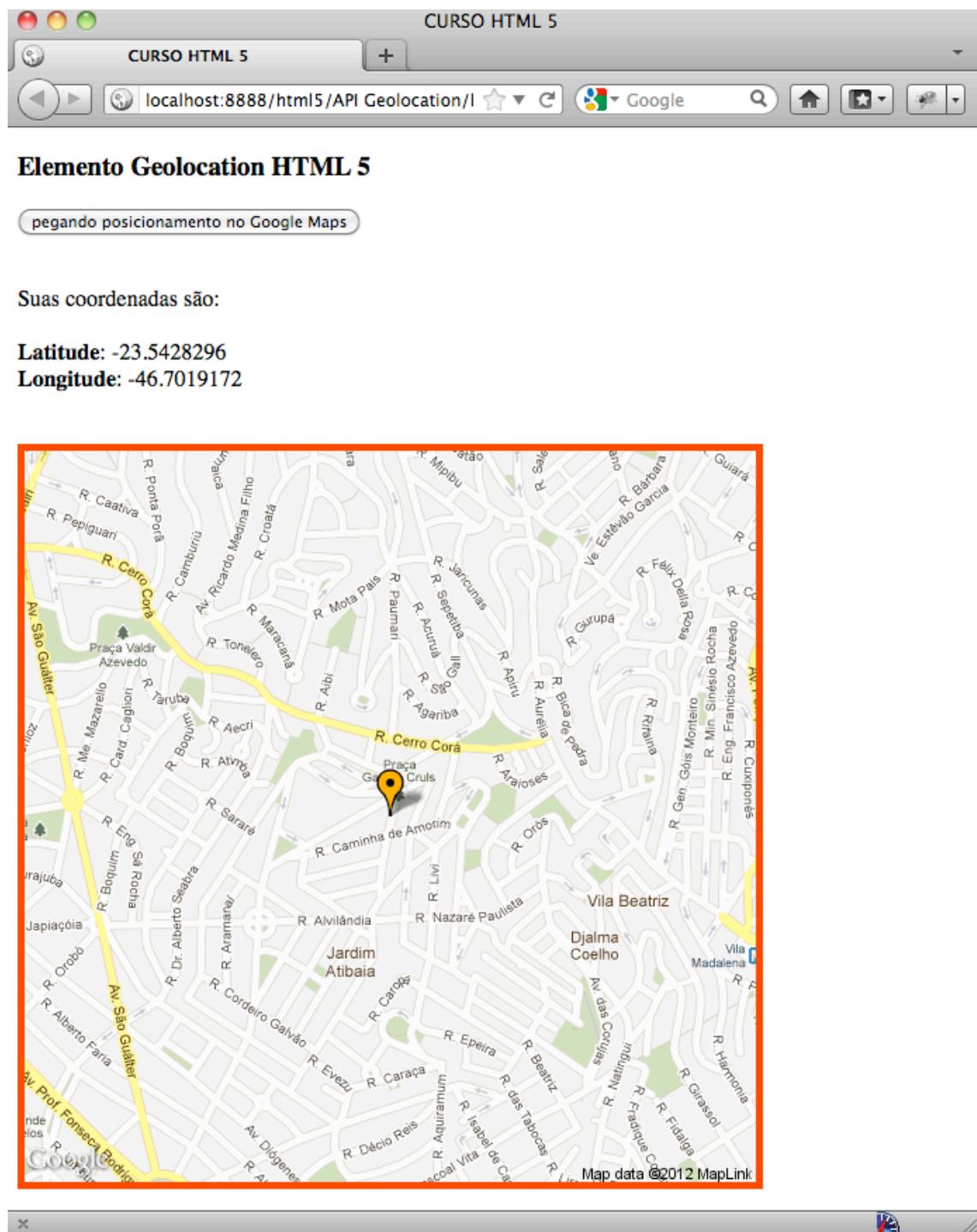
</script>
...

```

Para a integração com API Maps do Google é necessário referenciar o *link* que define os recursos do tipo de mapa desejado, nesse caso o estático.

Para esse tipo de mapa, os recursos de configuração são todos especificados como atributos no próprio *link* de acesso.

- ✓ **Zoom** = Define amplitude da visualização do mapa na tela do navegador.
- ✓ **Size** = Medidas referentes a largura e altura física da imagem do mapa.
- ✓ **Markers** = Marcador visual (pin) da posição geográfica solicitada.
- ✓ **Sensor** = Atributo que define a sensibilidade da atualização da posição geográfica solicitada.



Como a API Maps foi desenvolvida pela equipe do Google, todos os parâmetros especificados e qualquer tipo de alteração devem seguir o formato especificado pelo Google.

Caso contrário, o resultado da API pode ser insatisfatório ou nulo.

Mapas dinâmicos

Ao contrário do exemplo anterior, o mapa dinâmico permite o trabalho com recursos interativos de *zoom* e redimensionamento.

Exemplo :

HTML

```
...
<section>
    <h3>Elemento Geolocation</h3>
    <button>pegando coordenadas do browser</button>
    <br>
    <p id='log'></p>
    <br>
    <img id="mapa" >
</section>
...
```

CSS

```
...
<style>
    #mapa{
        width:550px;
        height:350px;
        display:none;
        border: 5px solid #F30;
        overflow: hidden;
    }
</style>
...
```

Javascript

```
...
<script src="http://maps.google.com/maps/api/js?sensor=true"></script>
<script src="modernizr.js"></script>
<script>
    window.onload = function(){
        var bt = document.getElementsByTagName('button')[0];
        var box = document.getElementById('log');
        var msg;

        if(!Modernizr.geolocation){
            alert('Seu browser não aceita Geolocalização!');
        }
        else{
            bt.addEventListener('click', pegaCoordenadas, false);
        }
        function pegaCoordenadas(){
            navigator.geolocation.getCurrentPosition(erro,sucesso);

            function erro(err){
                switch(err.code){
                    case 1:
                        msg = 'Permissão para obter posição negada!';
                        break;
                }
            }
        }
    }
</script>
```

```

        ***
        case 2:
        msg = 'Erro de conexão!';
        break;

        case 3:
        msg = 'Tempo de conexão esgotado!';
        break;
        default:
        msg = 'Não foi possível obter os dados!';
        break;
        }
        box.innerHTML = '<strong>' +msg+ '</strong>';
    }
};//Fim da func erro

function sucesso(position) {
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;
    var mapa = document.getElementById('mapa');
    msg = 'Suas coordenadas são:<br /><br />';
    msg += '<strong>Latitude</strong>: '+latitude+'<br />';
    msg += '<strong>Longitude</strong>: '+longitude+'<br />';
    box.innerHTML = msg;
//Definições da API Google Maps
    var mapa = document.getElementById('mapa');
    var pos = new google.maps.LatLng(latitude,longitude);
    var op = {
        zoom: 15,
        center: pos,
        mapTypeId: google.maps.MapTypeId.ROADMAP
    }
    var gmap = new google.maps.Map(mapa, op);
    var image = 'pin.png';
    new google.maps.Marker({
        position: pos,
        map: gmap,
        icon: image
    });
};//fim da func sucesso
};

</script>
...

```

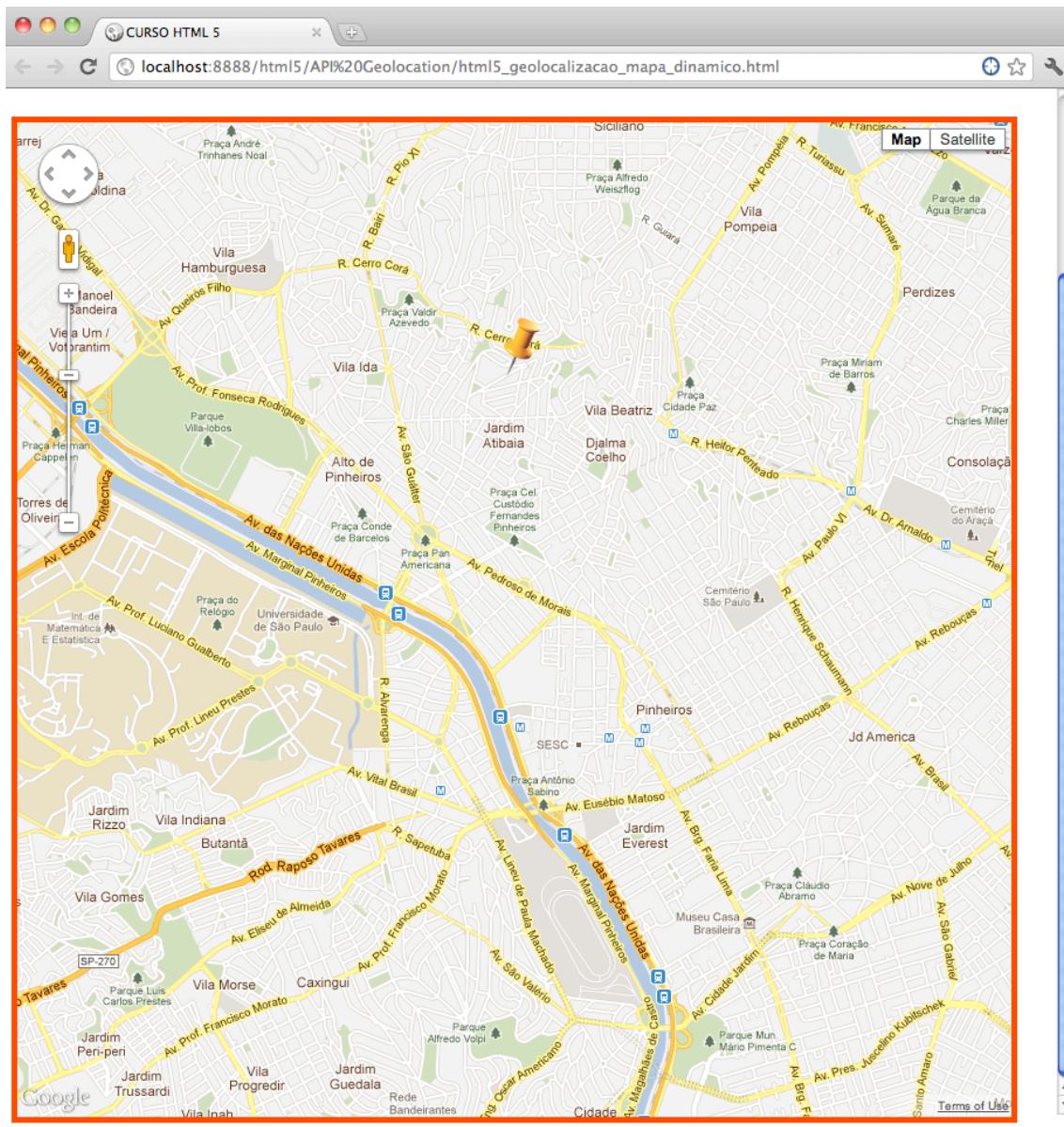
Como dito anteriormente, a integração com esse tipo de mapa dinâmico possui configurações específicas, e sua codificação deve seguir as orientações descritas nas instruções da API.

Nesse código foi usado o objeto *JSON(Javascript Object Notation)* para definição das opções do mapa.

Foi criado também um objeto *Marker* para definir o marcador de posicionamento no mapa, com o posicionamento das coordenadas de latitude e longitude, desenho do mapa e imagem do marcador. A chamada para o endereço da API desse mapa está referenciada agora no começo do código, com a tag *<script>*.

No código do exemplo anterior, essas configurações estavam todas definidas na própria *url* da API. Agora estão mais estruturadas no corpo do código.





O mapa acima possui controles dinâmicos que podem ser acionados através do mouse, ou com a barra lateral esquerda.

Como já foi dito, existem outras possibilidades de renderização de mapas na API Maps do Google, inclusive a geocodificação reversa, onde digita-se um endereço e o mesmo é convertido em posicionamento visual no mapa. Consulte a API para mais informações.

* * *

Capítulo 10: API Armazenamento Local

* Visão Geral.....	105
* Suporte dos Browsers.....	105
* Métodos.....	106
* Propriedades.....	112

Visão Geral

Um problema que sempre perseguiu desenvolvedores de aplicações *web* foi a persistência de dados. As formas encontradas desde os primórdios da *internet* foram os famosos *cookies* e as sessões. O *cookie* é um arquivo de texto armazenado na máquina do usuário, e que permite a gravação e utilização de dados durante a navegação pela aplicação *web*.

Porém, ele possui uma série de limitações que faz com que seu uso seja restrito, ou mesmo evitado em determinadas situações de desenvolvimento.

São elas:

- ✓ Capacidade de armazenamento de dados em média de 45kb por *cookie*.
- ✓ Armazenamento em torno de 20 *cookies* por domínio, e 300 cookies por navegador.
- ✓ Os *cookies* são enviados pelo protocolo *HTTP* a cada requisição, o que gera em algumas situações, um tráfego de rede desnecessário.
- ✓ Por se tratar de um arquivo de texto puro, *cookies* não são seguros para informações sigilosas, pois estão sujeitos a ataques maliciosos.

Para contornar essas limitações o HTML 5 trouxe a API *Web Storage*, com dois tipos de objetos similares ao *cookie*, e que foram projetados para armazenar dados estruturados no cliente *web*.

Suporte dos Browsers

A API *Web Storage* é uma das características HTML 5 que está amplamente adaptada pelos principais *browsers* do mercado.

- ✓ Google Chrome 3.0+
- ✓ Safari 4.0+
- ✓ Firefox 3.0+
- ✓ Opera 10.5+
- ✓ IE 8.0+

Apesar de ser suportada amplamente pelos navegadores atuais, é uma boa prática detectar se o agente do usuário está apto a renderizar as características da API.

Exemplo :

Javascript

```
...
<script src="modernizr.js"></script>
<script>
    window.onload = function() {
        if(!Modernizr.localStorage) {
            alert('Seu browser não aceita a API Web Storage!');
        }
        else{
            ...código da API
        }
    }
</script>
...
***
```

Métodos

A API *Web Storage* traz dois métodos que vão trabalhar com persistência de dados na aplicação *web*, porém, com focos diferentes no modo de atuar.

São eles: *localStorage* e *sessionStorage*.

localStorage

Esse método foi criado com o objetivo de armazenar e manipular dados entre as janelas ou abas do navegador do usuário. Durante a transação na aplicação *web* é possível compartilhar os dados do usuário, entre as janelas ou abas do navegador. Os dados são mantidos armazenados localmente pelo *browser*, e serão excluídos somente pela propriedade *removeItem*.

Exemplo:

HTML

```
...
<section>
  <header>
    <h1>Armazenamento Local com HTML 5</h1>
  </header>
  <div id="main">
    <form id="frm">
      <p>
        <h4>Você acha o Barcelona o melhor time do mundo?</h4>
        <input type="radio" name="quiz" value="s">&ampnbspSim<br>
        <input type="radio" name="quiz" value="n">&ampnbspNão<br>
        <input type="radio" name="quiz" value="x">&ampnbspNão curte
          Futebol<br><br>
        <button id="votar">&ampnbspvotar&ampnbsp</button>
      </p>
    </form>
    
  </div>
  <div id="resultado">
    <label>Sim: </label>
    <br>
    <label>Não: </label><br>
    <label>Não curte futebol: </label>
  </div>
  <span id="resposta" class="erro"></span>
</section>
...
```

CSS

```
...
<style>
  #main{
    border: 3px solid #F60;
    margin: 0 auto;
    height: 250px;
    width: 550px;
    padding: 10px;
  }

```

```

header{
    display:block;
    width: 450px;
    margin: 0 auto;
}
#frm{
    margin-left: 210px;
    width: 300px;
    display: block;
}
p{margin-top: 65px;}
#rotulo{
    position: absolute;
    margin: -300px 0 0 -10px;
    z-index: -1;
}
.erro{
    font: bold 15px Arial,Helvetica;
    color: #F00;
}
#resposta{
    position:absolute;
    margin-left: 770px;
    margin-top: -23px;
}
#resultado{
    width: 500px;
    height: auto;
    margin-left: 550px;
    position: absolute;
    display: none;
}
input[name='s'],input[name='n'],input[name='x']{width:0;}
#sim,#nao,#nocurte{
    width: 0;
    height: 7px;
}
label{font: bold 11px Arila,Helvetica;};
#sim{background: #6C6;};
#nao{background: #F00;};
#nocurte{background: #666;};
</style>
...

```

Javascript

```

...
<script>
window.onload = function(){
    var s = 0;
    var n = 0;
    var x = 0;
    var grafico = document.getElementsByTagName('img');
    var resposta = document.querySelector('#resposta');
    var radio = document.getElementsByName('input');

```

```

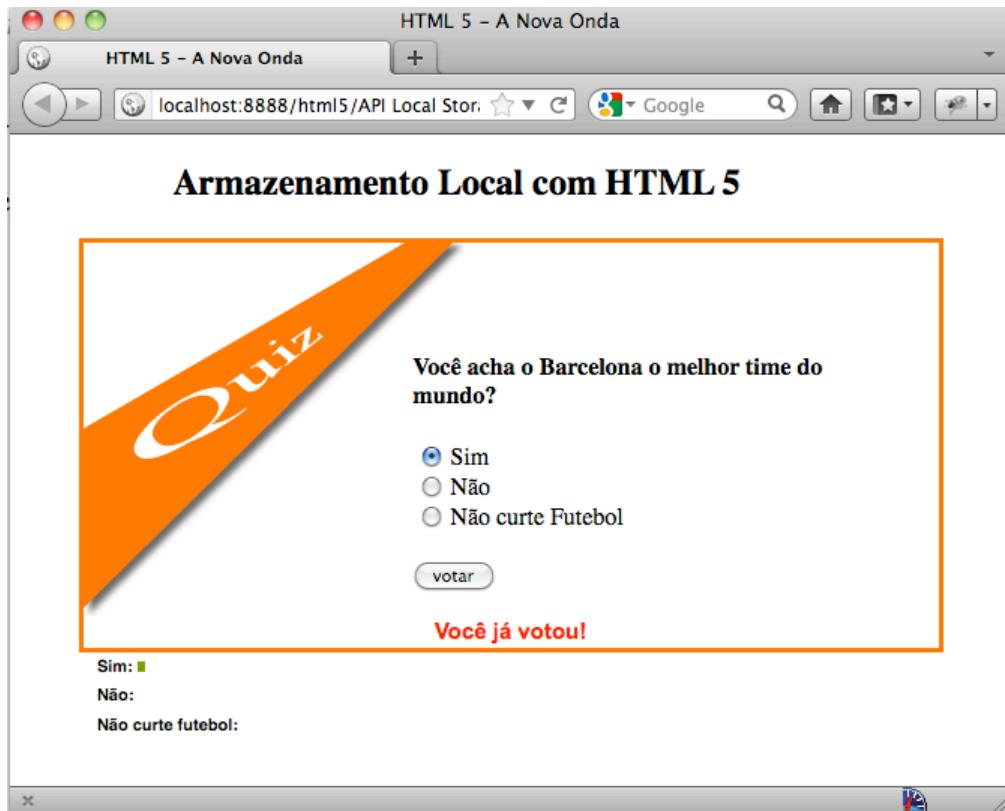
var resultado = document.querySelector('#resultado');
var btVotar = document.querySelector('#votar');
var btLimpar = document.querySelector('#limpar');
btVotar.addEventListener('click', votar, false);
function votar(e){
    e.preventDefault();
    if(!radio[0].checked && !radio[1].checked &&
       !radio[2].checked){
        resultado.style.display = 'none';
        resposta.innerHTML = 'Escolha uma opção da
                             pesquisa!';
        resposta.setAttribute('class','erro');
        return false;
    }
    if(localStorage.getItem('id') == null){
        resposta.innerHTML = '';
        for(i=0;i < radio.length;i++){
            if(radio[i].checked){
                resultado.style.display = 'block';
                localStorage.setItem('id','votou');
                switch(radio[i].value){
                    case 's':
                        s +=5;
                        grafico[1].style.width = s+'px';
                        break;
                    case 'n':
                        n +=5;
                        grafico[2].style.width = n+'px';
                        break;
                    case 'x':
                        x +=5;
                        grafico[3].style.width = x+'px';
                        break;
                }//fim do switch
            } //fim do if
        } //fim do for
    } //fim do if
    else{
        resposta.innerHTML = 'Você já votou!';
    }
};//fim da func votar
};//fim da func global
</script>
...

```

No exemplo acima, o objeto *localStorage* armazena um *id* do usuário, e se o mesmo tentar votar novamente será bloqueado.

Isso acontece porque o objeto *localStorage* armazena o *id* do usuário no *browser* após a votação, e recupera esse dado na próxima chamada do *script*, verificando que o *id* já está definido bloqueia seu acesso.

A vantagem desse tipo de armazenamento em relação ao *cookie*, é que a quantidade de dados armazenados é superior aos 64kb permitidos pelo *cookie*.



Para poder votar novamente sem o bloqueio pelo objeto localStorage, temos que usar o método `removeItem` que remove o *id* armazenado no *browser*. Para isto, vamos acrescentar um botão limpar no código usando esse método.

Exemplo:

HTML

```
...
<section>
  <header>
    <h1>Armazenamento Local com HTML 5</h1>
  </header>
  <div id="main">
    <form id="frm">
      <p>
        <h4>Você acha o Barcelona o melhor time do mundo?</h4>
        <input type="radio" name="quiz" value="s">&ampnbspSim<br>
        <input type="radio" name="quiz" value="n">&ampnbspNão<br>
        <input type="radio" name="quiz" value="x">&ampnbspNão curte
          Futebol<br><br>
        <button id="votar">&ampnbspvotar&ampnbsp</button>&ampnbsp&ampnbsp
        <button id="limpar">&ampnbsplimpar&ampnbsp</button>
      </p>
    </form>
    
  </div>
```

```

<div id="resultado">
    <label>Sim: </label>
    <br>
    <label>Não: </label><br>
    <label>Não curte futebol: </label>
</div>
<span id="resposta" class="erro"></span>
</section>
...

```

Javascript

```

...
<script>
window.onload = function() {
    var s = 0;
    var n = 0;
    var x = 0;
    var grafico = document.getElementsByTagName('img');
    var resposta = document.querySelector('#resposta');
    var radio = document.getElementsByName('input');
    var resultado = document.querySelector('#resultado');
    var btVotar = document.querySelector('#votar');
    var btLimpar = document.querySelector('#limpar');
    btVotar.addEventListener('click', votar, false);
btLimpar.addEventListener('click', limpar, false);
    function votar(e) {
        e.preventDefault();
        if(!radio[0].checked && !radio[1].checked &&
           !radio[2].checked){
            resultado.style.display = 'none';
            resposta.innerHTML = 'Escolha uma opção da
                                  pesquisa!';
            resposta.setAttribute('class','erro');
            return false;
        }
        if(localStorage.getItem('id') == null){
            resposta.innerHTML = '';
            for(i=0;i < radio.length;i++){
                if(radio[i].checked){
                    resultado.style.display = 'block';
                    localStorage.setItem('id','votou');
                    switch(radio[i].value){
                        case 's':
                            s +=5;
                            grafico[1].style.width = s+'px';
                            break;
                        case 'n':
                            n +=5;
                            grafico[2].style.width = n+'px';
                            break;
                        case 'x':
                            x +=5;
                            grafico[3].style.width = x+'px';
                            break;
                    }//fim do switch
                } //fim do if
            } //fim do for
        }
    }
}

```

```

        } //fim do if
    else{
        resposta.innerHTML = 'Você já votou!';
    }
}; //fim da func votar

function limpar(){
    localStorage.removeItem();
}
}; //fim da func global
</script>
...

```



sessionStorage

Já o método `sessionStorage` armazena e manipula dados enquanto a janela ou aba do navegador estiver ativa. No momento em que a janela ou aba é fechada, os dados são excluídos automaticamente. Os dados gerados com o método `sessionStorage` somente serão visualizados na janela ou aba em que foram criados.

Propriedades

A API possui um conjunto de propriedades para manipular os dados armazenados através dos métodos explicados anteriormente. São eles:

setItem

Essa propriedade define um dado armazenado usando o conceito de chave/valor para sua criação.

Exemplo:

Javascript

```
...
<script>
    localStorage.setItem('nome do item', 'valor do item');
</script>
...
```

getItem

Essa propriedade recupera um dado armazenado com a propriedade *setItem* através de seu nome.

Exemplo:

Javascript

```
...
<script>
    localStorage.getItem('nome do item');
</script>
...
```

removeItem

Essa propriedade exclui um dado armazenado com a propriedade *setItem* através de seu nome.

Exemplo:

Javascript

```
...
<script>
    localStorage.removeItem('nome do item');
</script>
...
```

Estas propriedades são válidas para ambos os métodos *localStorage* e *sessionStorage*.

Capítulo 11: CSS 3

* Visão Geral.....	114
* Suporte dos Browsers.....	114
* Pseudo Classes CSS 3	115
* Seletores de Atributos CSS 3	118

Visão Geral

Com a chegada do conceito de Web 2.0, o processo de desenvolvimento de aplicações ganhou um novo contorno visando tornar o ambiente do usuário mais acessível, lúdico e dinâmico.

Além disso, a tendência a mobilidade através de celulares, smartphones, tablets e laptops ampliou o raio de ação do usuário sobre o ambiente *web*.

Com isso, houve a necessidade de uma reestruturação geral no sistema de codificação, tanto em forma como em conteúdo.

Vemos a chegada do HTML 5 com uma proposta mais semântica de marcação, e também da CSS 3 como camada de estilização mais eficiente e dinâmica. Nos capítulos seguintes mostraremos as principais características desse novo nível das Folhas de Estilos, com suas aplicações práticas no dia a dia do desenvolvimento *web*.

Suporte dos Browsers

A compatibilidade dos recursos da CSS 3 ainda não é de 100% em todos os navegadores atuais.

Veja a tabela abaixo:

Característica CSS 3					
	v8	v9	v11.1	v5.1	v15
RGBA	✓	✓	✓	✓	✓
HSL	✓	✓	✓	✓	✓
Background Size	✓	✓	✓	✓	✓
Multiple Backgrounds	✓	✓	✓	✓	✓
Border Image	✓	✗	✓	✓	✓
Border Radius	✓	✓	✓	✓	✓
Box Shadow	✓	✓	✓	✓	✓
Text Shadow	✓	✗	✓	✓	✓
Opacity	✓	✓	✓	✓	✓
Animations	✓	✗	✗	✓	✓
Columns	✓	✗	✓	✓	✓
Gradiente	✓	✗	✓	✓	✓
Reflections	✗	✗	✓	✓	✓
Transforms	✓	✓	✓	✓	✓
Transitions	✓	✗	✓	✓	✓
@Font-Face	✓	✓	✓	✓	✓

Algumas características para serem aceitas pelos navegadores devem ser indentificadas pelo motor de renderização do *browser*, através do seu prefixo de fabricação, para adequar a sintaxe da característica CSS 3 aplicada.

Exemplo:

CSS

```
...
<style>
  -webkit-border-top-left-radius: 35px; //Chrome e Safari
  -moz-border-radius-topleft: 35px; //Firefox
  -o-border-top-left-radius: 35px; //Opera
  border-top-left-radius: 35px; //IE 9
</style>
...
***
```

Pseudo Classes CSS 3

O nível 3 da CSS trouxe novas pseudo classes de controle e incrementou o uso de outras pouco usadas no nível 2. São elas:

seletor:root

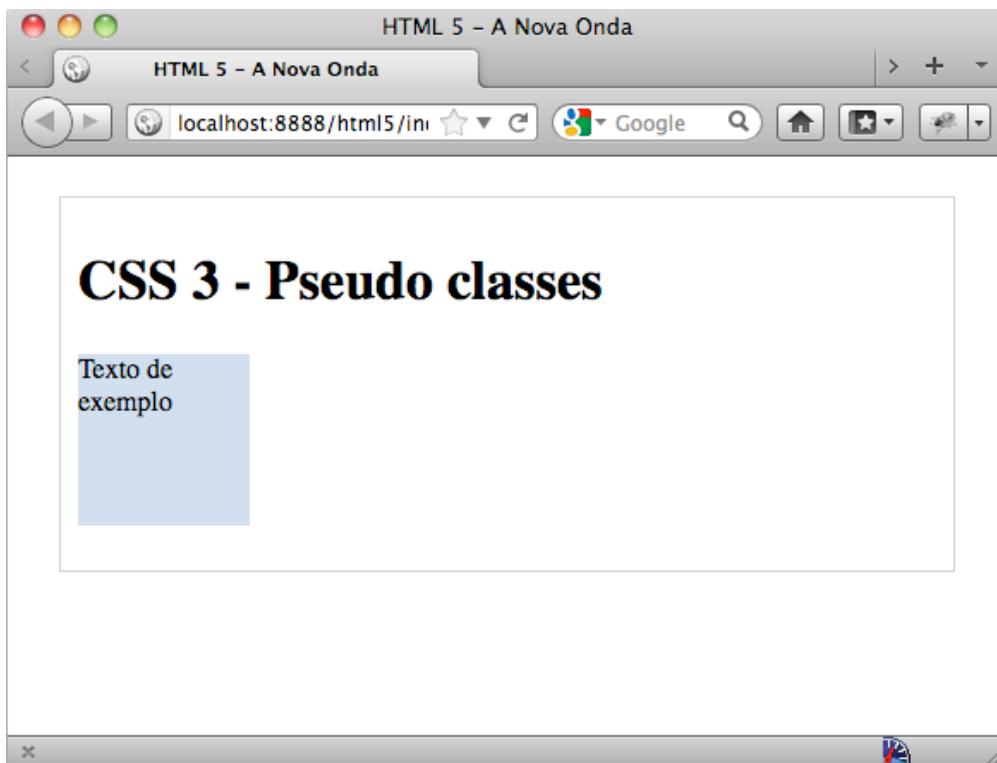
Esse seletor representa o elemento raiz (*root*) do documento, geralmente, o elemento *html*.

Exemplo:

CSS

```
...
<style>
  :root{padding: 15px; }
</style>
...
```

No código acima é aplicado um preenchimento de 15px para o elemento raiz *html*, consequentemente, para o elemento *body* também.



seletor:nth-child(n)

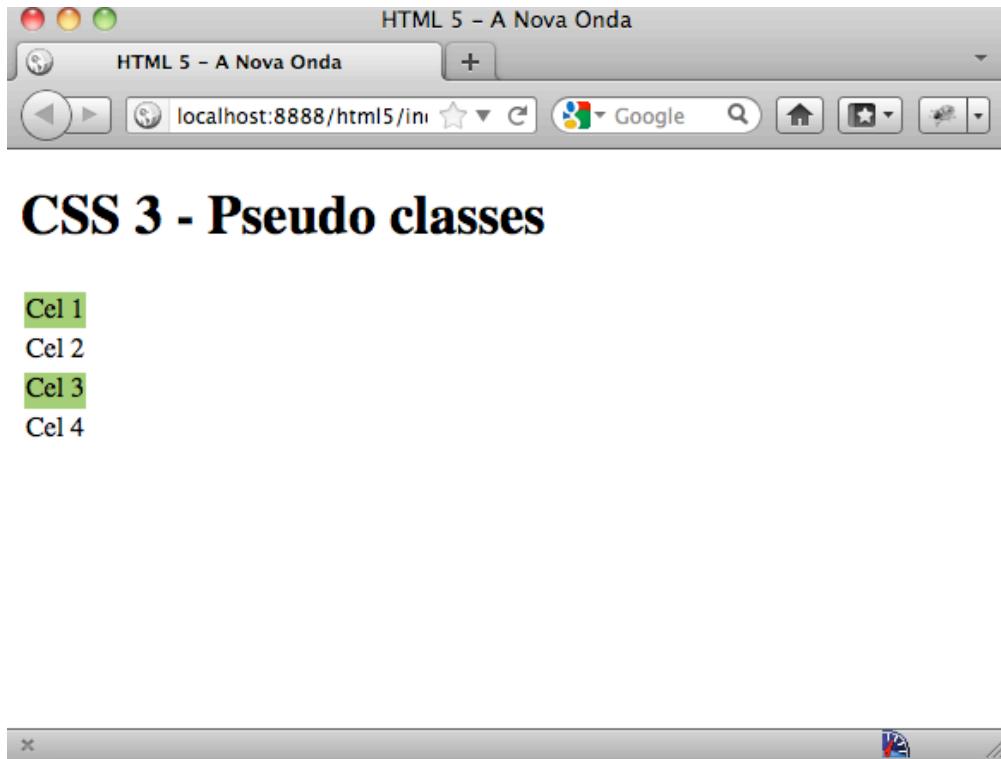
Representa um número x de elementos irmãos do seletor especificado, identificados por um índice inteiro e positivo ou de valor zero, e tenha um elemento pai.

Exemplo:

CSS

```
...
<style>
  tr:nth-child(odd){background:rgb(153,204,102);}
</style>
...
***
```

No código acima são selecionados todos os irmãos ímpares do elemento *tr* em uma tabela, e aplicada uma cor em seus fundos. É possível usar também o parâmetro *even* que representa elementos pares.



seletor:last-child

Representa um último nó filho de um seletor especificado que seja o pai desse elemento.

Exemplo:

CSS

```
...
<style>
  tr:last-child{background:rgb(153,204,102);}
</style>
...
```

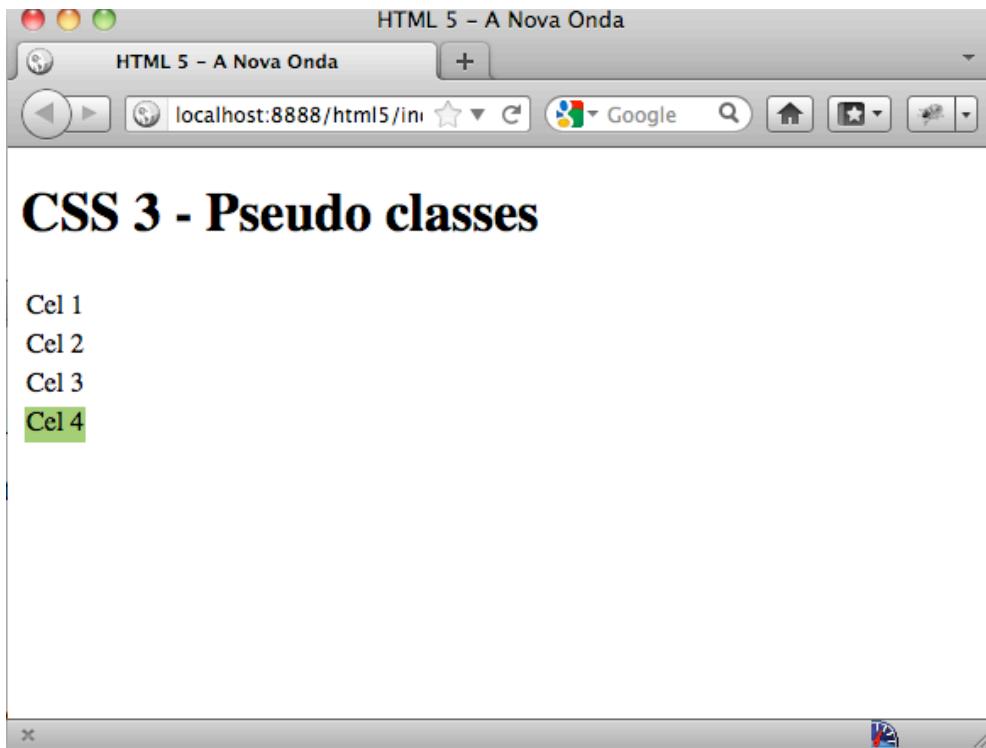
No código acima é aplicado uma cor de fundo no último nó filho do elemento *tr* em uma tabela.

Outra opção é a pseudo classe *first-child*, que acessa o primeiro nó filho do seletor especificado e que seja seu elemento pai.

Exemplo:

CSS

```
...
<style>
  tr:first-child{background:rgb(153,204,102);}
</style>
...
```



seletor:empty

Essa pseudo classe captura um elemento vazio, e sem nós filhos, no documento HTML.

Exemplo:

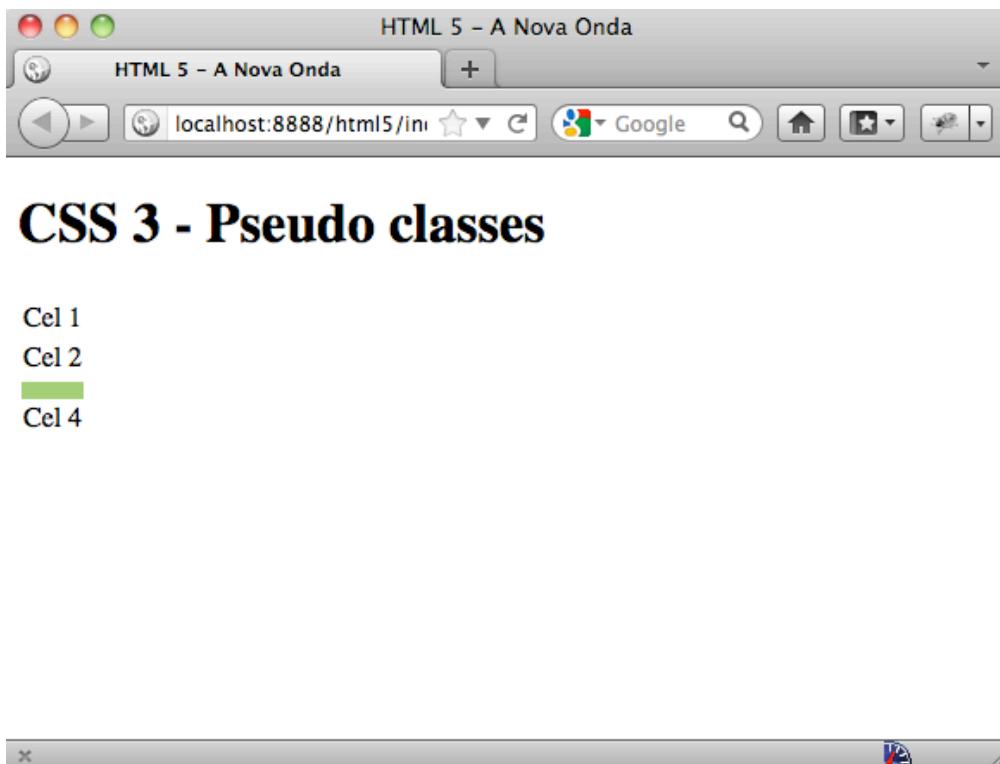
CSS

```
...
<style>
  tr:empty{
    background: #99cc66;
    height: 10px;
  }
</style>
...
```

No código acima é aplicado uma cor de fundo no elemento *tr* vazio em uma tabela.

Para que o elemento possa ser selecionado com a pseudo classe *empty*, ele deve estar totalmente vazio, inclusive sem o espaço HTML .

Caso o elemento esteja com o espaço HTML, o mesmo será considerado conteúdo e portanto o elemento não está vazio.



Seletores de Atributos CSS 3

Já utilizada no nível 2 da CSS, a captura de seletores pelos seus atributos ganhou maior evidência agora no nível 3. A captura de seletor pelo atributo permite manter o código mais limpo em relação a inserção de *id* e *class*, sem deixar de proporcionar especificidade na seleção do seletor CSS.

É possível capturar praticamente qualquer seletor, desde que o mesmo possua um atributo identificável pelo DOM.

Os principais seletores de atributos são:

seletor[attr]

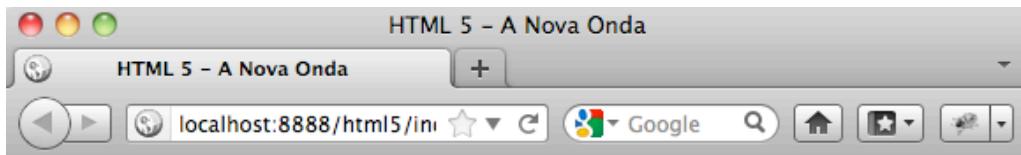
Captura o seletor pelo seu atributo especificado entre colchetes.

Exemplo:

CSS

```
...
<style>
  p[id]{
    background: rgb(153,204,102);
    height: 10px;
  }
</style>
...
```

O código acima captura todos os parágrafos no documento HTML que possuam um atributo *id* declarado, e aplica uma formatação de fundo e altura.



CSS 3 - Seletores de atributos

Primeiro parágrafo

Segundo parágrafo

Terceiro parágrafo

Quarto parágrafo



seletor[attr='x']

Captura um seletor que possua um atributo com um valor específico, indentificado entre aspas simples ou duplas.

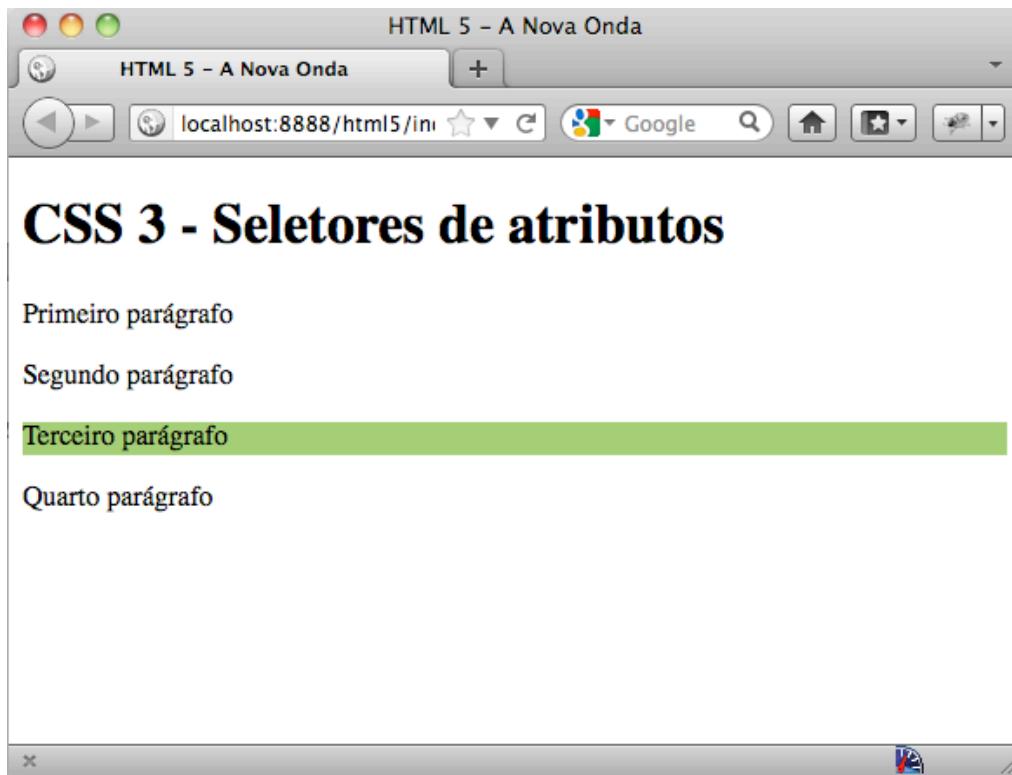
Exemplo:

CSS

```
...
<style>
  p[id='tree'] {
    background: rgb(153, 204, 102);
    height: 10px;
  }
</style>
...
```

O código acima captura o parágrafo no documento HTML que possuam um atributo *id* declarado com um valor *tree*, e aplica uma formatação de fundo e altura.

OBS.: Lembre-se de que não é permitido um atributo *id* com valores duplicados em um mesmo documento HTML.



seletor[attr~='x']

Captura um seletor com um atributo cujo valor possua uma referência ao item especificado, ou parte dele, entre aspas simples ou duplas.

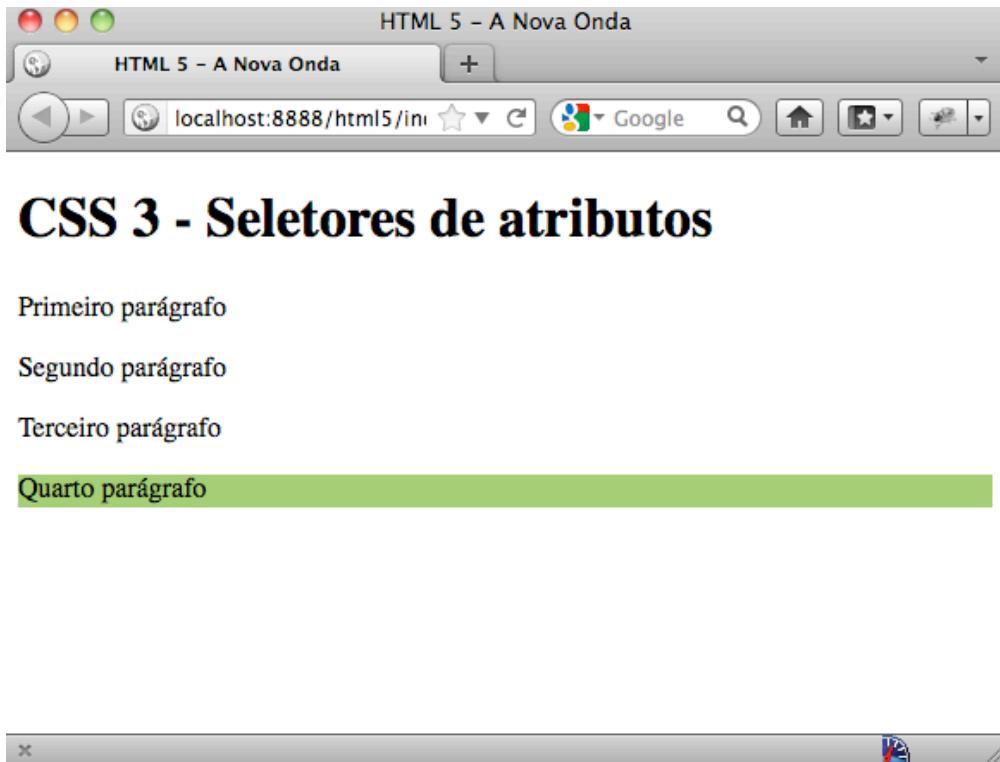
Exemplo:

CSS

```
...
<style>
  p[name~='item'] {
    background: rgb(153, 204, 102);
    height: 10px;
  }
</style>
...
```

HTML

```
...
<body>
  <h1>CSS 3 - Seletores de atributos</h1>
  <p>Primeiro parágrafo</p>
  <p>Segundo parágrafo</p>
  <p>Terceiro parágrafo</p>
  <p name='novo item'>Quarto parágrafo</p>
</body>
...
```



seletor[attr^='x']

Captura um seletor com um atributo cujo valor inicie exatamente com o item especificado entre aspas simples ou duplas.

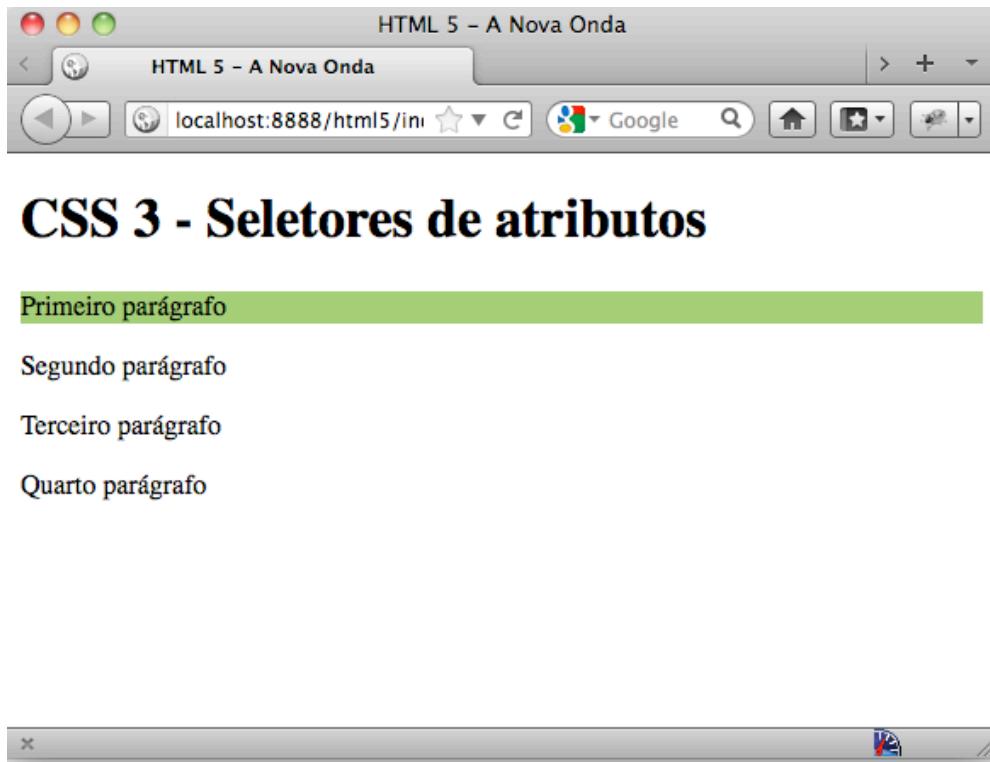
Exemplo:

CSS

```
...
<style>
  p[name^='item'] {
    background: rgb(153, 204, 102);
    height: 10px;
  }
</style>
...
```

HTML

```
...
<body>
  <h1>CSS 3 - Seletores de atributos</h1>
  <p name='item novo'>Primeiro parágrafo</p>
  <p>Segundo parágrafo</p>
  <p>Terceiro parágrafo</p>
  <p name='novo item'>Quarto parágrafo</p>
</body>
...
```



seletor[attr\$='x']

Captura um seletor com um atributo cujo valor termine exatamente com o item especificado entre aspas simples ou duplas.

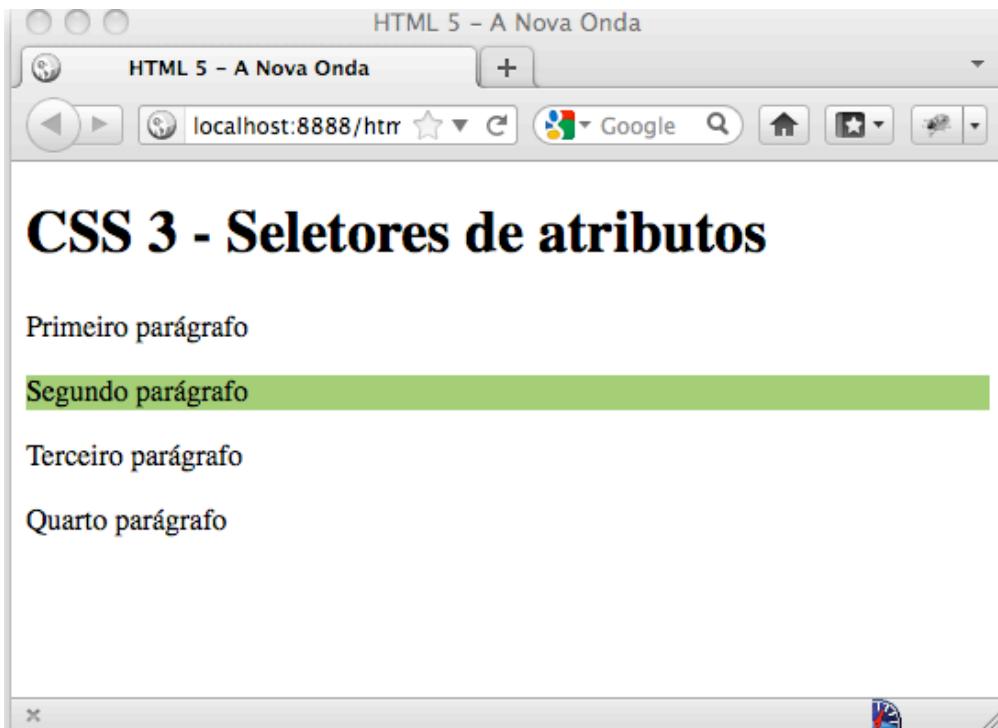
Exemplo:

CSS

```
...
<style>
  p[name$='antigo']{
    background: rgb(153,204,102);
    height: 10px;
  }
</style>
...
```

HTML

```
...
<body>
  <h1>CSS 3 - Seletores de atributos</h1>
  <p name='item novo'>Primeiro parágrafo</p>
  <p name='item antigo'>Segundo parágrafo</p>
  <p>Terceiro parágrafo</p>
  <p>Quarto parágrafo</p>
</body>
...
```



seletor[attr*='x']

Captura um seletor com um atributo que contenha qualquer valor do item especificado entre aspas simples ou duplas.

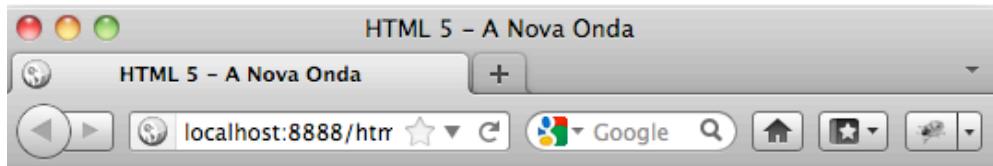
Exemplo:

CSS

```
...
<style>
  p[name*='x'] {
    background: rgb(153,204,102);
    height: 10px;
  }
</style>
...
```

HTML

```
...
<body>
  <h1>CSS 3 - Seletores de atributos</h1>
  <p name='item'>Primeiro parágrafo</p>
  <p>Segundo parágrafo</p>
  <p name='texto'>Terceiro parágrafo</p>
  <p name='info'>Quarto parágrafo</p>
</body>
...
```



CSS 3 - Seletores de atributos

Primeiro parágrafo

Segundo parágrafo

Terceiro parágrafo

Quarto parágrafo



elemento a + elemento b

Captura um elemento b que precede um elemento a.

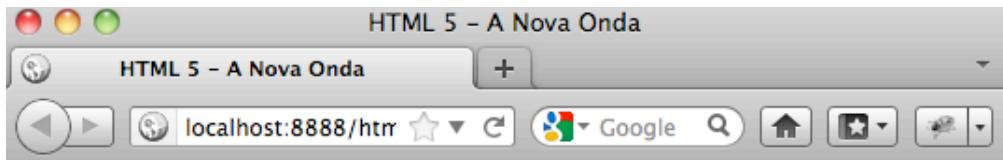
Exemplo:

CSS

```
...
<style>
  div + p{color:rgb(204,0,0);}
</style>
...
```

HTML

```
...
<body>
  <h1>CSS 3 - Seletores de atributos</h1>
  <div>
    <p name='item'>Primeiro parágrafo</p>
    <p>Segundo parágrafo</p>
    <p name='texto'>Terceiro parágrafo</p>
    <p name='info'>Quarto parágrafo</p>
  </div>
  <p>Este parágrafo está fora da div</p>
</body>
...
```



CSS 3 - Seletores de atributos

Primeiro parágrafo

Segundo parágrafo

Terceiro parágrafo

Quarto parágrafo

Este parágrafo está fora da div



elemento:enabled

Captura o elemento cujo estado é habilitado no documento HTML.

Exemplo:

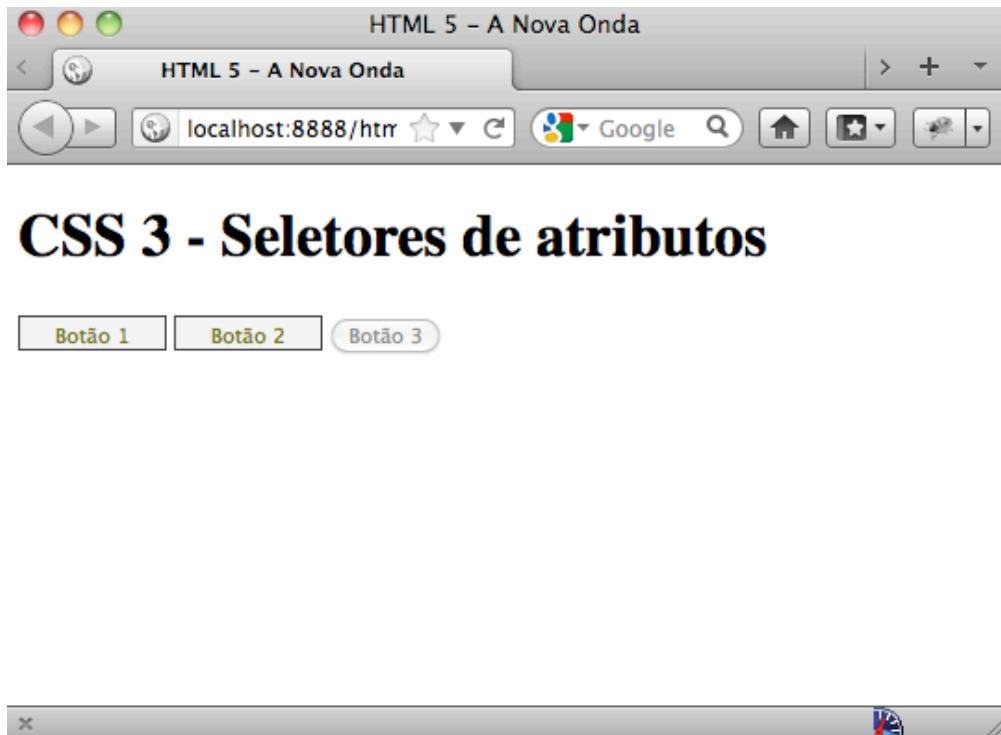
CSS

```
...
<style>
button:enabled{
    border: 1px solid #333;
    height: 19px;
    width: 80px;
    color: #666600;
}
</style>
...
```

HTML

```
...
<body>
    <h1>CSS 3 - Seletores de atributos</h1>
    <div>
        <button enabled='true'>Botão 1</button>
        <button>Botão 2</button>
        <button disabled='true'>Botão 3</button>
    </div>
</body>
...
```

OBS.: Por padrão, um elemento que não possua o atributo *enabled* declarado é considerado habilitado.



elemento:disabled

Ao contrário do item anterior, captura o elemento cujo estado é desabilitado no documento HTML.

Exemplo:

CSS

```
...
<style>
button:disabled{
    border: 1px solid #333;
    height: 19px;
    width: 80px;
    color: #666600;
}
</style>
...
```

HTML

```
...
<body>
<h1>CSS 3 - Seletores de atributos</h1>
<div>
<button enabled='true'>Botão 1</button>
<button>Botão 2</button>
<button disabled='true'>Botão 3</button>
</div>
</body>
...
```

Capítulo 12: Web Fonts

* Visão Geral	128
* Suporte dos Browsers	128
* @Font Face.....	129

Visão Geral

Um dos problemas da maioria dos desenvolvedores *web*, é adequar o design da aplicação com as fontes que não são nativas do sistema operacional do usuário. Um bom *layout* pode ir por água a baixo, se ao carregar a página o usuário não possuir a fonte original em sua máquina.

Normalmente, o sistema de renderização do *browser* gera uma mensagem informando ao usuário se deseja substituir a fonte original por outra similar, que na maioria das vezes não é a mais adequada.

Para resolver esse problema, muitos *designers* usavam uma imagem com o(s) texto(s) cuja fonte não fosse original do sistema. O problema resultante dessa tática é que os sistemas de busca não conseguem interpretar imagens, e portanto, o conteúdo do documento torna-se menos semântico para o sistema de ranqueamento dos mesmos.

A CSS 3 trouxe um novo recurso para solucionar esse impasse chamado *web fonts*.

Com esse recurso é possível armazenar as fontes desejadas no servidor *web*, e permitir que a aplicação carregue-as sem a interferência do usuários.

Suporte dos Browsers

Característica CSS 3					
	v5	v7, v8 e v9	v10	v5.1	v15
@font-face	ttf e otf	eot	ttf e otf	ttf e otf	ttf e otf

O navegador Internet Explorer reconhece apenas a fonte de extensão eot.

Os navegadores Firefox, Chrome, Safari e Opera aceitam as fontes *true type* e *open type*.

Abaixo, seguem os nomes e as abreviações dos tipos de fontes usados como *web fonts*.

Nome	Abreviatura
True Type Font	ttf
Open Type Font	otf
Embedded Open Type Font	eot
Scalable Vector Graphics	svg

Para trabalhar com *web fonts* é necessário converte-las nos formatos acima especificados, e para isso, existem vários conversores grátis na *web*.

O mais conhecido é o *Font Squirrel* que pode ser acessado no endereço <http://www.fontsquirrel.com/fontface/generator>.

Para converter as fontes, baixe uma versão *true type* da mesma e use o *Font Squirrel* para criar um pacote com as versões citadas acima.

@Font-Face

A diretiva CSS `@font-face` permite que sejam utilizadas fontes fora do padrão do sistema. Sua sintaxe obedece o padrão CSS, porém, com uma configuração diferenciada conforme mostrado abaixo:

```
@font-face {
    font-family: 'Nome da família da fonte criada';
    //Baixa as fontes do servidor web
    src: url('endereço do local da font/arquivo original da font') format('formato da font');
    //Procura a fonte no computador do usuário antes de baixá-la
    src: local('endereço do local da font/arquivo original da font');
}
```

Após as definições acima, podemos aplicá-las nos elementos HTML desejados.

Exemplo:

CSS

```
...
<style>
    @font-face {
        font-family: Imperador;
        src: url("fonts/imperator.eot");
        src: url("fonts/imperator.ttf") format("truetype");
    }
    p{font: bold 27px Imperador;}
</style>
...

```

HTML

```
...
<body>
    <h1>CSS 3 - @Font-Face</h1>
    <p>Texto de exemplo para mostrar @font-face</p>
</body>
...

```



Capítulo 13: Textos

* Visão Geral	131
* Suporte dos Browsers	131
* Atributo Word-Wrap	131
* Capitular	132
* Sombra	134

Visão Geral

Dentre os elementos de marcação de um documento HTML, os textos são os mais importantes em termos de *design* e estruturação de conteúdo.

Porém, muitas vezes são negligenciados no processo de construção de uma aplicação web, o que com certeza, ocasionará um resultado desastroso.

O dilema de grande parte dos *web designers* sempre foi produzir um texto objetivo, eficiente e que produzisse um estilo dentro da composição do *layout*. Alguns recursos usados com o nível 2 da CSS muitas vezes atendiam ao *design*, mas deixavam o conteúdo fora de uma estrutura semântica que permitisse aos mecanismos de busca uma adequada interpretação.

Um deles era poder usar fontes alternativas as do sistema, onde usava-se uma imagem para representar um título, ou bloco de texto, sem prejudicar o *layout* original. Em compensação, prejudicava-se os sistemas de busca que não podiam interpretar corretamente seu conteúdo.

A CSS 3 trouxe o recurso de *web fonts*, e outros, para compor a estrutura de *design* da aplicação sem essa perda semântica.

Suporte dos Browsers

Característica CSS 3					
	v5	v9	v10	v5.1	v15
Word-Wrap	✓	✓	✓	✓	✓
Capitular	✓	✓	✓	✓	✓
Sombra	✓	✗	✓	✓	✓

Atributo Word-Wrap

Quando um texto é digitado em um container HTML ele vai se acomodando ao mesmo, e pode produzir dois tipos de resultados em virtude dessa situação:

- a) O texto excede os limites do elemento no qual está contido, se o seu comprimento for maior que esse container.
- b) Se adequa a largura do elemento em que está contido usando os limites desse container, para também limitar o tamanho das linhas produzidas. Para isso, utiliza o recurso de quebra de linha automática chamado *word-wrap*. O CSS 3 incorporou esse atributo, criado originalmente pela Microsoft, em função do resultado prático que ele produz. Assim sendo, hoje ele é um atributo que pode ser configurado em qualquer elemento HTML que sirva de container de texto.

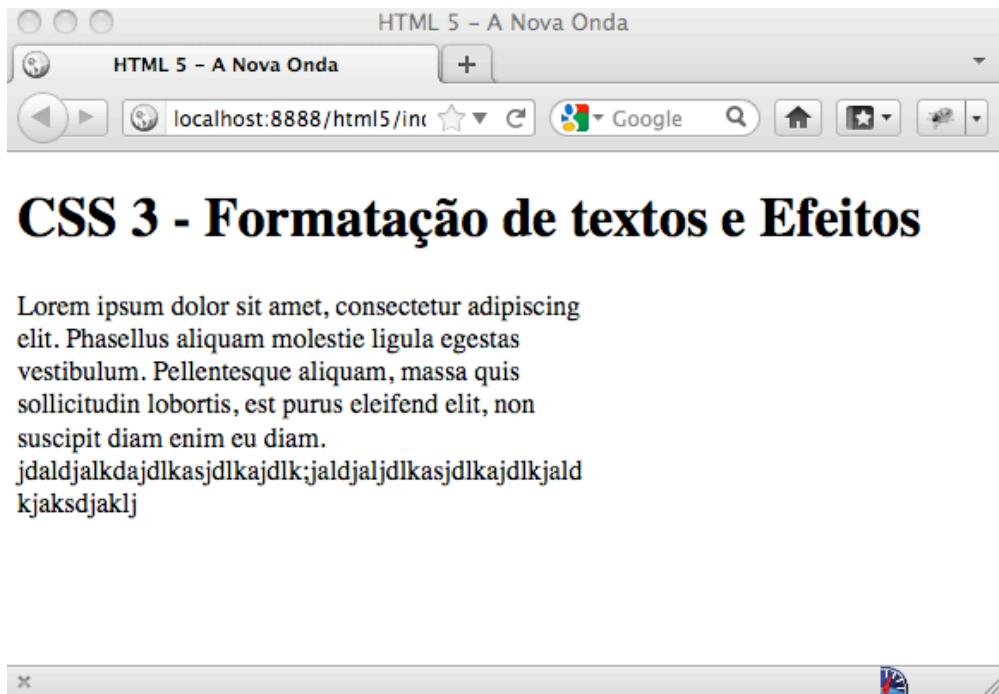
Exemplo:

HTML

```
...
<body>
  <h1>CSS 3 – Formatação de textos e Efeitos</h1>
  <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.
  Phasellus aliquam molestie ligula egestas vestibulum. Pellentesque
  aliquam, massa quis sollicitudin lobortis, est purus eleifend elit,
  non suscipit diam enim eu diam.
  jdaldjalkdajdlkasjdlkajdlk;jaldjaljdlkasjdlkajdlkjaldkjaksdjaklj</p>
</body>
...
***
```

CSS

```
...
<style>
  p{
    width: 330px;
    height:auto;
    word-wrap:break-word;
  }
</style>
...
```



O atributo *word-wrap* possui dois tipos de valores permitidos: *normal* e *break-word*.

O valor *normal* é o padrão e não determina a contenção do texto que extrapola os limites do *container* que o possui.

Já o valor *break-word* determina que o limite do texto digitado seja o mesmo limite do *container* no qual está incluso. Esse recurso funciona como a quebra de linha automática utilizada comumente em processadores de textos, como o *Word* por exemplo.

Se desejar ver o resultado diferente do que o apresentado na janela acima, basta comentar a linha CSS onde se encontra o atributo *word-wrap*.

Capitular

Um efeito muito usado em trabalhos impressos que envolvem grande fluxo de textos é a *capitular*. Esse efeito consiste em tornar a primeira letra do bloco de texto maior que o restante, destacando-a com margens, cor e até uma família de fonte diferente do texto original.

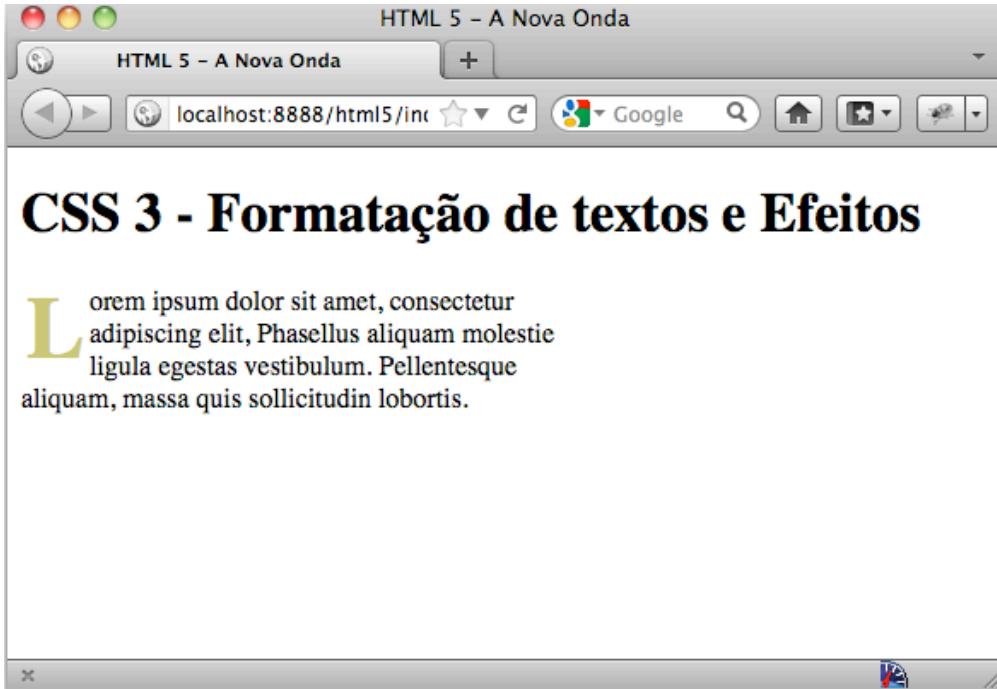
Esse recurso foi trazido para o ambiente digital pela CSS através do pseudo elemento *first-letter*.

Exemplo:**CSS**

```
...
<style>
p{
    width: 330px;
    height: auto;
    word-wrap: break-word;
}
p:first-letter{
    font: bold 51px 'Trebuchet';
    color: rgba(153,153,0,0.6);
    float: left;
    padding: 0 3px;
    margin-top: 7px;
}
</style>
...
```

HTML

```
...
<body>
<h1>CSS 3 – Formatação de textos e Efeitos</h1>
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit,
Phasellus aliquam molestie ligula egestas vestibulum. Pellentesque
aliquam, massa quis sollicitudin lobortis.</p>
</body>
...
```



Sombra

Um efeito muito utilizado é a sombra (*shadow*) que permite adicionar ao elemento uma característica de profundidade. No nível 2 da CSS para conseguir esse efeito tanto em objetos como textos, demandava várias configurações que na maioria das vezes não se compatibilizavam entre os navegadores, principalmente o Internet Explorer.

Com a chegada do nível 3 da CSS, aplicar sombra em boxes e textos tornou-se extremamente simples.

Nesse capítulo vamos aprender a aplicar o efeito sombra em textos puros com a CSS 3. Essa propriedade possui um conjunto de parâmetros de configuração conforme segue:

```
...
  h1{text-shadow: offsetX,offsetY,blur,color;}
```

✓ **offsetX**

Deslocamento no eixo horizontal x da sombra projetada.
Pode ser um número inteiro positivo ou negativo.

✓ **offsetY**

Deslocamento no eixo vertical y da sombra projetada.
Pode ser um número inteiro positivo ou negativo.

✓ **blur**

Nível do desfoque aplicado a sombra projetada.
Pode ser um número inteiro positivo.

✓ **color**

Cor aplicada a sombra projetada.
Pode ser usado o padrão *hexadecimal* ou *rgba*.

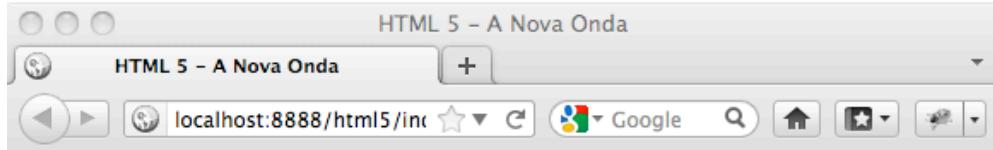
Exemplo:

CSS

```
...
<style>
  h1{text-shadow: 7px -5px 5px rgba(204,0,0,0.8);}
  p{
    width: 330px;
    height:auto;
    word-wrap:break-word;
    text-shadow:
  }
</style>
...
```

HTML

```
...
<body>
  <h1>CSS 3 - Formatação de textos e Efeitos</h1>
  <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit,
  Phasellus aliquam molestie ligula egestas vestibulum. Pellentesque
  aliquam, massa quis sollicitudin lobortis.</p>
</body>
...
```



CSS 3 - Formatação de textos e Efeitos

Placeholder text (Lorem ipsum):

```
 Lorem ipsum dolor sit amet, consectetur adipiscing
elit, Phasellus aliquam molestie ligula egestas
vestibulum. Pellentesque aliquam, massa quis
sollicitudin lobortis.
```



O recurso de sombra de texto permite aplicar várias configurações ao mesmo elemento, criando uma série de efeitos compostos ao texto ou bloco de texto.

Exemplo:

CSS

```
...
<style>
  h1{text-shadow: 7px -5px 5px rgba(204,0,0,0.8),
             0 30px 5px rgba(204,0,0,0.8);
  }
  p{
    width: 330px;
    height:auto;
    word-wrap:break-word;
    text-shadow:
  }
</style>
...
```

Aplique as modificações acima e veja o resultado novamente no navegador.

Capítulo 14: Bordas

* Visão Geral	137
* Suporte dos Browsers	137
* Bordas Arredondadas.....	137
* Bordas com Imagem.....	139

Visão Geral

O elemento borda sempre foi um desafio para os *web designers* quando se tratava de *layouts* com cantos arredondados.

Principalmente quando esses *layouts* tinham de ser orgânicos, ajustando-se ao padrão de monitor do usuário. Várias técnicas foram desenvolvidas para chegar ao efeito visual desejado com o elemento borda, mas nunca sem "sangue, suor e lágrimas", pois lembramos que existe o fator de compatibilização e um navegador chamado Internet Explorer.

Com a chegada do nível 3 da CSS um dos efeitos mais aguardados foi o de bordas arredondadas (*border-radius*), que veio dar mais facilidade no momento de criação de um *design* mais arredondado. Além dessa propriedade, a CSS 3 trouxe a possibilidade de criação de bordas personalizadas através de imagens (*border-image*).

Suporte dos Browsers

Característica CSS 3					
	v5	v9	v10	v5.1	v15
border-radius	✓	✓	✓	✓	✓
border-image	✓	✗	✓	✓	✓

Bordas Arredondadas

A propriedade *border-radius* permite arredondar as bordas de qualquer elemento HTML. Essa propriedade permite o arredondamento geral das extremidades, como também de cada uma das extremidades de acordo com seu eixo de posicionamento (top, left, right e bottom).

Por ser uma propriedade ainda não padronizada entre os fabricantes de navegadores, temos que configurá-la através do motor de renderização de cada um deles.

```
...
div{
    -webkit-border-radius: valor em pixels do arredondamento;
    -moz-border-radius: valor em pixels do arredondamento;
    -o-border-radius: valor em pixels do arredondamento;
    border-radius: valor em pixels do arredondamento;
}
...
```

Para configurar o padrão de arredondamento para cada um dos eixos do elemento HTML, temos que observar a escrita correspondente a cada motor de renderização, pois aqui também não há um consenso ainda.

```
...
div{
    -webkit-border-top-left-radius: 35px;
    -moz-border-radius-topleft: 35px;
    border-top-left-radius: 35px;
}
...
```

Exemplo:

CSS

```
...
<style>
  div {
    width: 150px;
    height: 150px;
    margin: 35px auto;
    background: #066;
    -webkit-border-top-left-radius: 35px;
    -webkit-border-bottom-right-radius: 35px;
    -moz-border-radius-topleft: 35px;
    -moz-border-radius-bottomright: 35px;
    border-top-left-radius: 35px;
    border-bottom-right-radius: 35px;
  }
</style>
...

```

HTML

```
...
<body>
  <h1>CSS 3 – Bordas</h1>
  <div></div>
</body>
...

```



No código CSS 3 acima não é necessário nenhuma marcação especial do motor de renderização para os navegadores Internet Explorer 9 e Opera, pois os mesmos entendem a marcação padrão especificada pelo W3C.

Bordas com Imagens

Outro efeito muito aguardado com o nível 3 da CSS é o de personalizar bordas com imagens (*border-image*). Com essa propriedade é possível criar uma imagem como padrão de preenchimento de uma borda, e aplicá-la no elemento HTML definido.

A propriedade *border-image* possui os seguintes parâmetros de configuração:

source

Define a *url* da imagem que será utilizada como padrão para a estilização da borda do elemento HTML.

slice

Esse parâmetro define o comprimento que configura a distância de cada um dos limites da imagem (*top*, *right*, *bottom* e *left*), marcando a área que poderá ser usada como moldura do elemento HTML.

repeat

Esse parâmetro define o processo de repetição da imagem ao longo da borda. Pode ser configurado com os seguintes valores:

- ✓ *Repeat*

Repete a imagem em pequenos pedaços (ladrilhos) ao longo dos eixos de posicionamento do Modelo de Caixa, usando duas estruturas chaves: *Topo* => *Bottom* e *Right* => *Left*.

- ✓ *Stretch*

Esse parâmetro estica a imagem para preencher o comprimento da borda do elemento HTML.

- ✓ *Stretch*

Esse parâmetro estica a imagem para preencher o comprimento da borda do elemento

- ✓ *Round*

Preenche o comprimento da borda com pequenos pedaços da imagem, e com a quantidade dos pedaços da imagem em valores inteiros. Se for necessário, o elemento será redimensionado para acomodá-lo ao espaço de preenchimento se necessário.

As unidades de valores definidas no parâmetro *repeat* para os eixos de posicionamento, não precisam do complemento *px* de *pixels*, pois servem para definir elementos *bitmap*, *jpg*, *png*, bem como *svg* (*Scalable Vector Graphics*).

Exemplo:

CSS

```
...
<style>
p {
    width: 330px;
    height: 85px;
    padding: 10px;
    word-wrap: break-word;
    border-width: 11px;
    -moz-border-image: url(border.png) 31 stretch repeat;
    -webkit-border-image: url(border.png) 31 stretch repeat;
    -o-border-image: url(border.png) 31 stretch repeat;
    border-image: url(border.png) 31 stretch repeat;
}
</style>
...
```

HTML

```
...
<body>
  <h1>CSS 3 - Bordas</h1>
  <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit,
  Phasellus aliquam molestie ligula egestas vestibulum. Pellentesque
  aliquam, massa quis sollicitudin lobortis.</p>
</body>
...
```



A propriedade *border-width* aplicada no exemplo acima é aceita somente no navegador Firefox.

O Firefox possui também uma propriedade que permite adicionar multiplas cores a borda de um elemento HTML.

A sintaxe para essa propriedade é:

```
...
seletor{-moz-border-*-colors: cor(es);}
...
```

O asterisco representa o eixo de posicionamento em que será aplicado a cor na borda.

Exemplo:

CSS

```
...
<style>
  p {
    -moz-left-width: 5px;
    -moz-border-left-colors: blue yellow red;
  }
</style>
...
```

Capítulo 15: Fundos

* Visão Geral.....	142
* Suporte dos Browsers.....	142
* Multiplos Fundos.....	115
* Tamanho de Fundos	144

Visão Geral

Um dos elementos decorativos mais utilizados no desenvolvimentos de aplicações web são os fundos (*backgrounds*).

Um layout criativo possui 50% de sua eficiência em cima de um fundo bem elaborado, que contraste com os elementos importantes do conteúdo, sem ser chamativo ou mesmo um obstáculo a leitura do mesmo.

Manipular fundos no nível 2 da CSS não era um bicho de sete cabeças, mas também não nos permitia uma ampla configuração e liberdade para sua criação. Eram necessários recursos e marcações extras, muitas vezes para realizar efeitos considerados simples.

Com a CSS nível 3, abre-se um leque de possibilidades com a capacidade de manipulação de múltiplos fundos, redimensionamento de imagens de fundo instantâneamente entre outros recursos.

Suporte dos Browsers

Característica CSS 3					
	v5	v9	v10	v5.1	v15
Multiplos Backgrounds	✓	✓	✓	✓	✓
Background Size	✓	✓	✓	✓	✓
Background Clip	✓	✓	✓	✓	✓
Background Origin	✓	✓	✓	✓	✓
Mask	✗	✗	✗	✓	✓

Múltiplos Fundos

O nível 3 da CSS trouxe a possibilidade de manipulação de mais de um fundo aplicado a um elemento HTML. Não houve a criação de nenhuma propriedade especial, sendo usada a propriedade já conhecida *background*, acrescida das funcionalidades CSS 3. A nova sintaxe da propriedade *background* no nível 3 da CSS é:

```
...
  seletor {
    background: url(imagem) [repeat] [posX] [posY],
                url(imagem) [repeat] [posX] [posY];
  }
...

```

url

Define a imagem que será utilizada como fundo do elemento HTML especificado.

repeat

Esse parâmetro define como a imagem de fundo preencherá o espaço do elemento HTML especificado. Possui os valores: *no-repeat*, *repeat*, *repeat x*, *repeat y*.

- ✓ *no-repeat*

Esse valor não repete a imagem definida como fundo do elemento HTML.

- ✓ *repeat*

Esse valor repete a imagem definida como fundo do elemento HTML, e é o valor padrão do parâmetro *repeat*.

- ✓ *repeat x*

Esse valor repete a imagem definida como fundo do elemento HTML através do eixo horizontal (*x*) do plano cartesiano.

✓ *repeat y*

Esse valor repete a imagem definida como fundo do elemento HTML através do eixo vertical (*y*) do plano cartesiano.

Para acrescentar mais de um fundo ao elemento HTML deve-se separar as configurações dos mesmos por vírgula. Não há um limite para aplicação de múltiplos fundos na especificação técnica da CSS 3.

Exemplo:

CSS

```
...
<style>
  div {
    width: 1000px;
    height: 601px;
    margin: 35px auto;
    background: url(elefante.png) no-repeat center 280px,
                url(lago.jpg) no-repeat left top;
  }
</style>
...
```

HTML

```
...
<body>
  <h2>Multiplos backgrounds com CSS3</h2>
  <div id="box"></div>
</body>
...
```



Tamanho de Fundos

Junto com a possibilidade de manipular múltiplos fundos a CSS 3 trouxe a propriedade *background-size*, que permite definir uma escala de tamanho para a imagem de fundo.

Essa propriedade pode ser manipulada com *jQuery* para aplicar configurações *on-line* as imagens de fundo, possibilitando o usuário interagir facilmente com os elementos de fundo do *layout*.

A sintaxe dessa propriedade é:

```
...
  seletor {
    background-size: width(px/%) height(px/%); }
...

```

Utilizando o exemplo do tópico anterior, será acrescentado um tamanho as duas imagens usadas como fundos (elefante e lago).

Exemplo:

CSS

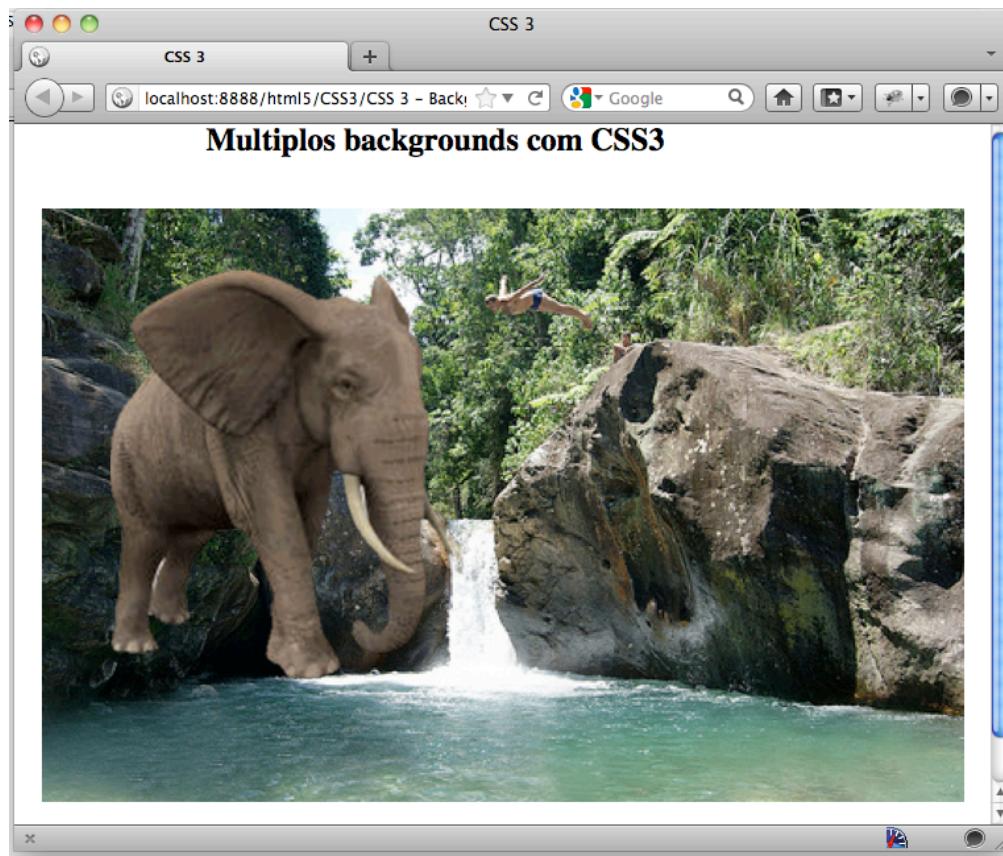
```
...
<style>
  div {
    width: 1000px;
    height: 601px;
    margin: 35px auto;
    background: url(elefante.png) no-repeat center 280px,
                url(lago.jpg) no-repeat left top;
  }
div {background-size: auto,130%;}
</style>
...
```

ou

```
...
<style>
  div {
    width: 1000px;
    height: 601px;
    margin: 35px auto;
    background: url(elefante.png) no-repeat center 280px,
                url(lago.jpg) no-repeat left top;
    background-size: auto,130%;
  }
</style>
...
```

A aplicação da propriedade *background-size* acima faz com que a largura da primeira imagem (elefante) fique dentro das medidas originais, e a segunda imagem (lago) tenha um redimensionamento escalar (largura x altura) de 30% do original.

Como foi especificado apenas um valor para cada imagem, eles são tomados como referência para largura e altura.



Perceba que com o redimensionamento da imagem do lago para +30% do original, o elefante parece estar flutuando sobre o lago. Para fazer o acerto de posição, basta redefinir o eixo de vertical *y* da imagem.

Exemplo:

CSS

```
...
<style>
div {
    width: 1000px;
    height: 601px;
    margin: 35px auto;
    background: url(elefante.png) no-repeat center 135px,
                url(lago.jpg) no-repeat left top;
    background-size: auto,130%;
}
</style>
...
```

Recarregue a janela do exercício anterior com a nova configuração do posicionamento *y* da imagem de fundo, para ver a figura do elefante na posição correta.

Capítulo 16: Sombras

* Visão Geral	147
* Suporte dos Browsers	147
* Sombra em Boxes	147

Visão Geral

Como dito no capítulo 13 sobre textos, o efeito sombra foi uma das facilidades aguardadas pelos *designers* pelo nível 3 da CSS 3.

Projetar sombras sobre objetos além de produzir um destaque, também causa a sensação de profundidade com a ilusão de perspectiva.

Para aplicar uma sombra sobre um objeto HTML a CSS 3 trouxe a propriedade *box-shadow*, que permite manipular esse efeito de maneira simples e sem marcação extra.

Suporte dos Browsers

Característica CSS 3					
v5	✓	✓	✓	✓	✓
Box Shadow					

Sombra em Boxes

A propriedade CSS 3 para aplicar um efeito de sombra em objetos HTML é a *box-shadow*. Abaixo segue a sintaxe de marcação para essa propriedade:

```
...
  seletor {
    box-shadow: [ posX ] [ posY ] [ blur ] [ color ];
  }
...

```

posX

Define a posição horizontal do efeito de sombra no eixo x do plano cartesiano. O valor desse parâmetro pode ser um número inteiro positivo ou negativo.

Os valores positivos deslocam a sombra da esquerda para a direita, e os valores negativos deslocam a sombra da direita para a esquerda.

posY

Define a posição vertical do efeito de sombra no eixo y do plano cartesiano. O valor desse parâmetro pode ser um número inteiro positivo ou negativo.

Os valores positivos deslocam a sombra do topo para a parte inferior, e os valores negativos deslocam a sombra da parte inferior para o topo.

blur

Esse parâmetro define a intensidade do desfoque aplicado na sombra, e aceita valores numéricos inteiros. Valores em ordem crescente diminuem a intensidade do desfoque, e valores em ordem decrescente aumentam a intensidade do desfoque.

color

Define a cor aplicada na sombra e pode ser configurada com os valores *rgba* ou *hexadecimal*.

Exemplo:

CSS

```
...
<style>
  img {
    width: 250px;
    height: 250px;
    -webkit-box-shadow: 10px 10px 13px rgba(51, 102, 153, 0.5);
    -moz-box-shadow: 10px 10px 13px rgba(51, 102, 153, 0.5);
    box-shadow: 10px 10px 13px rgba(51, 102, 153, 0.5);
  }
  div{
    margin: 0 auto;
    width: 400px;
  }
</style>
...

```

HTML

```
...
<body>
  <div>
    <h1>CSS 3 - Sombras</h1>
    
  </div>
</body>
...

```



Capítulo 17: Transformações

* Visão Geral	150
* Suporte dos Browsers	150
* Transformações	150
Translate	150
Rotate	153
Skew	155
Scale	157

Visão Geral

No mundo dinâmico da *web* estar limitado a elementos estáticos e sem vida é um passo para o fracasso. Pode manipular elementos HTML das mais variadas formas sempre foi o interesse maior de qualquer web designer, e para isso vários recursos programáticos foram usados ao longo desses anos como *javascript*, *flash* e mais recentemente *jQuery*. Porém, muitas vezes o efeito desejado é extremamente simples (uma rotação por exemplo), e o custo/benefício da inserção de um código com *kbytes* a mais no documento HTML não compensa.

Com a chegada do nível 3 da CSS essa frustração tem os seus dias contados, pois, mesmo estando em projeto de finalização pelo grupo de trabalho do W3C, ela trouxe uma série de características que manipulam as transformações mais elementares de forma simples e objetiva. O melhor de tudo é o processo ser nativo da própria CSS, sem necessidade de linguagem de programação.

Suporte dos Browsers

Característica CSS 3					
	v5	v9	v10	v5.1	v15
transform	✓	✓	✓	✓	✓

As versões dos navegadores acima aceitam a propriedade *transform* com a adição do prefixo de seus fabricantes.

Transformações

O nível 3 da CSS traz um conjunto de funções para manipulação de transformações lineares aplicadas a elementos HTML. Essas transformações podem ser feitas em dois contextos distintos: 2D e 3D.

Como os navegadores atuais não estão preparados para renderização nativa do contexto 3D, vamos abordar nessa obra apenas o contexto 2D. Para aplicar transformações em elementos HTML, a CSS 3 trouxe a propriedade *transform* manipula as funções: *translate*, *rotate*, *skew* e *scale*.

A sintaxe para manipulação da propriedade *transform* é:

```
...
seletor {
    -webkit-transform: function(parameters);
    -moz-transform: function(parameters);
    -o-transform: function(parameters);
}
...
...
```

Translate

A função *translate* desloca o posicionamento do elemento HTML no eixo x (horizontal) e no eixo y (vertical), conforme os parâmetros especificados.

A sintaxe da função é:

```
...
seletor {-webkit-transform: translate(posX,posY); }
...
...
```

Exemplo:

CSS

```
...
<style>
  img {
    width: 250px;
    height: 250px;
    -webkit-transform: translate(35px, 25px);
    -moz-transform: translate(35px, 25px);
    -o-transform: translate(35px, 25px);
  }
  #square {
    height: 250px;
    width: 250px;
    background: #CCC;
    position: absolute;
    z-index: -1;
  }
</style>
...

```

HTML

```
...
<body>
  <div>
    <h1>CSS 3 - Transformações</h1>
    <div id="square"></div>
    
  </div>
</body>
...

```



Os valores x e y aplicados a função *translate* devem ser números inteiros, e podem ser positivos ou negativos. Os números negativos no eixo horizontal deslocam o elemento da direita para esquerda, e no eixo vertical da parte inferior para o topo em relação ao seu *container* pai.

Exemplo:

CSS

```
...
<style>
  img {
    width: 250px;
    height: 250px;
    -webkit-transform: translate(35px, -35px);
    -moz-transform: translate(35px, -35px);
    -o-transform: translate(35px, -35px);
  }
  #square {
    height: 250px;
    width: 250px;
    background: #CCC;
    position: absolute;
    z-index: -1;
  }
</style>
...
```



No exemplo acima, foi utilizado uma imagem de fundo como ponto de referência ao deslocamento da imagem do logo.

Rotate

Essa função, como o próprio nome sugere, rotaciona o elemento HTML em seu próprio eixo central.

O valor do parâmetro da função é definido em graus (*degree*), e deve ser um número inteiro positivo ou negativo.

Com valores inteiros e positivos o elemento é rotacionado no sentido horário, já com valores negativos e inteiros o objeto é rotacionado no sentido anti-horário.

A sintaxe da função é:

```
...
  seletor {-prefixo-transform: rotate(valor deg); }
...

```

Exemplo:

CSS

```
...
<style>
  img {
    width: 250px;
    height: 250px;
    -webkit-transform: rotate(180deg);
    -moz-transform: rotate(180deg);
    -o-transform: rotate(180deg);
  }
  #square {
    height: 250px;
    width: 250px;
    background: #CCC;
    position: absolute;
    z-index: -1;
  }
</style>
...
```

HTML

```
...
<body>
  <div>
    <h1>CSS 3 – Transformações</h1>
    <div id="square"></div>
    
  </div>
</body>
...
```

No código acima o elemento imagem é rotacionado no sentido horário.

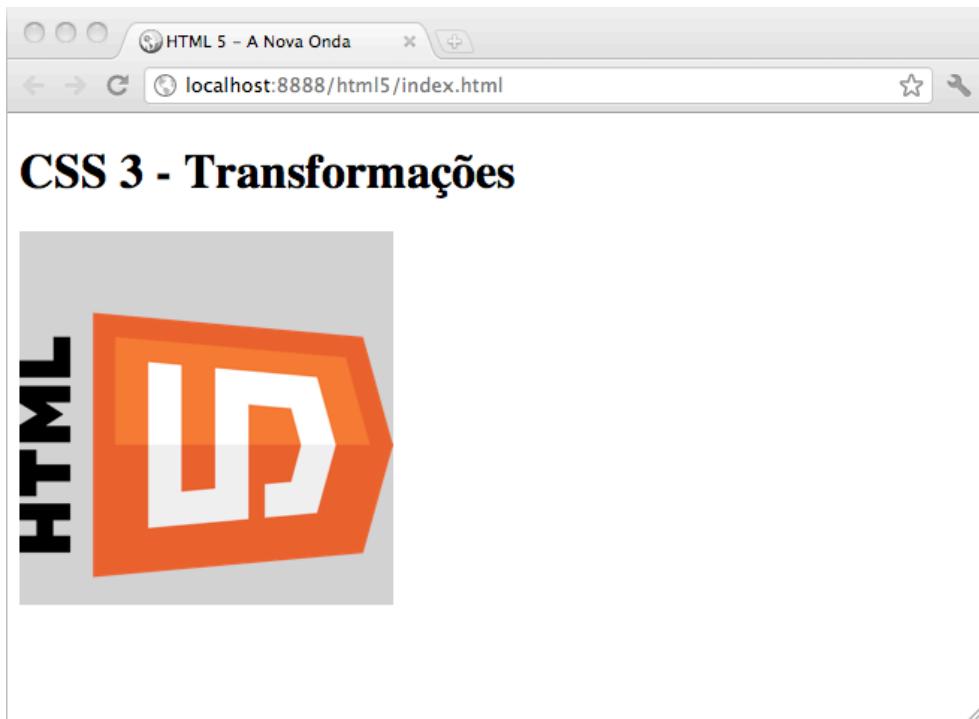


No código abaixo, o elemento é rotacionado no sentido anti-horário com valores negativos.

Exemplo:

CSS

```
...
<style>
  img {
    width: 250px;
    height: 250px;
    -webkit-transform: rotate(-90deg);
    -moz-transform: rotate(-90deg);
    -o-transform: rotate(-90deg);
  }
  #square {
    height: 250px;
    width: 250px;
    background: #CCC;
    position: absolute;
    z-index: -1;
  }
</style>
...
```



Skew

A função *skew* permite alterar o eixo horizontal e vertical de um elemento HTML definindo um ângulo de inclinação. O valor do ângulo de inclinação pode ser positivo ou negativo.

A sintaxe da função *skew* pode ser escrita de duas formas:

```
...
<style>
  seletor {prefixo-transform:skew(valorX deg, valorY deg);}
</style>
...
ou
...
<style>
  seletor {
    prefixo-transform:skewX(valor deg);
    prefixo-transform:skewY(valor deg);
  }
</style>
...
```

Por questões de economia de digitação é adotada a primeira sintaxe como referência. Quando é usada a primeira opção se um dos parâmetros é omitido, a função adota como padrão o valor zero para o eixo *y*.

Exemplo:

CSS

```
...
<style>
    img {
        width: 250px;
        height: 250px;
        -webkit-transform: skew(-25deg, 0deg);
        -moz-transform: skew(-25deg, 0deg);
        -o-transform: skew(-25deg, 0deg);
    }
</style>
...
```

HTML

```
...
<body>
    <div>
        <h1>CSS 3 - Transformações</h1>
        
    </div>
</body>
...
```



Valores negativos definem uma inclinação no sentido horário, valores positivos definem uma inclinação no sentido anti-horário.

Scale

A função *scale* permite aumentar ou diminuir o tamanho original de um elemento HTML, redefinindo as medidas proporcionalmente entre a largura e altura, chamada de medidas escalares.

Os valores aplicados a função *scale* não possuem unidades, e são representadas por números positivos ou negativos. O valor padrão é 1, e portanto, aplicar o valor 3 a um elemento é atribuir uma proporção de 300% em relação ao tamanho original do elemento.

A sintaxe da função *scale* pode ser escrita de duas formas:

```
...
<style>
  seletor {prefixo-transform:scale(valorX, valorY);}
</style>
...
ou
...
<style>
  seletor {
    prefixo-transform:scaleX(valor);
    prefixo-transform:scaleY(valor);
  }
</style>
...
```

Ao contrário da função anterior, quando um dos valores for omitido a função *scale* toma o valor especificado como padrão para as duas posições (*x* e *y*).

O valor padrão de unidade para a função *scale* é 1, e cada unidade inteira acrescida ao valor padrão representa 100% do valor original. Unidades fracionadas representam proporções em relação ao valor original.

```
...
<style>
  seletor {
    prefixo-transform:scale(2);
  }
</style>
...
```

O código acima aplica um valor que representa duas vezes a medida original de largura e altura de um elemento HTML.

Já o código a seguir diminui em 50% o valor original de largura e altura do elemento HTML.

```
...
<style>
  seletor {
    prefixo-transform:scale(0.5);
  }
</style>
...
```

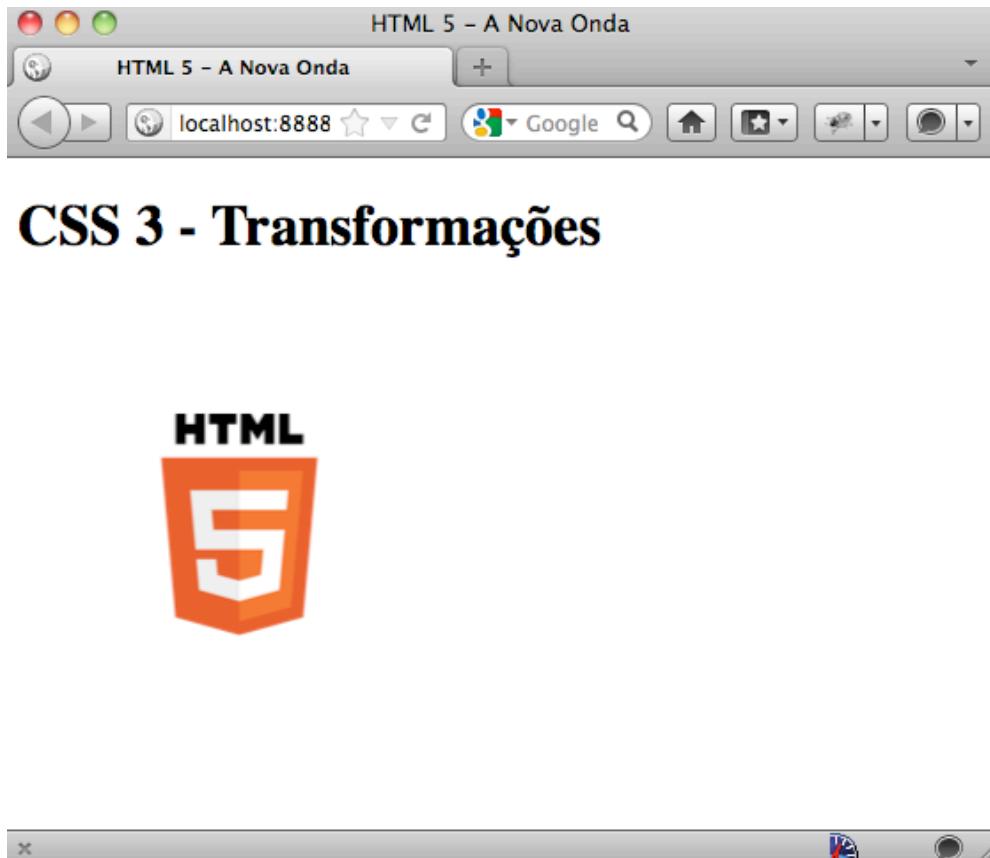
Exemplo:

CSS

```
...
<style>
    img {
        width: 250px;
        height: 250px;
        -webkit-transform: scale(0.5);
        -moz-transform: scale(0.5);
        -o-transform: scale(0.5);
    }
</style>
...
```

HTML

```
...
<body>
    <div>
        <h1>CSS 3 - Transformações</h1>
        
    </div>
</body>
...
```



Capítulo 17: Transições

* Visão Geral	160
* Suporte dos Browsers	160
* Transições	160
Propriedades	160
Duração	160
Suavizadores	161
Temporizador	161

Visão Geral

No ambiente *web* uma das formas de prender a atenção do usuário é aplicar uma mudança no estado original dos elementos HTML. Mudar a altura ou largura de uma *div* em determinado local da interface, pode chamar a atenção para as informações contidas na mesma.

Porém, no nível 2 da CSS quando ocorre uma mudança de estado em qualquer elemento HTML, não há um efeito visual suave, agradável ao olhos do público-alvo. Esse efeito de transição é abrupto, o que muitas vezes faz com que *web designers* recorram a ferramentas externas para uma composição mais harmônica, como por exemplo o *Flash*.

O nível 3 da CSS traz recursos de transições para ser aplicados nas mudanças de estados de elementos HTML, com mais suavidade e eficiência, sem a necessidade de códigos de programação ou ferramentas externas. Simplesmente com marcação CSS.

Com essa característica, a CSS 3 torna-se uma ferramenta de estilização e animação compatível com dispositivos móveis como celulares, iPhone e tablets.

Suporte dos Browsers

Característica CSS 3					
	v5	v9	v10	v5.1	v15
transition	✓	✗	✓	✓	✓

As versões dos navegadores acima aceitam a propriedade *transition* com a adição do prefixo de seus fabricantes.

Transições

Para poder aplicar transições de estados em elementos HTML com suavidade, a CSS 3 trouxe a função *transition*.

Ela possui um conjunto de propriedades que permitem criar efeitos com controle de duração, tempo de execução, suavização e com as principais propriedades CSS dos elementos HTML.

Suas propriedades são:

Propriedades

Para definir qual característica deve ser animada com a função *transition* podemos usar a propriedade *property*. Com ela podemos criar efeitos de transição para uma, ou mais características de um elemento HTML.

Como valor são aceitos os seguintes parâmetros: *all*, *none* ou uma propriedade CSS específica.

```
...
<style>
  seletor{-prefixo-transition-property: height;}
</style>
...
```

Duração

A propriedade *duration* define o período de tempo que a transição leva para ser completada. O valor especificado pode ser uma unidade de tempo em segundos ou milisegundos, sendo o valor padrão zero.

...

```

<style>
  seletor{-prefixo-transition-duration: 2s; }
</style>
...

```

Suavizadores

É possível controlar a suavidade do efeito com que a transição é realizada entre os estados do elemento HTML, com a propriedade *timing-function*.

Ela aceita como valores: *ease*, *ease-in*, *ease-out*, *ease-in-out*. O valor padrão é *ease*.

```

...
<style>
  seletor{-prefixo-transition-timing-function: ease; }
</style>
...

```

Temporizador

Uma transição pode ter seu início controlado por uma propriedade que funciona como um temporizador, chamada *delay*. Sua função básica é retardar o inicio de execução da transição, sendo definido como valor uma unidade em milisegundos ou segundos.

```

...
<style>
  seletor{-prefixo-transition-delay: 250ms; }
</style>
...

```

Para demonstrar a aplicação das propriedades da função *transition* foi criado o código abaixo:

Exemplo:

CSS

```

...
<style>
  img {
    width: 250px;
    height:250px;
    /*Prefixo para navegador Chrome e Safari*/
    -webkit-transition-property:-webkit-transform;
    -webkit-transition-duration:2s;
    -webkit-transition-timing-function:ease-out;
    -webkit-transition-delay:1s;
    /*Prefixo para navegador Mozilla Firefox*/
    -moz-transition-property:-webkit-transform;
    -moz-transition-duration:2s;
    -moz-transition-timing-function:ease-out;
    -moz-transition-delay:1s;
    /*Prefixo para navegador Opera*/
    -o-transition-property:-webkit-transform;
    -o-transition-duration:2s;
    -o-transition-timing-function:ease-out;
    -o-transition-delay:1s;
    /*Prefixo para navegador IE 9*/
    -ms-transition-property:-webkit-transform;
    -ms-transition-duration:2s;
    -ms-transition-timing-function:ease-out;
    -ms-transition-delay:1s;
    cursor:pointer;
  }
  img:hover {

```

```

width: 250px;
height: 250px;
-webkit-transform: rotate(45deg);
-moz-transform: rotate(45deg);
-o-transform: rotate(45deg);
-ms-transform: rotate(45deg);
}
</style>
...

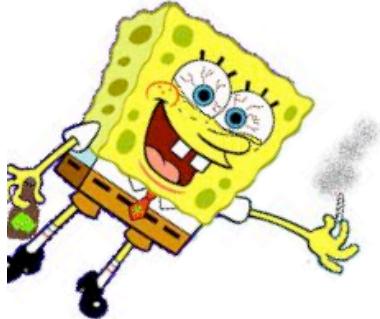
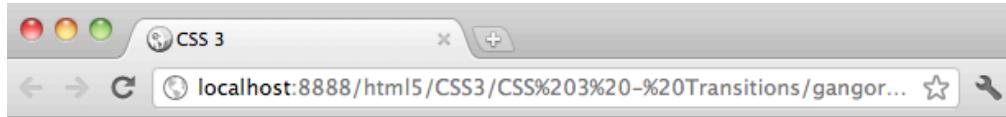
```

HTML

```

...
<body>
  <div>
    <h1>CSS 3 – Transições</h1>
    
  </div>
</body>
...

```



A função *transition* pode ser marcada com uma sintaxe curta chamada atalho, conforme abaixo:

```

...
<style>
  img {
    width: 250px;
    height: 250px;
    -webkit-transition: -webkit-transform 2s ease-out 1s;
    -moz-transition: -moz-transform 2s ease-out 1s;
    -o-transition: -o-transform 2s ease-out 1s;
    -ms-transition: -ms-transform 2s ease-out 1s;
    cursor: pointer;
  }
</style>
...

```

Capítulo 19: Media Queries

* Visão Geral	166
* Suporte dos Browsers	166
* Media Types	166
* Media Queries.....	167

Visão Geral

Desde que foi criado o HTML tem como foco principal ser uma marcação portável, e acessível, em todos os dispositivos que possibilitam acesso a *internet*.

Porém, como o formato destes dispositivos varia de maneira significativa, o profissional *web* tem de planejar seu *layout* para que se adapte a cada um deles.

Esse desafio foi amenizado com a CSS 2 e o atributo *media types*, que definia o tipo de media usada e carregava o arquivo CSS correspondente. Por exemplo: Se em uma aplicação fosse necessário imprimir um conjunto de informações, criava-se um folha de estilos específica para impressão e definia-se o *media types* como *print*.

Esse processo funcionou bem até a chegada dos novos dispositivos como iPhone, tablets, netbooks, tvs de LCD entre outros.

Não é mais suficiente declarar o tipo de *media*, mas adaptar a estrutura do *layout* para que responda de forma dinâmica a mudança do dispositivo de acesso a *internet*. Esse processo é chamado de *Design Responsivo* e para acompanhá-lo a CSS 3 trouxe as *media queries*.

Neste capítulo será abordado a estrutura de funcionamento das *media queries*, e como usá-las para criar um *layout* mais responsivo

Suporte dos Browsers

Característica CSS 3					
	v5	v9	v10	v5.1	v15
@media queries	✓	✓	✓	✓	✓

Media Types

Desde o seu surgimento a CSS foi criada com o objetivo de formatar as informações da aplicação, para que possam ser acessadas dos principais dispositivos conectados na *web*.

Para facilitar esse processo o W3C criou um atributo chamado *media*, onde é definido como valor o tipo de dispositivo que vai acessar a CSS criada. Por exemplo, se houver a necessidade de imprimir as informações da aplicação *web* pode-se criar um CSS para o dispositivo impressora, e outra para a tela do computador (*screen*).

É possível usar a sintaxe do atributo *media* de três maneiras:

1. Criando um *link* externo com o arquivo CSS:

```
...
<link rel='stylesheet' href='arquivo.css' media='tipo de dispositivo'>
...
```

2. Usando a diretiva *@import*:

```
...
@import url("arquivo.css") tipo de dispositivo (print, screen e etc.);
```

3. Usando a diretiva *@media*:

```
...
@media tipo de dispositivo{
    ...código CSS...
}
```

Existe um tipo de *media type* para cada dispositivo, conforme abaixo:

screen

Define como tipo de *media* a tela do monitor de um dispositivo.

print

Define o dispositivo impressora como tipo de *media*.

handheld

Define uma folha de estilo para PDA's, celulares, e dispositivos que possuam tela de monitor de pequenas proporções.

projection

Define como tipo de *media* um dispositivo do tipo projetor, *data show, power point*.

aural

Define como tipo de *media* leitores de tela.

tv

Define como tipo de *media* aparelhos de tv's.

tty

Define uma CSS para dispositivos como terminais, totens, teletypes e dispositivos com tela limitada.

Media Queries

Com o avanço da mobilidade e variedade dos dispositivos que acessam a internet, o processo de definição das CSS para tipos de *media* tornou-se insuficiente diante da dinâmica de adaptação do visual da aplicação ao dispositivo do usuário.

Com a chegada de tablets, iPhone, smartphones, blackberry e etc., uma aplicação *web* tem que estar pronta para adaptar-se ao tipo de *media* do usuário não somente no aspecto de largura de tela, mas na estrutura geral dos elementos que compõem o *layout* da aplicação *web*.

Com a chegada da CSS 3 vieram as *media queries*, que são as *media types* com a possibilidade de usar expressões para configurar as características do dispositivo do usuário.

As *media queries* podem ser configuradas no documento *web* de três formas diferentes:

1. Criando um *link* externo com o arquivo CSS:

```
...
<link rel='stylesheet' href='arquivo.css' media='tipo de dispositivo e
(expressão lógica)'>
...

```

2. Usando a diretiva *@import*:

```
...
@import url("arquivo.css") device and (expression);
...
```

3. Usando a diretiva `@media`:

```
...
@media device and (expression) {rules}
...
```

Características de Media

São informações sobre o tipo de dispositivo usado pelo usuário para visualizar a aplicação *web*, como resolução, dimensões, cores e etc. Essas informações são usadas para avaliar uma expressão lógica, e definir quais as regras de estilização serão aplicadas.

Um exemplo de expressão lógica pode ser "aplicar estilo x a dispositivos que possuam tela com largura de 480px" ou "apenas em dispositivos com orientação de visualização horizontal".

A sintaxe padrão para definição das características de *media* é:

```
...
@media device and (expression) {rules}
...
```

Tipos de Características de Media

Existe um conjunto de características que podem ser utilizadas para definição do dispositivo, e forma de visualização da página da aplicação *web*.

width

Define a largura da janela de exibição de um dispositivo de *media*. Geralmente, isso significa a largura atual da janela do *browser* incluindo as barras de rolagem.

Exemplo:

CSS

```
...
<style>
  @media device and (width:600px) { rules }
</style>
...
```

A regra acima define que serão aplicadas as regras de CSS para dispositivos cuja tela de visualização tenham a largura de 600px.

max-width

Configura um valor máximo de largura para a janela de exibição (*browser*), e ajusta os elementos de *layout* a esse valor caso o usuário aumente a dimensão horizontal da mesma.

Exemplo:

CSS

```
...
<style>
  @media device and (max-width:720px) { rules }
</style>
...
```

min-width

Configura um valor mínimo de largura para a janela de exibição (*browser*), e ajusta os elementos de *layout* a esse valor mínimo caso o usuário diminua a dimensão horizontal da mesma.

Exemplo:

CSS

```
...
<style>
  @media device and (min-width:480px) { rules }
</style>
...
```

height

Define a altura da janela de exibição de um dispositivo de *media*. Geralmente, isso significa a altura atual da janela do *browser*.

Exemplo:

CSS

```
...
<style>
  @media device and (height:720px) { rules }
</style>
...
```

max-height

Configura um valor máximo de altura para a janela de exibição (*browser*), e ajusta os elementos de *layout* a esse valor caso o usuário use uma visualização no seu dispositivo com essa configuração de altura.

Exemplo:

CSS

```
...
<style>
  @media device and (max-height:860px) { rules }
</style>
...
```

min-height

Configura um valor mínimo de altura para a janela de exibição (*browser*), e ajusta os elementos de *layout* a esse valor caso o usuário use uma visualização no seu dispositivo com essa configuração de altura.

Exemplo:

CSS

```
...
<style>
  @media device and (min-height:560px) { rules }
</style>
...
```

device-width

Essa característica define a largura da tela do dispositivo que está renderizando a página *web*, ao invés da janela do *browser*.

Exemplo:

CSS

```
...
<style>
  @media device and (device-width: 960px) { rules }
</style>
...
```

device-height

Essa característica define a altura da tela do dispositivo que está renderizando a página *web*, ao invés da janela do *browser*.

Exemplo:

CSS

```
...
<style>
  @media device and (device-height: 420px) { rules }
</style>
...
```

A característica *device* (*width* ou *height*) é útil para desenvolvimento de aplicações *web* direcionadas para dispositivos móveis como celulares, que possuem pequenas telas de visualização.

orientation

Essa característica permite a definição do modo de visualização de uma página *web*, baseada na posição do dispositivo que está sendo usado para sua renderização.

Existem duas formas de visualização referentes a posição do dispositivo: Horizontal e Vertical. A posição horizontal é conhecida como *landscape* (paisagem), e a posição vertical como *portrait* (retrato). Por padrão, a maioria dos dispositivos vem com a posição de visualização vertical. Sua sintaxe é:

Exemplo:

CSS

```
...
<style>
  @media device and (orientation: landscape or portrait) { rules }
</style>
...
```

aspect ratio

Quando desenvolvemos aplicações *web* para uma variada quantidade de modelos e tamanhos de telas, uma característica importante é a relação de aspecto que existe entre a tela de visualização e o conteúdo mostrado. Essa característica é definida com *aspect ratio*.

O *aspect ratio* trabalha com valores inteiros e positivos para as posições horizontal e vertical, representando o aspecto proporcional entre largura e altura do dispositivo usado.

Para representarmos uma tela de dispositivo com *aspect ratio* quadrada podemos usar a proporção de 1/1, porém, se desejarmos uma proporção de tela tipo panorâmica (*wide screen*) podemos definir um *aspect ratio* de 16/9 em geral.

É possível definir o *aspect ratio* tanto para a janela do *browser* como para a tela do dispositivo.

Exemplo:

CSS

```
...
<style>
  @media device and (aspect-ratio:horizontal/vertical) {rules}
</style>
...
```

ou

```
...
<style>
  @media device and (device-aspect-ratio:horizontal/vertical) {rules}
</style>
...
```

Encadeamento Características de Media

É possível definir um encadeamento de múltiplas queries para um mesmo tipo de dispositivo, adicionando as expressões com o operador lógico *and* (e).

A sintaxe padrão é:

```
...
<style>
  @media device and (expression) and (expression) {rules}
</style>
...
```

Exemplo:

CSS

```
...
<style>
  @media only screen and (aspect-ratio:16/9) and (orientation: landscape) {rules}
</style>
...
```

É possível também definir características para diferentes dispositivos de *media* de modo encadeado, separando cada um dos dispositivos com vírgula conforme mostrado abaixo:

Exemplo:

CSS

```
...
<style>
  @media only screen and (aspect-ratio:16/9) and (orientation: landscape), projection and (orientation:landscape){rules}
</style>
...
```

Capítulo 20: Template Layout

* Visão Geral	173
* Suporte dos Browsers	173
* Módulo Template Layout	173
* Pseudo-elemento :slot	173
* Propriedade display	174
* Propriedade position	176

Visão Geral

O W3C criou uma especificação dentro do nível 3 da CSS chamado Módulo *Template Layout*, com o objetivo de melhorar a organização dos elementos e informações referentes ao *layout* da aplicação web.

Com essa nova característica é possível dividir a construção do *layout* em duas partes distintas:

- 1) Definição dos elementos mestres e um *grid* de formatação como base;
- 2) Formatação geral dos elementos.

As propriedades criadas com essa especificação são associadas a uma característica de *layout* de um elemento, formando um *grid* invisível para posicionar o elemento e seus descendentes dentro do *layout*.

Isso permite um maior controle e flexibilidade na adaptação da aplicação web, aos vários dispositivos existentes.

Suporte dos Browsers

Característica CSS 3					
	v5	v9	v10	v5.1	v15
Módulo Template Layout	x	x	x	x	x

Essa característica não é suportada pelos *browsers* atuais por ser uma especificação rascunho do W3C.

Porém, existe um *plug in* desenvolvido em *jquery* que simula os resultados dessa especificação, e pode ser baixado no endereço <http://code.google.com/p/css-template-layout/>

Vale a pena entendermos essa especificação, pois ela será de grande utilidade num futuro próximo.

Módulo Template Layout

A função do Módulo *Template Layout* é definir *slots* (encaixes) dentro da estrutura do *layout*, para o posicionamento dos elementos HTML.

A ligação com os elementos HTML do *layout* é feita com duas propriedades conhecidas da CSS 2, *display* e *position*, porém, com valores e funcionalidades acrescidas as já existentes.

Pseudo-elemento ::slot

Esse pseudo-elemento define um encaixe para um elemento HTML, permitindo que seja feito um endereçamento individual para o mesmo com um conjunto de regras CSS definidas.

A sintaxe para definição de um *slot* é:

Exemplo:

CSS

```
...
<style>
  body::slot(a) {background-color: #006633; }
</style>
...
```

Propriedade display

Essa propriedade define a organização dos *slots* criados, formando um *grid* invisível da estrutura do *layout*.

Com a propriedade *display* posicionamos os elementos conforme a estrutura de layout desejado, usando como referência o modelo de tabela, colocando os elementos em colunas e linhas.

A diferença do modelo tradicional de tabela e o usado pelo pseudo-elemento *slot*, é que o número de linhas e colunas não dependem do conteúdo sendo fixada pelo valor da propriedade, e a ordem dos descendentes no código não é relevante.

Os valores da propriedade *display* podem ser configurados por:

- ✓ a (letra)
- ✓ @ (arroba)
- ✓ . (ponto)

Cada letra que definimos na propriedade *display* representa um *slot* diferente na estrutura do *layout*, e quando a repetimos no sentido horizontal e vertical estamos fazendo referência a um mesmo *slot*.

A repetição da letra para um slot, guardando as proporções, é como configurar o atributo *rowspan* ou *colspan* que expande linhas ou colunas dentro do elemento *table* HTML.

O sinal de @ (arroba) define um sinal para o *slot* referido.

Já o ." (ponto) define um espaço em branco no referido *slot*.

Devemos observar algumas regras no uso da propriedade *display* para não haver a geração de erro ou não interpretação pelo *browser*. São elas:

- ✓ Não é possível criar um slot que não seja retangular;
- ✓ Não é criar vários slots com a mesma letra;
- ✓ Não é permitido usar mais de um @ (arroba) em um template.

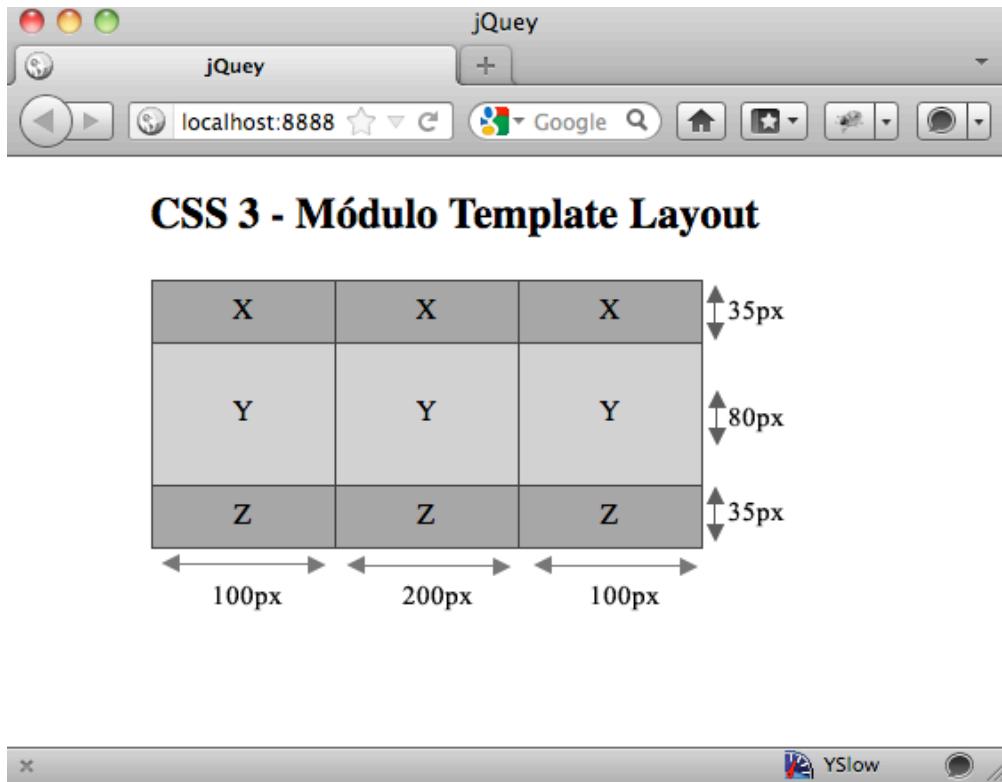
Quando um slot é criado podemos definir sua altura diretamente na propriedade *display*, com a seguinte sintaxe:

```
...
<style>
  1. display: "x      x      x" /35px
  2.         "y      y      y" /80px
  3.         "z      z      z" /35px
  4.     100px   200px  100px
</style>
...
```

O código acima pode ser interpretado da seguinte maneira:

1. Foram definidos 3 *slots* diferentes na propriedade *display* x,y e z;
2. As medidas na lateral das linhas 1, 2 e 3 referem-se a altura das mesmas.
3. Os valores colocados na 4^a linha referem-se a largura de cada uma das colunas.

A linha que não possui uma altura específica é definida pelo seu conteúdo.



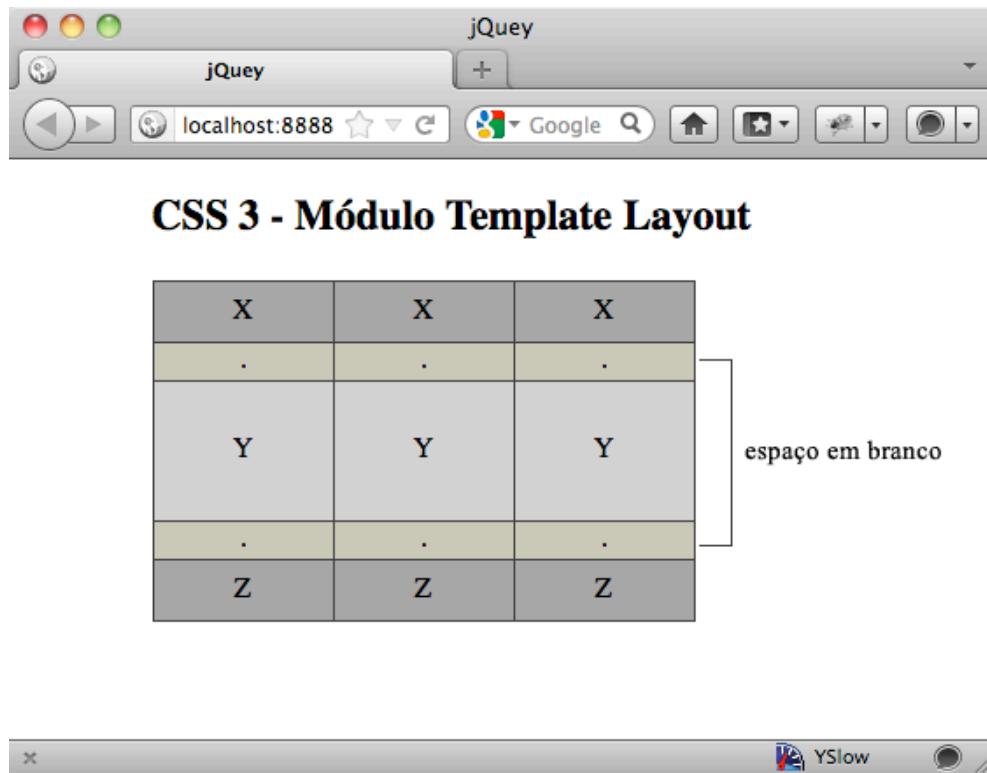
Quando é necessário acrescentar espaços na estrutura do *layout* podemos usar o ponto na propriedade *display*, para fazer a composição do *slot*.

A sintaxe para a inserção do ponto é a seguinte:

```
...
<style>
  1. display: "x      x      x" /35px
  2.      ".      .      ." /15px
  3.      "y      y      y" /80px
  4.      ".      .      ." /15px
  5.      "z      z      z" /35px
  6.      100px   200px   100px
</style>
...
```

Nas linhas 2 e 4 são definidos slots com espaços em branco configurados, e suas alturas definidas em 15px cada uma.

Obs.: Na definição de valores podem ser usadas as mesmas medidas da CSS como porcentagem e *em*.



Propriedade position

Essa propriedade define qual a linha e coluna será usada como posicionamento do elemento dentro *template*. É definida uma letra para cada elemento, e podem ser inseridos vários elementos em um mesmo *slot*.

Se vários elementos forem inseridos em um mesmo *slot*, e houver uma propriedade *position* para esse *slot*, todos os elementos terão esse mesmo valor.

A sintaxe para definição da propriedade *position* é:

```
...
<style>
  body{
    display: "x      y"
              "100px   250px";
  }
  #header{
    position: x;
  }
  #content{
    position: y;
  }
</style>
...
```

Em seguida, será apresentado um exemplo de código usando o *Módulo Template Layout* como estrutura de montagem de *layout*.

Em primeiro lugar, temos que baixar o *plugin* para simular os efeitos CSS do *Módulo Template Layout* no endereço <http://code.google.com/p/css-template-layout/>.

Abaixo o código HTML:

```
...
<body>
<div class="geral">
    <header>
        <h1>DRC Treinamentos Profissionais</h1>
    </header>
    <section>
        <article>
            <h2>Formação Front-End</h2>
            <p> A DRC Treinamentos criou uma formação técnica para profssionais que desejam atuar na função de front-end, com módulos específicos que fornecem os conhecimentos necessários para Design e Programação.</p>

            <p>Com os conhecimentos acima citados esse profissional está apto a integrar equipes de agências digitais, software-houses e departamento de TI em empresas envolvidas com aplicações web, sendo o responsável pela integração do projeto vindo do departamento de criação (layout) com a estrutura digital, e preparando-a para integração com a parte lógica chamada back-end (programação com o servidor e manipulação de banco de dados);</p>

            <p>A formação front-end DRC foi divida em 3 módulos, permitindo ao aluno a possibilidade de adquirir o conhecimento necessário ao seu momento profissional, possibilitando a atualização para novos níveis técnicos com os módulos avançados.<br>Além disso, temos workshops e palestras para atualização das tendências do mercado de tecnologia web.<br><br>Venha conhecer-nos!</p>
        </article>
    </section>

    <aside class="menulateral">
        <h3>Cursos</h3>
        <ul>
            <li><a href="">Design para Web</a></li>
            <li><a href="">HTML 5</a></li>
            <li><a href="">CSS 3</a></li>
            <li><a href="">Javascript</a></li>
            <li><a href="">jQuery</a></li>
            <li><a href="">PHP</a></li>
            <li><a href="">mySQL</a></li>
        </ul>
    </aside>
    <footer>
        <p>Rua Joaquim Floriano, 733, 8.º Andar Itaim Bibi São Paulo Tel.: 11 3168 2123 Fax: 11 3079 9485 </p>
    </footer>
    <aside class="pub">
        <p>Área reservada para elementos de publicidade.</p>
    </aside>
</div>
</body>
```



Javascript

```
...
<script src="jquery.js"></script>
<script>
$(document).ready(function() {
    $.setTemplateLayout(); // função do plug-in
});
</script>
...
```

CSS

```
...
<style>
* {
    margin: 0;
    padding: 0;
}

aside, section, article, header, footer{
    display: block;
}

body {
    font: normal 15px Arial, Helvetica;
}

.geral {
    width:980px;
    margin:auto;
    background:rgba(102,153,102,0.5);
    display: "x" "x" "x"
        ". . ." /10px
        "y" "g" "h"
        ". . ." /10px
        "z" "z" "z"
    150px * 150px;
}

.geral::slot(y) {
    background:rgba(102,102,204,1);
}

aside.menulateral {
    position:y;
    background:rgba(255,204,153,1);
    padding:10px;
}

aside.pub {
    position:h;
    background:rgba(153,153,102,1);
}

section {
    position:g;
    padding:10px;
    background:rgba(204,102,204,1);
}
```

```

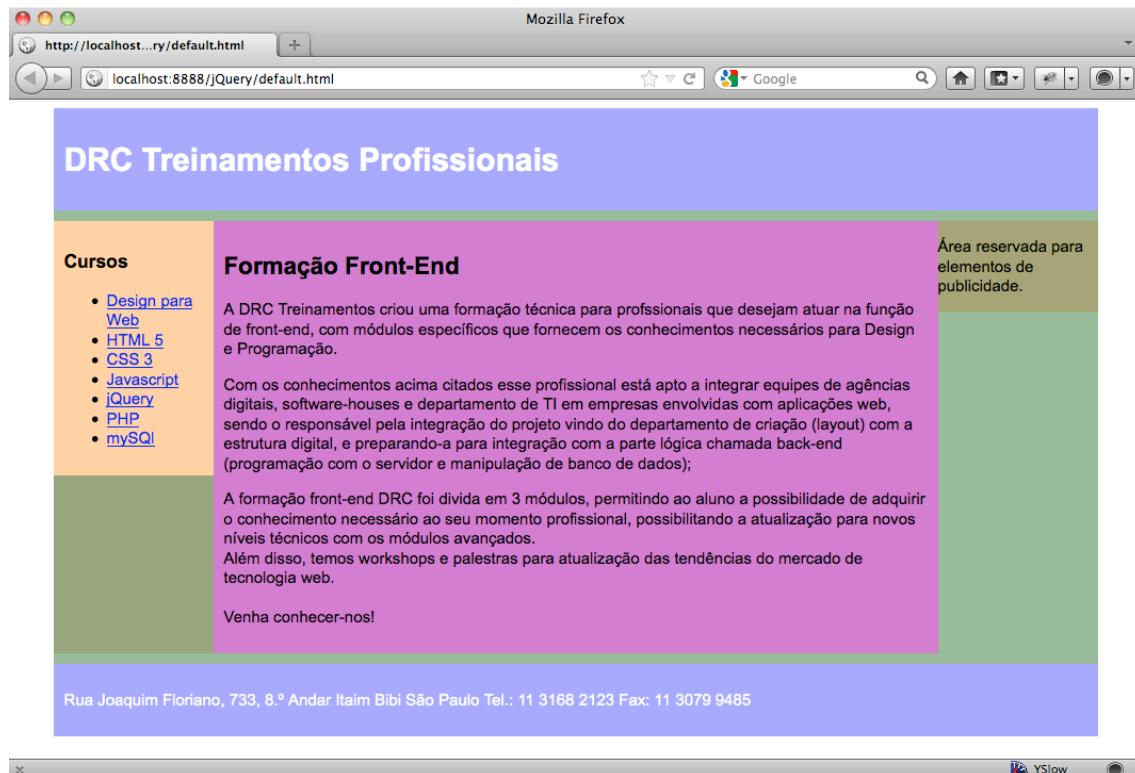
header {
    position:x;
}

footer {
    position:z;
}

header, footer {
    background:rgba(153,153,255,1);
    padding:10px;
    color:#FFF;
}

</style>
...

```



Como dito no início do capítulo, a especificação CSS 3 *Módulo Template Layout* ainda não está disponível nativamente nos browsers atuais.

Porém, como uma futura especificação a ser homologada, esse é o melhor momento de conhecê-la e compreendê-la, e por isso resolvemos incluí-la nessa apostila.



Capítulo 21: User Interface

* Visão Geral	181
* Suporte dos Browsers	181
* Propriedade resize	181
* Propriedade box-sizing	185
* Propriedade outline-offset	188

Visão Geral

O nível 3 da CSS trouxe características novas que interagem diretamente com alguns elementos da Interface do Usuário, como aparência, caixas de tamanho ajustável ao layout, bordas e deslocamentos.

Podemos usar essas novas características para incrementar o processo de desenvolvimento do *layout* da interface e sua interatividade com o usuário.

Suporte dos Browsers

Característica CSS 3					
	v5	v9	v10	v5.1	v15
resize	✓	✗	✗	✓	✓
box-sizing	✓	✓	✓	✓	✓
outline-offset	✓	✗	✓	✓	✓

Propriedade resize

Essa propriedade define se um elemento é redimensionável pelo usuário ou não.
Os valores da propriedade *resize* são:

none

Não é permitido ao usuário redimensionar o elemento especificado.

O *browser* omite o ícone colocado no canto inferior direito do elemento que permite ao usuário redimensioná-lo com o *mouse*.

Exemplo:

CSS

```
...
<style>
div{
    border:1px solid #000;
    padding:10px;
    width:300px;
    resize:none;
    overflow:auto;
    height: 21px;
}
</style>
...
```

HTML

```
...
<body>
    <h1>CSS 3 - User Interface</h1>
    <div>Aplicando a propriedade resize</div>
</body>
...
```



both

Permite ao usuário redimensionar o elemento especificado por sua largura e altura, ou seja, ambas.

O *browser* coloca um ícone no canto inferior direito do elemento que permite ao usuário redimensioná-lo com o *mouse*.

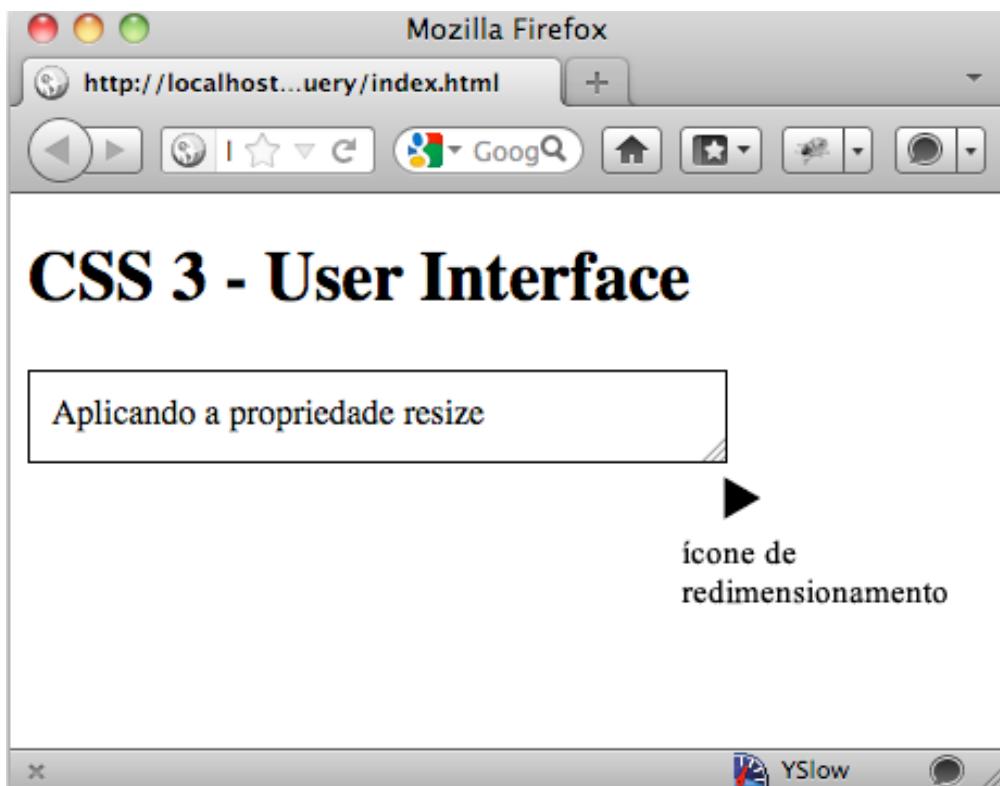
Exemplo:

CSS

```
...
<style>
  div{
    border:1px solid #000;
    padding:10px;
    width:300px;
    resize:both;
    overflow:auto;
    height: 21px;
  }
</style>
...
```

HTML

```
...
<body>
  <h1>CSS 3 - User Interface</h1>
  <div>Aplicando a propriedade resize</div>
</body>
...
***
```



horizontal

Permite ao usuário redimensionar o elemento especificado por sua largura.

O *browser* coloca um ícone no canto inferior direito do elemento que permite ao usuário redimensioná-lo com o *mouse*.

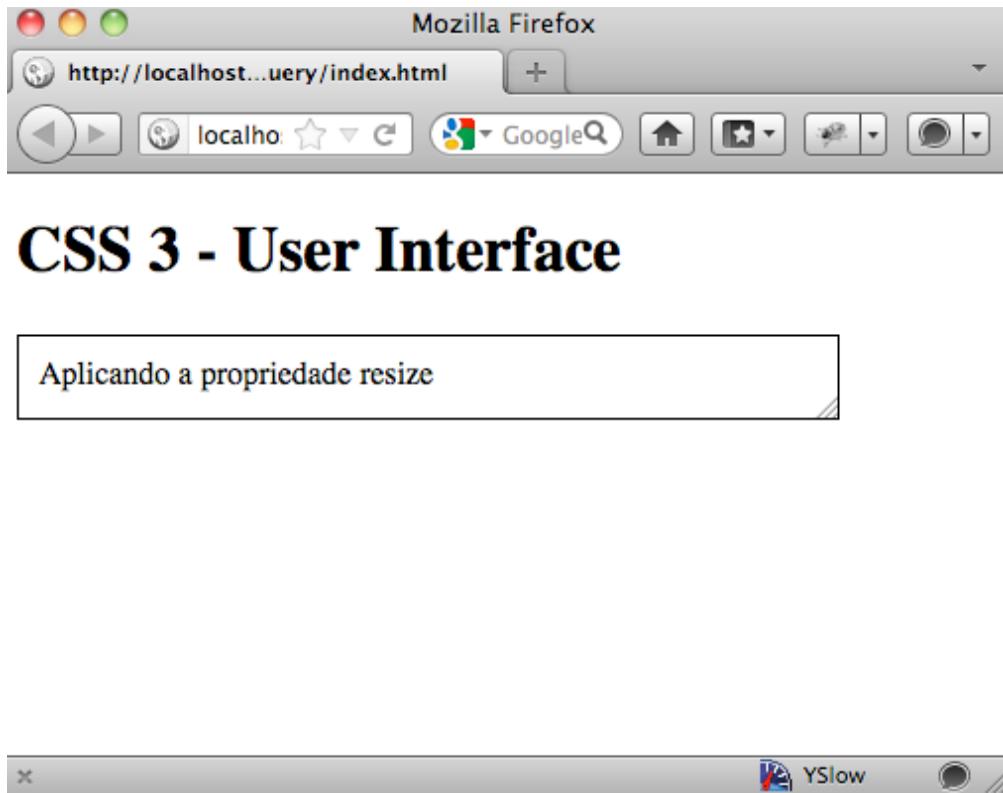
Exemplo:

CSS

```
...
<style>
div{
    border:1px solid #000;
    padding:10px;
    width:300px;
    resize:horizontal;
    overflow:auto;
    height: 21px;
}
</style>
...
```

HTML

```
...
<body>
    <h1>CSS 3 - User Interface</h1>
    <div>Aplicando a propriedade resize</div>
</body>
...
```



vertical

Permite ao usuário redimensionar o elemento especificado por sua altura.

O *browser* coloca um ícone no canto inferior direito do elemento que permite ao usuário redimensioná-lo com o *mouse*.

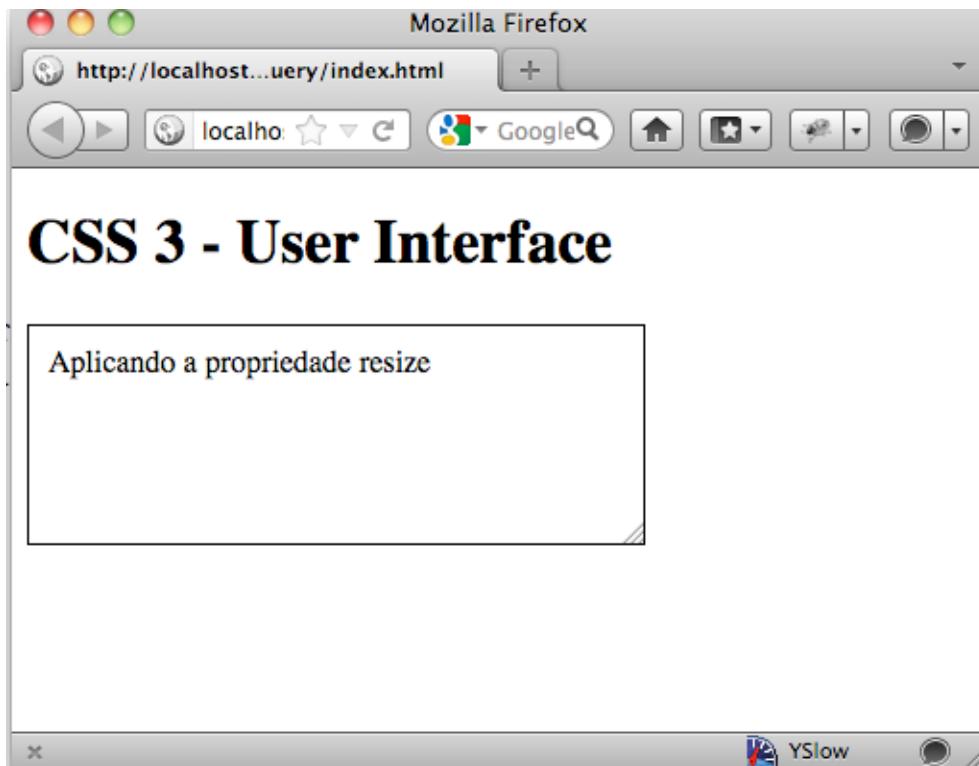
Exemplo:

CSS

```
...
<style>
div{
    border:1px solid #000;
    padding:10px;
    width:300px;
    resize:vertical;
    overflow:auto;
    height: 21px;
}
</style>
...
```

HTML

```
...
<body>
    <h1>CSS 3 - User Interface</h1>
    <div>Aplicando a propriedade resize</div>
</body>
...
```



Propriedade box-sizing

A propriedade *box-sizing* é usada para alterar o padrão do modelo de caixa CSS usado para calcular as larguras e alturas de elementos. Essa propriedade possui valores que determinam como o elemento vai se comportar dentro da estrutura do *layout*. São eles:

content-box

Esse valor é definido como estilo padrão. Nele as propriedades de largura e altura são calculadas levando em consideração somente o conteúdo do elemento, excluindo preenchimentos, bordas e margens.

Exemplo:

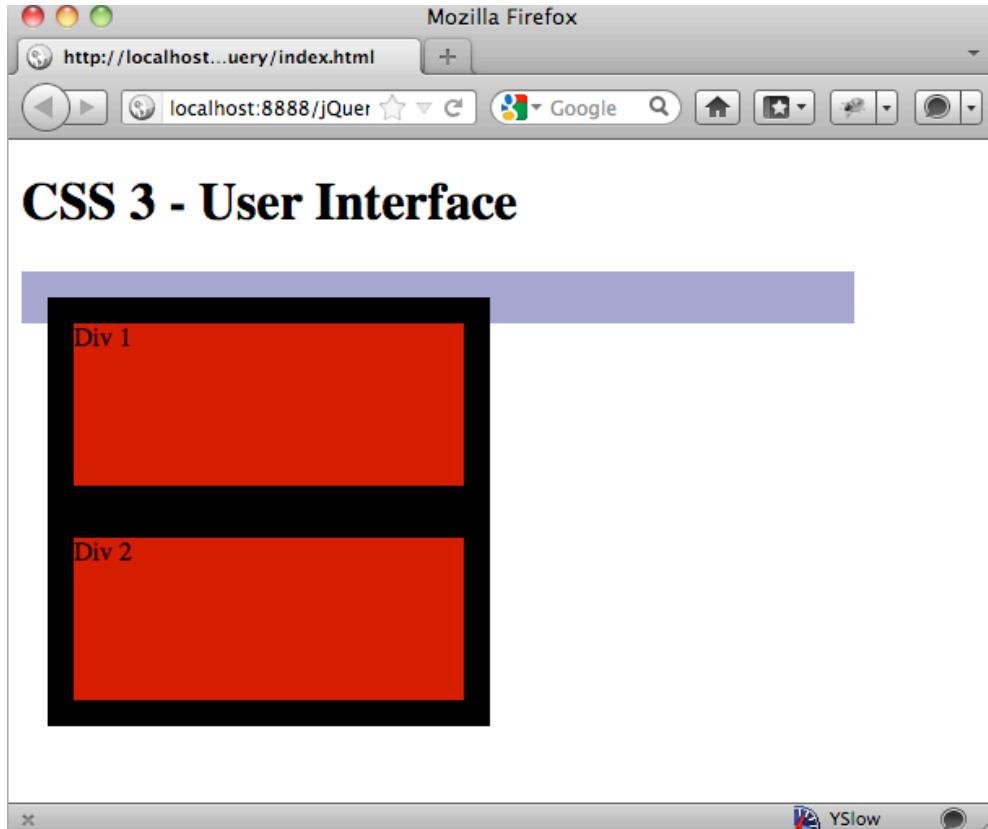
CSS

```
...
<style>
  div.main{
    width: 80%;
    border:1em solid #99C;
  }

  div.box{
    box-sizing:content-box;
    -moz-box-sizing:content-box;
    -webkit-box-sizing:content-box;
    width:50%;
    height: 100px;
    float:left;
    background: #C00;
  }
</style>
...
```

HTML

```
...
<body>
  <h1>CSS 3 - User Interface</h1>
  <div class="main">
    <div class="box">Div 1</div>
    <div class="box">Div 2</div>
  </div>
</body>
...
```



border-box

Esse valor calcula a largura e altura do elemento *container* incluindo o preenchimento e a borda, e excluindo apenas a margem do mesmo. Esse modelo de caixa é usado pelo Internet Explorer quando ele entra no modo *Quirks Mode*.

Exemplo:

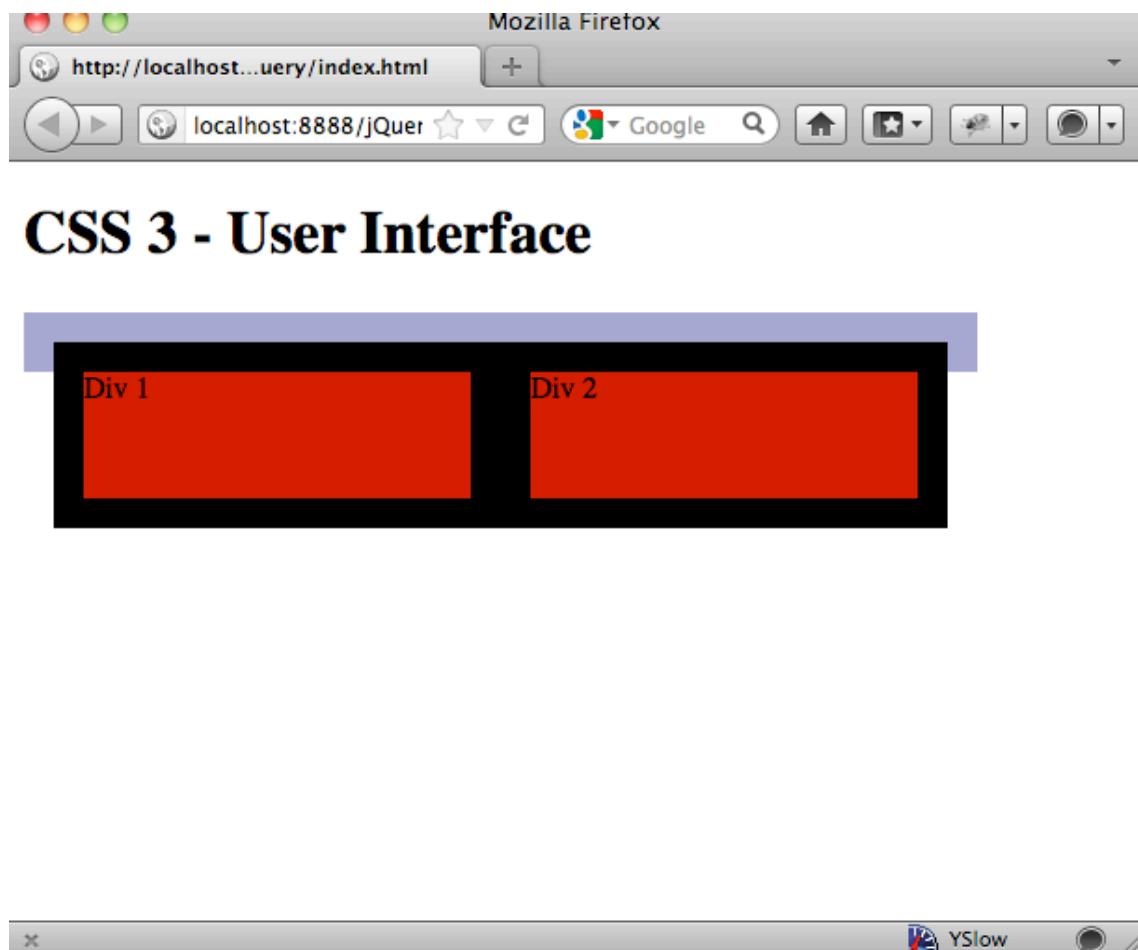
CSS

```
...
<style>
  div.main{
    width: 80%;
    border:1em solid #99C;
  }
...
```

```
div.box{  
    box-sizing:border-box;  
    -moz-box-sizing:border-box;  
    -webkit-box-sizing:border-box;  
    width:50%;  
    height: 100px;  
    float:left;  
    background: #C00;  
}  
</style>  
...
```

HTML

```
...  
<body>  
    <h1>CSS 3 - User Interface</h1>  
    <div class="main">  
        <div class="box">Div 1</div>  
        <div class="box">Div 2</div>  
    </div>  
</body>  
...
```



Propriedade outline-offset

Essa propriedade é usada para definir o espaço entre um esboço e a margem, ou borda, de um elemento. Um esboço é uma linha desenhada em torno do elemento, e fora da borda limite do mesmo.

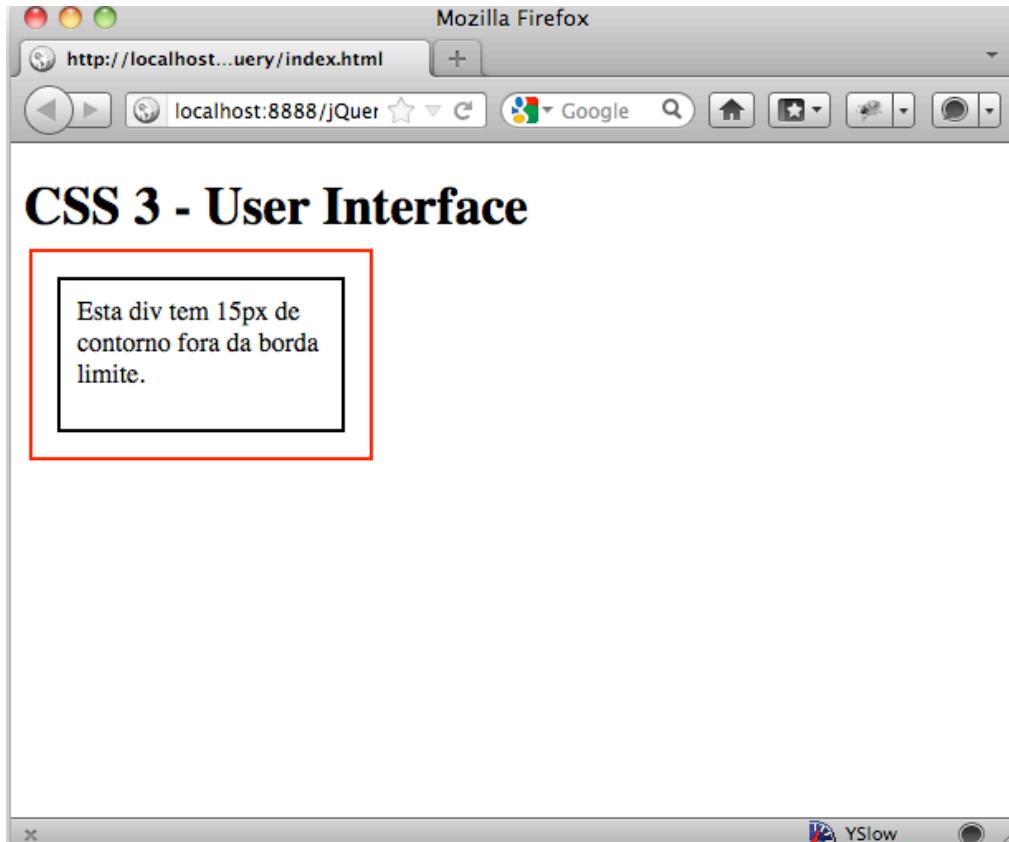
Exemplo:

CSS

```
...
<style>
  p{
    margin:20px;
    width:150px;
    padding:10px;
    height:70px;
    border:2px solid #000;
    outline:2px solid #F00;
    outline-offset:15px;
  }
</style>
...
```

HTML

```
...
<body>
  <h1>CSS 3 - User Interface</h1>
  <p>Este parágrafo tem 15px de contorno fora da borda limite.</p>
</body>
...
```



Referências Bibliográficas

Foram usadas no desenvolvimento dessa apostila algumas referências externas, cujo *link* colocamos abaixo:

W3C Consortium

<http://www.w3.org/TR/CSS/>

W3 Schools

<http://www.w3schools.com/css3/default.asp>

Mozilla Developer Network

<https://developer.mozilla.org/en-US/>

Webkit Open Source Project

<http://www.webkit.org/>



Moacyr Minero

moacyrminero

@moaminero

moacyrminero@hotmail.com

Informações sobre o Autor

● Formação Acadêmica

Publicidade e Propaganda (UNIP)

Processos Gerenciais (UNICID)

● Ocupação Profissional

Web Developer e Coordenador Web do Centro de Treinamentos DRC.

Palestrante sobre Padrões Web e Desenvolvimento HTML 5 para iPhone e iPad.

Autor de material didático para área de web.

● Conhecimento Técnico

HTML 5, CSS 3, Javascript, jQuery, AJAX, Desenvolvimento de App para iPhone/iPad, PHP, mySQL e Orientação a Objeto.