

HTML5

**A LINGUAGEM DE MARCAÇÃO
QUE REVOLUCIONOU A WEB**

Maurício Samy Silva

Novatec

HTML 5

**A LINGUAGEM DE MARCAÇÃO
QUE REVOLUCIONOU A WEB**

Maurício Samy Silva

Novatec

HTML 5

**A LINGUAGEM DE MARCAÇÃO
QUE REVOLUCIONOU A WEB**

Maurício Samy Silva

Novatec

Copyright © 2011 da Novatec Editora Ltda.

Todos os direitos reservados e protegidos pela Lei 9610 de 19/02/1998.
É proibida a reprodução desta obra, mesmo parcial, por qualquer processo, sem prévia autorização,
por escrito, do autor e da Editora.

Editor: Rubens Prates

Capa: Carolina Kuwabata

Revisão gramatical: Débora Facin

Editoração eletrônica: Camila Kuwabata e Carolina Kuwabata

ISBN: 978-85-7522-261-4

Histórico de impressões:

Fevereiro/2013	Terceira reimpressão
Maio/2012	Segunda reimpressão
Novembro/2011	Primeira reimpressão
Julho/2011	Primeira edição

Novatec Editora Ltda.

Rua Luís Antônio dos Santos 110
02460-000 – São Paulo, SP – Brasil

Tel.: +55 11 2959-6529

Fax: +55 11 2950-8869

E-mail: novatec@novatec.com.br

Site: www.novatec.com.br

Twitter: twitter.com/novateceditora

Facebook: facebook.com/novatec

LinkedIn: linkedin.com/in/novatec

Dados Internacionais de Catalogação na Publicação (CIP)
(Câmara Brasileira do Livro, SP, Brasil)

Silva, Maurício Samy
HTML 5 / Maurício Samy Silva. -- São Paulo :
Novatec Editora, 2011.

Bibliografia.
ISBN 978-85-7522-261-4

1. HTML (Linguagem de programação para
Internet) 2. Internet (Rede de computadores)
3. Websites - Desenvolvimento I. Título.

11-06541

CDD-005.133

Índices para catálogo sistemático:

1. HTML : Linguagem de programação para
Internet : Desenvolvimento de sites : Ciência
da computação 005.133
vc20130221

Sumário

Agradecimentos	12
Declaração de responsabilidade	12
Sobre o autor	13
Introdução	15
Capítulo 1 ■ Apresentação da HTML5	20
1.1 Introdução	20
1.2 Histórico	22
1.3 Criação da HTML5	26
1.4 Estrutura da especificação para a HTML5	27
1.5 Princípios de desenvolvimento da HTML5	28
1.5.1 Compatibilidade	28
1.5.2 Utilidade	32
1.5.3 Interoperabilidade	34
1.5.4 Acesso universal	35
1.6 Diferenças entre HTML5 e HTML4	36
1.6.1 Introdução	36
1.6.2 Sintaxe	37
1.7 A HTML5 nos navegadores atuais	39
1.7.1 Os novos elementos e o Internet Explorer	39
1.7.2 Os novos elementos e o modelo de renderização	40
1.7.3 Biblioteca JavaScript Modernizr	40
1.8 Templates HTML5	42
1.8.1 Template mínimo	46
1.8.2 Template HTML5 – versão 1	48
1.8.3 Template HTML5 – versão 2	49
1.8.4 Template XHTML5 – versão 1	50
1.8.5 Template XHTML5 – versão 2	51
1.8.6 Template XHTML5 – versão 3	52
Capítulo 2 ■ Novidades na HTML5	54
2.1 Introdução	54
2.2 Modelo de conteúdo	54

2.3 Atributos globais	56
2.3.1 accesskey	56
2.3.2 class	57
2.3.3.contenteditable	57
2.3.4 contextmenu	58
2.3.5 dir	59
2.3.6 draggable	59
2.3.7 hidden	60
2.3.8 id	60
2.3.9 item*	61
2.3.10 lang	61
2.3.11 spellcheck	61
2.3.12 style	62
2.3.13 tabindex	62
2.3.14 title	62
2.4 Novos elementos	62
2.4.1 article	62
2.4.2 aside	63
2.4.3 audio	64
2.4.4 canvas	64
2.4.5 command	64
2.4.6 menu	65
2.4.7 datalist	67
2.4.8 details	68
2.4.9 summary	69
2.4.10 figure	70
2.4.11 figcaption	70
2.4.12 footer	71
2.4.13 header	73
2.4.14 hgroup	74
2.4.15 keygen	75
2.4.16 mark	75
2.4.17 meter	76
2.4.18 nav	77
2.4.19 output	78
2.4.20 progress	80
2.4.21 ruby	81
2.4.22 section	81
2.4.23 source	82
2.4.24 time	82
2.4.25 track	83
2.4.26 video	83

Capítulo 3 ■ Áudio e vídeo	84
3.1 audio.....	84
3.2 video	91
3.2.1 track	99
3.2.2 Codificação de vídeos	101
3.3 API para áudio e vídeo.....	113
3.3.1 Atributos	115
3.3.2 Métodos	123
3.3.3 Eventos	124
Capítulo 4 ■ Canvas	140
4.1 Canvas	140
4.1.1 API de canvas.....	144
Capítulo 5 ■ Atributos HTML	171
5.1 Atributos globais	171
5.1.1 accesskey	171
5.1.2 class	173
5.1.3 contenteditable	173
5.1.4 contextmenu	174
5.1.5 dir	174
5.1.6 draggable	175
5.1.7 dropzone	176
5.1.8 hidden	176
5.1.9 id	177
5.1.10 itemid, itemprop, itemref, itemscope, itemtype.....	178
5.1.11 lang	178
5.1.12 spellcheck	178
5.1.13 style.....	179
5.1.14 tabindex.....	179
5.1.15 title	180
5.2 Novos atributos.....	180
5.2.1 async.....	181
5.2.2 autocomplete	181
5.2.3 autofocus	181
5.2.4 autoplay e controls	182
5.2.5 challenge	182
5.2.6 default	182
5.2.7 dirname	183
5.2.8 form	183
5.2.9 formaction	183
5.2.10 formenctype	183
5.2.11 formmethod	184
5.2.12 formnovalidate	184

5.2.13 formtarget	184
5.2.14 icon	185
5.2.15 keytype	185
5.2.16 kind	185
5.2.17 list	186
5.2.18 loop	186
5.2.19 low	187
5.2.20 manifest	187
5.2.21 max	187
5.2.22 min	188
5.2.23 novalidate	188
5.2.24 open	188
5.2.25 optimum	188
5.2.26 pattern	188
5.2.27 ping	189
5.2.28 placeholder	189
5.2.29 poster	190
5.2.30 preload	190
5.2.31 pubdate	190
5.2.32 radiogroup	190
5.2.33 required	191
5.2.34 reversed	191
5.2.35 sandbox	192
5.2.36 scoped	193
5.2.37 seamless	193
5.2.38 sizes	194
5.2.39 srctdoc	194
5.2.40 srclang	195
5.2.41 step	195
5.2.42 wrap	195
Capítulo 6 ■ Formulários	196
6.1 Introdução	196
6.2 Atributos	197
6.2.1 placeholder	198
6.2.2 autofocus	198
6.2.3 required	198
6.2.4 autocomplete	199
6.2.5 dirname	200
6.2.6 form	201
6.2.7 formaction	202
6.2.8 formenctype	202
6.2.9 formmethod	202
6.2.10 formnovalidate	203

1.11 formtarget	203
1.12 list	204
1.13 max	204
1.14 min	204
1.15 novalidate	205
atributo type para elemento input	205
1.3.1 search	205
1.3.2 tel	206
1.3.3 url	207
1.3.4 email	208
1.3.5 datetime	209
1.3.6 date	209
1.3.7 time	210
1.3.8 month	210
1.3.9 week	211
1.3.10 datetime-local	211
1.3.11 number	212
1.3.12 range	212
1.3.13 color	213
Capítulo 7 ■ Geolocalização	214
7.1 Introdução	214
7.2 Segurança e privacidade	215
7.3 Interface Geolocation	216
7.4 Métodos do objeto geolocation	217
7.4.1 getCurrentPosition(callback sucesso, callback erro, {opções posição})	217
7.4.2 watchPosition(callback sucesso, callback erro, {opções posição})	223
7.5 Google Maps API	224
7.5.1 Introdução	224
7.5.2 API do Google Static Maps	224
7.5.3 API V3 do Google Maps	229
Capítulo 8 ■ Armazenamento de dados	233
8.1 Web Storage	233
8.1.1 sessionStorage	235
8.1.2 localStorage	237
8.1.3 Armazenagem com JSON	241
8.1.4 Evento storage	244
8.1.5 Banco de dados	246
8.2 Verificando suporte	251
Capítulo 9 ■ APIs para comunicação	252
9.1 Web Messaging	252
9.2 Web Socket	258

9.2.1 Propriedades e métodos	260
9.2.2 Eventos.....	261
9.3 Web Workers	263
Capítulo 10 ■ Offline	275
10.1 Introdução	275
10.2 Manifesto	276
10.2.1 Atualização do manifesto	278
10.3 API Application cache	279
10.3.1 Propriedades e métodos.....	280
10.3.2 Eventos	281
10.3.3 Atualizando o cache.....	283
10.4 Verificando suporte.....	284
10.5 Exemplo prático	284
Capítulo 11 ■ Acessibilidade.....	287
11.1 WAI-ARIA.....	287
11.1.1 Role.....	288
11.1.2 Atributos	294
11.2 Microdados	297
11.2.1 Sintaxe.....	298
11.3 Microdados DOM API	301
11.3.1 Suporte nos navegadores	302
Apêndice A ■ Elementos da HTML5	303
Apêndice B ■ Atributos da HTML5.....	307
Apêndice C ■ Atributos para eventos da HTML5	311
Referências	314
Índice remissivo	315

Ao meu neto, Leonardo, e aos seus pais, Carol e Rogério.

Agradecimentos

Agradeço a Deus, por ter me dado forças, disposição e motivação para escrever este livro.

Sou grato aos profissionais da Novatec Editora, em particular ao editor, Rubens Prates, que, ao longo de todo o processo de criação, esteve presente guiando-me com suas dicas e informações sobre as particularidades e implicações editoriais próprias à criação de um livro.

Meu maior agradecimento é a você, leitor, por interessar-se em aprender HTML5 e honrar-me com a leitura deste livro.

Isenção de responsabilidade

Todos os esforços foram feitos para assegurar o fornecimento de informações as mais precisas, completas e exatas possíveis neste livro. Contudo, tais informações são fornecidas “como estão” e sem nenhuma garantia, seja expressa, seja implícita. O autor, a editora, os distribuidores e qualquer entidade envolvida direta ou indiretamente na sua comercialização não assumirão responsabilidade alguma por eventual prejuízo ou dano, direto ou indireto, consequente aos dados contidos neste livro.

Sobre o autor

Mauricio Samy Silva é graduado em Engenharia Civil pelo Instituto Militar de Engenharia (IME). Fundador e ex-diretor-presidente da Planep Engenharia, exerceu o magistério paralelamente à Engenharia, foi, ao longo de 25 anos, professor de Geometria Descritiva e Matemática. Seu primeiro contato com programação para computadores deu-se ainda na graduação, ao aprender a linguagem Fortran. Naquela época, era comum passar noites trabalhando em uma máquina perfuradora de cartões que seriam processados em um poderoso IBM/360. Em 1982, adquiriu seu primeiro computador pessoal, um TK-80 com processador Z-80A de 3,25MHz, teclado de membrana com 40 teclas e memória RAM de 1KB expansível até 16KB. Trabalhou com os modelos TK-82, TK-85, TK-2000, DGT-100 e assim por diante, até os dias atuais.

Sua experiência com desenvolvimento de sites iniciou-se em 1999, com o FrontPage, considerada uma ferramenta sensacional na construção de sites. Em 2002, por acaso, teve seu primeiro contato com o site do W3C e, como ficou vivamente impressionado com a proposta desse consórcio, começou a pesquisar e a estudar as Web Standards, tendo como fonte de consulta o material publicado na internet em língua inglesa. Ao contrário do que ocorre nos dias atuais, em que vários desenvolvedores escrevem em blogs pessoais sobre Web Standards, a literatura a respeito do assunto em português era, então, praticamente nula, mas, mesmo assim, quando encontrada, limitava-se ao básico do básico.

No segundo semestre de 2003, estimulado pela completa falta de material de consulta gratuita na internet e impulsionado por sua veia de educador desenvolvida durante anos em sala de aula, decidiu lançar o site CSS para Web Design, o qual é nacionalmente conhecido como o site Maujor, hospedado em <http://www.maujor.com/>. A proposta inicial do site era divulgar a Cascading Style Sheet (CSS), ou folha de estilo em cascata, com base no compartilhamento de suas experiências com tal técnica. Com a pronta aceitação e o sucesso crescente do site, o objetivo inicial tornou-se mais ambicioso e passou a ser a divulgação das Web Standards.

No início de 2006, com a popularização e a adesão maciça aos blogs, criou o Blog do Maujor, hospedado em <http://www.maujor.com/blog/>, com o propósito semelhante ao do site, mas com uma dinâmica mais ativa e a efetiva participação de seus leitores.

Tanto o site quanto o blog constituem referência nacional para Web Standards e, sem dúvida, foram e continuam sendo um dos grandes responsáveis pela divulgação e popularização das Web Standards, assim como, em particular, das CSS no Brasil.

Maujor, como é conhecido o autor, é ainda um ativo frequentador de fóruns, escreve para vários portais brasileiros voltados a desenvolvedores web, é autor de artigos em inglês e de trabalhos relacionados às CSS publicados em sites internacionais sobre Web Standards. Ocupa o segundo lugar na lista de tradutores mundiais de documentos do W3C por quantidade de documentos versados. Publicou em seu site mais de 40 traduções de artigos sobre Web Standards escritos por consagrados autores internacionais. Traduziu para o português as ferramentas para testes de acessibilidade em sites, denominadas analisador de contraste de cores e barra de ferramentas para acessibilidade na web, ambas em parceria com o WAT-C, um consórcio australiano voltado ao desenvolvimento de ferramentas destinadas a testes de acessibilidade.

Maujor é autor dos seguintes livros: *Construindo sites com CSS e (X)HTML; Criando sites com HTML; JQuery – A biblioteca do programador JavaScript; AJAX com jQuery; JavaScript – Guia do Programador; CSS3 – Desenvolva aplicações web profissionais com uso dos poderosos recursos de estilização das CSS3; jQuery Mobile; e jQuery UI*, todos publicados pela Novatec Editora.

Introdução

O objetivo deste livro é fornecer aos profissionais envolvidos no desenvolvimento web os conceitos fundamentais para a criação de sites e aplicações com uso da linguagem de marcação HTML5. Trata-se de um livro pioneiro que mostra e explica as funcionalidades dessa linguagem de marcação e pretende fornecer ao leitor uma visão do estágio em que se encontra a especificação.

O livro possui 11 capítulos e apêndices. Os capítulos foram estruturados de modo que a consulta ao livro possa ser feita de acordo com o capítulo que interessa, sem que o entendimento do capítulo anterior seja pré-requisito para a compreensão dos capítulos posteriores.

Para quem foi escrito este livro

O livro destina-se a pessoas envolvidas na criação de sites tanto na área de design quanto na de desenvolvimento e programação, com conhecimento da linguagem HTML e noções básicas de JavaScript.

O objetivo é fornecer informações detalhadas do funcionamento da linguagem, estudando seus elementos e atributos, bem como apresentar as APIs que dela fazem parte ou estão relacionadas. Explicações teóricas em linguagem corrente e clara, suspensando, sempre que possível, o jargão técnico avançado, são acompanhadas de exemplos práticos explicados passo a passo e complementados por arquivo HTML para consulta, disponíveis online no site do livro em <http://livrohtml5.com.br> e também para download para consulta local.

Para tirar o máximo proveito dos ensinamentos contidos em cada capítulo, é necessário razoável conhecimento da linguagem de marcação HTML, de CSS e de JavaScript.

Os iniciantes irão se beneficiar deste livro por principiarem seus estudos em uma fonte que aborda as mais modernas técnicas de escrita do código, ensejando uma mudança no rumo de seu estudo que irá reduzir a curva de aprendizado e acelerar

tal processo. Não se intimide com conceitos ou terminologias que lhe sejam completamente desconhecidos nos primeiros capítulos. Com a sequência da leitura, as dúvidas tenderão a desaparecer naturalmente.

Convenções tipográficas

Com a finalidade de destacar diferentes tipos de informação, adotaram-se algumas convenções tipográficas mostradas a seguir.

Dica

Texto contendo uma dica sobre o assunto tratado:



No mês de outubro de 1994, Tim Berners-Lee em parceria com a CERN, onde a web foi por ele inventada, criaram o World Wide Web Consortium (W3C), com sede no Laboratório da Ciência da Computação do Massachusetts Institute of Technology (MIT).

Alerta

Texto contendo um lembrete sobre procedimento extra em relação ao assunto tratado:



Os atributos da API não são definitivos. Uns poderão ser abandonados; outros, acrescidos, pois as especificações para a HTML5 estão em fase de rascunho. Os atributos aqui mostrados são atualmente previstos na especificação, mas não significa que podemos usá-los indiscriminadamente em fase de produção.

Terminologia

Texto estabelecendo a adoção de grafia-padrão em todo o livro para termos ou frases com mais de uma terminologia, tradução ou significado:



O termo inglês *browser* é usado no jargão da internet para designar um programa capaz de ler e apresentar ao usuário os conteúdos de um documento web escrito em linguagem de marcação. Browser vem do verbo *to browse*, que significa folhear.

Chamada

Uma chamada para uma seção anterior na qual o assunto em questão foi explicado com detalhes.

Exemplo: Outra funcionalidade importante da biblioteca é que ela agregou o desenvolvido por Remy Sharp e mostrado no item 1.1.8 que faz com que o Internet Explorer...

Nesse exemplo, a chamada é para a seção 1.1.8 do capítulo 1.

Notas e scripts

Marcações e scripts que exemplificam a teoria, transcreveram-se somente os que interessam ao assunto tratado. Omitiram-se os trechos que não dizem respeito ou não são relevantes ao entendimento do assunto, para não ocupar espaço necessário no livro.

- Blocos de marcação são marcados com fonte monoespacada:

```
<article>
    <h1>Pequenas tarefas</h1>
    <footer>Publicado em <time pubdate>05/02/2011</time>. </footer>
    <p>Trocar a lâmpada da sala da sala.</p>
</article>
```

- Trechos de marcação que merecem destaque são marcados em negrito:

```
<div contenteditable="true">
    <p spellcheck="true">
        Conteúdo editável a ser verificado pelo corretor ortográfico
    </p>
</div>
```

Para explicar passo a passo cada linha de um script, este é apresentado com suas linhas numeradas e, a seguir, os comentários são referenciados ao número da linha comentada:

1. <script type="text/javascript">
2. function id(id) {
3. return document.getElementById(id);
4. }
5. function init () {
6. var container = id('fundo-controles');
7. var ctr = id('som');
8. var play = id('play');
9. var pause = document.getElementById('pause');
10. var vmais = document.getElementById('vmais');
- ...

Código comentado:

Linha(s)	Descrição
Linha 2 a 4	Cria a função <code>id()</code> que simplifica a sintaxe para <code>document.getElementById()</code> .
Linhas 5 a 31	Cria a função <code>init()</code> que contém o script a ser executado depois que a página é carregada. Na linha 32 chama-se essa função no evento <code>onload</code> .
Linhas 6 a 11	Cria variáveis para armazenar os elementos do DOM que serão manipulados pelo script.

Arquivos para download

Os códigos dos exemplos mostrados no livro estão disponíveis para download e/ou consulta on-line no site do livro, em <http://livrohtml5.com.br>. Os documentos estão separados em pastas por capítulos e foram nomeados com sintaxe própria, conforme o exemplo a seguir:

c3-audio-ex1.html

Esse é um arquivo do capítulo 3 contendo o primeiro exemplo demonstrativo de uso do elemento `audio`.

Adicionalmente, no final do script, há uma indicação do arquivo em que foi inserido, conforme convenção mostrada no exemplo a seguir:

```
<audio controls>
  <source src="som.ogg" type="audio/ogg">
  <source src="som.mp3" type="audio/mpeg">
  <source src="som.wav" type="audio/wave">
  <p>Seu navegador não suporta o elemento audio da HTML5.
  <br>Faça <a href="som.mp3">download de som.mp3</a></p>
</audio>
```



[c3-audio-ex1.html]

Destaques em geral

Palavras ou termos cujo significado deva ser destacado são grafados em itálico.

Por exemplo:

A presença do atributo `controls` faz com que o navegador renderize uma barra de controle nativa contendo botões do tipo *play* e *pause* bem como controle de volume.

Valores variáveis

Valores variáveis em códigos são grafados em itálico.

Por exemplo:

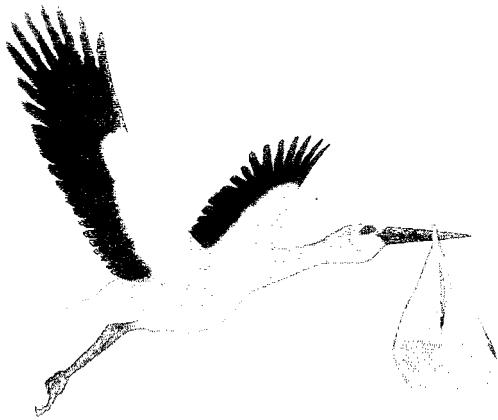
```
addTrack(tipo, legenda, idioma)
```

Site do livro

O site de suporte a este livro está localizado em <http://livrohtml5.com.br>.

No site, incluíram-se as facilidades relacionadas a seguir:

- **Arquivo de códigos:** os códigos dos exemplos mostrados no livro estão disponíveis tanto para consulta online quanto para download.
- **Errata:** efetuou-se um exaustivo trabalho de revisão tipográfica. Contudo, a prática mostra que nenhum livro está isento de erros – e com este não há de ser diferente. Uma página do site dedicada à errata encontra-se disponível para consulta.
- **Feedback:** incentivam-se vivamente os leitores a emitir opinião sobre qualquer aspecto do livro. Serão de grande valia informações sobre qualquer erro detectado para aperfeiçoar futuras edições e atualizar a errata. Você pode se comunicar com a editora pelo e-mail novatec@novatec.com.br ou diretamente com o autor pelo e-mail maujorcss@maujor.com.



CAPÍTULO 1

Apresentação da HTML5

Neste capítulo, será apresentada a linguagem de marcação HTML5, relatando-se suas origens, finalidades e destinação. Será feito um breve relato histórico de sua evolução, examinando as motivações que resultaram na sua criação, a filosofia que norteia o desenvolvimento das especificações para a linguagem e as principais diferenças conceituais para as versões anteriores da linguagem HTML.

1.1 Introdução

HTML é a sigla em inglês para *HyperText Markup Language*, que, em português, significa linguagem para marcação de hipertexto.

O conceito de hipertexto admite um sem-número de considerações e discussões que fogem ao escopo deste livro. Para o bom entendimento das definições, podemos resumir hipertexto como todo o conteúdo inserido em um documento para a web e que tem como principal característica a possibilidade de se interligar a outros documentos da web. O que torna possível a construção de hipertextos são os links, presentes nas páginas dos sites que estamos acostumados a visitar quando entramos na internet.

Desde a invenção da web por Tim Berners-Lee, a HTML evoluiu por oito versões que são:

- HTML
- HTML +
- HTML 2.0

- HTML 3.0
- HTML 3.2
- HTML 4.0
- HTML 4.01 (versão atual)
- HTML5 (versão em fase de desenvolvimento)

Tecnicamente, o W3C considera oficialmente somente as versões HTML 2.0, HTML 3.2, HTML 4.0, HTML 4.01 e HTML5. As versões HTML e HTML+ são anteriores à criação do W3C e a versão HTML 3.0 não chegou a ser lançada oficialmente, transformando-se na versão HTML 3.2.

A web foi inventada em 1992 por Sir Tim Berners-Lee. Atualmente Tim é diretor do World Wide Web Consortium (W3C), pesquisador sênior do Laboratório da Ciência da Computação e Inteligência Artificial (CSAIL) do Instituto de Tecnologia de Massachusetts (MIT) e professor de Ciência da Computação na Universidade de Southampton, na Inglaterra.

Tim Berners-Lee trabalhava na Seção de Computação da Organização Europeia de Pesquisa Nuclear (CERN), com sede em Genebra, na Suíça, quando iniciou pesquisas visando a descobrir um método que possibilitasse aos cientistas do mundo inteiro compartilhar eletronicamente seus textos e pesquisas e que tivesse a funcionalidade de interligar os documentos. Estavam criadas as noções *web* e de *links* como são conhecidas atualmente.

Em 1990, Tim criou o protótipo de um navegador para rodar em computadores da NeXT, uma companhia fundada em 1985 por Steve Jobs, atual CEO da Apple. Inicialmente, o navegador foi chamado de WorldWideWeb e, posteriormente, renomeado para Nexus, a fim de evitar confusão com a World Wide Web.



O termo inglês *browser* é usado no jargão da internet para designar um programa capaz de ler e apresentar ao usuário os conteúdos de um documento web escrito em linguagem de marcação. Browser vem do verbo *to browse*, que significa folhear casualmente as páginas de um livro e foi traduzido para o português como navegador, gerando a tão bem conhecida expressão “navegar na internet”. São exemplos de navegadores o Internet Explorer, o Firefox, o Opera, o Chrome e o Safari, entre outros. Neste livro, adotaremos o termo em sua forma traduzida: navegador.

Tim Berners-Lee acreditava que seria possível interligar hipertextos em computadores diferentes com uso de links globais, também chamados de hiperlinks. Ele desenvolveu um software próprio e um protocolo para recuperar hipertextos,

denominado HTTP. O formato de texto que criou para o HTTP foi chamado de HTML. Tim tomou como base para criação da HTML a especificação SGML, que é um método internacionalmente reconhecido e aceito, contendo normas gerais para a criação de linguagens de marcação. A marcação para hiperlinks conduzindo a documentos que não estivessem em um mesmo computador obviamente não constava das normas para SGML e foi inventada por Tim, demonstrada pela primeira vez em 1990, em uma estação de trabalho NeXT, nos laboratórios da CERN. Estava criado o embrião da World Wide Web, bem como a primeira versão da linguagem HTML para a marcação de hipertextos. A partir daí, a evolução cronológica da HTML deu-se conforme relatado sumariamente a seguir.

1.2 Histórico

Em setembro de 1991, foi criada a lista de discussão eletrônica denominada *WWW-talk*, com o propósito de trocar ideias e experiências sobre a HTML desenvolvida por Tim Berners-Lee. Um dos frequentadores da lista era Dave Raggett, dos laboratórios da Hewlett-Packard, em Bristol, Inglaterra. Dave, empolgado com a nova ideia, desenvolveu suas pesquisas e acabou por escrever a HTML+, uma versão elaborada e enriquecida da HTML original desenvolvida por Tim. Em outubro de 1993, Dave Raggett deu por encerrada as discussões e, no mês seguinte, publicou a versão final da HTML+.

A HTML+ começa com a seguinte afirmação:

Documentos marcados com HTML+ são constituídos de títulos, parágrafos, listas, tabelas e figuras.

E continua estabelecendo:

Ao contrário da maioria das tecnologias destinadas à criação de documentos, a HTML+ não se destina a determinar a aparência; assim, nomes e tamanhos de fontes, margens, tabulações, espaçamentos entre os elementos não são funções da linguagem. Fica a cargo dos softwares responsáveis pela renderização dos documentos marcados com HTML+ a maneira como os documentos devam ser apresentados (talvez com base em configurações de preferência do usuário).

Convém salientar com muita ênfase que, desde sua criação, os idealizadores da HTML tiveram a preocupação de retirar da linguagem de marcação qualquer atribuição ou função de apresentação, ou seja, HTML destina-se exclusivamente a estruturar documentos. É nessa destinação que se fundamentam os princípios básicos do desenvolvimento seguindo os Padrões Web.

Em 1992 marcou o interesse do Centro Nacional de Aplicações para Supercomputadores (NCSA) da Universidade de Illinois – então representado por Joseph Hardin e Dave Thompson, os quais acabaram por desenvolver um navegador que denominado Mosaic. Da NCSA participaram ainda dois expoentes da web: Marc Andreessen, que apresentou a ideia de um elemento para marcação de imagens nos hipertextos – e depois ficou milionário vendendo produtos para a web –, e Eric Bina, um brilhante programador considerado um gênio da web.

Em 1993, Lou Montulli criou o navegador de texto denominado Lynx versão 2.0a e Dave Raggett começou a trabalhar no desenvolvimento do navegador Arena, que se destinava a demonstrar, na prática, a implementação de todas as funcionalidades inventadas e discutidas até então. Ainda nesse ano a Sun Microsystems lançou a versão 1 do navegador Mosaic.

As grandes companhias não se interessaram pela nova invenção, alegando não acreditar que pudessem tirar proveito algum da web e que se tratava de um meio de comunicação restrito e de emprego exclusivo na área acadêmica. Tal pensamento desestimulou as pesquisas que vinham sendo desenvolvidas, pois estas dependiam de patrocínio. As equipes, que até então desenvolviam seus projetos em horas vagas, refrearam o ímpeto inicial por falta de tempo e verba. O trabalho continuou, mas em um ritmo aquém do esperado pelos envolvidos no projeto.

No mês de maio de 1994, a Spyglass comprou a licença de comercialização da versão aperfeiçoada do navegador Mosaic. Ainda nesse mês de maio, a CERN organizou, em Genebra, a primeira conferência para a World Wide Web com a apresentação da HTML+, criada por Dave Raggett. A conferência contou com a participação de aproximadamente 380 pessoas, a maioria envolvida com atividades e instituições acadêmicas.



No mês de outubro de 1994, Tim Berners-Lee em parceria com a CERN, onde a web foi por ele inventada, criaram o World Wide Web Consortium (W3C), com sede no Laboratório da Ciência da Computação do Massachusetts Institute of Technology (MIT).

Com a criação de novos navegadores, a HTML tornou-se um caos, com cada fabricante inventando novas formas de marcação HTML exclusivas para seus navegadores. Em uma tentativa de organizar a situação, Dan Connolly e colaboradores fizeram um levantamento minucioso de tudo o que existia na HTML e propuseram, em julho de 1994, a especificação HTML 2.0, uma tentativa de consolidar e unificar as diferentes formas HTML de marcação que vinham sendo criadas. Adicionalmente, Dan e sua equipe escreveram a primeira Definição do Tipo de Documento (DTD) para a HTML 2.0, uma espécie de descrição matemática da linguagem.

Em setembro, foi criado pela Força-Tarefa para Engenharia de Internet (IETF) o primeiro grupo de trabalho para a HTML. A IETF é uma organização internacional que congrega interessados nos vários aspectos da internet, com a finalidade de desenvolver normas e tecnologias pertinentes à operação e à evolução da arquitetura da internet. A criação desse grupo de trabalho marcou a abertura do desenvolvimento da HTML para qualquer pessoa interessada, independentemente de pertencer ou não a uma organização ou órgão particular, isto é, a discussão tornou-se aberta ao público em geral.

Marc Andreessen e Jim Clark haviam fundado a Mosaic Communications que, em novembro desse ano, transformou-se na Netscape Communications Corporation. Nascia a Companhia que se tornaria, em pouco tempo, a senhora absoluta da web, impulsionada com a aceitação unânime da nova versão do navegador Mosaic. O sucesso da Netscape deveu-se, entre outros fatores, primordialmente, ao investimento maciço nas funcionalidades de proporcionar acesso à internet, mesmo para os usuários com dispositivos mais precários. A Netscape tornou-se a sensação e muitos ainda acreditam erroneamente que a web foi criada pela Netscape. Por outro lado, a companhia isolou-se em suas pesquisas e desenvolvimentos, adotando a política de inventar HTML própria, sem participação da comunidade interessada e passando ao largo de conferências e encontros públicos relacionados.

No mês de outubro de 1994, foi criado o World Wide Web Consortium (W3C), um consórcio internacional formado por empresas, instituições, pesquisadores, desenvolvedores e público em geral, com a finalidade de desenvolver a web a seu potencial máximo, criando normas e especificações aplicáveis aos diversos segmentos e setores da web, desde tecnologias e softwares até fabricantes e fornecedores.

O W3C tem sua sede principal distribuída em três lugares distintos: nos Laboratórios de Ciência da Computação do MIT, em Massachusetts, nos Estados Unidos, no Instituto Nacional de Pesquisas de Informática e Automação, na França, e na Universidade de Keiko, no Japão, além de escritórios espalhados em várias cidades do mundo. As mais proeminentes e brilhantes cabeças envolvidas com o desenvolvimento para a web foram convidadas a formar o núcleo básico do W3C. Nomes de projeção internacional no envolvimento com a web foram convidados a participar, destacando-se Henrick Frystyk Neilsen, Anselm Baird-Smith, Jay Sekora, Rohit Khare, Dan Connolly, Jim Gettys, Tim Berners-Lee, Susan Hardy, Jim Miller, Dave Raggett, Tom Greene, Arthur Secret, Karen MacArthur, Arnaud le Hors, Håkon Lie, Bert Bos e Chris Lilley, entre outros.

No final de 1995 assinalou o início de um desenvolvimento frenético de novas funcionalidades para a HTML, com prioridade para a criação de elementos e atributos de apresentação em total desacordo com o propósito inicial da linguagem, qual seja, manter uma linguagem exclusivamente de marcação e estruturação de textos. Foi adicionado elementos para definir tamanhos, tipos e cores de letras dos textos, adicionar som, texturas e toda uma parafernália relacionada, completamente fora do propósito inicial da HTML.

A versão final da HTML 2.0 foi publicada em 22 de setembro de 1995.

No março de 1995, Dave Raggett lançou sua proposta para a HTML 3.0, que vem com a primeira sugestão de uma marcação específica para estilização e apresentação, no mesmo tempo que também propõe a criação do atributo `class`. Surgiu, ainda, a marcação para tabelas, para notas de rodapé e formulários. A marcação para tabelas gerou grande discussão na época, tendo sido efetivada somente na versão seguinte da HTML.

Em setembro, a Netscape propôs o conceito de *frames* em documentos HTML e implementou essa funcionalidade em seu navegador, sem consultas à comunidade, tornando-a prática comum.

Em novembro, a Microsoft lançou a versão 2.0 do Internet Explorer. Bert Bos, Ericsson Lie, Dave Raggett, Chris Lilley e colaboradores iniciaram, nesse mesmo mês, a idealização das folhas de estilo em cascata (CSS).

Em dezembro, dissolveu-se o grupo de trabalho para a HTML.

Em fevereiro de 1996, o W3C criou o HTML ERB, um grupo formado por representantes da IBM, Microsoft, Netscape, Novell, Softquad e do W3C Consortium, encarregado de rever e padronizar a HTML com a finalidade de acabar com as implementações proprietárias.

Em dezembro, o HTML ERB transformou-se no Grupo de Trabalho da HTML, que existe até hoje. Esse mês assinalou o início dos estudos para a implementação de uma marcação provisoriamente denominada *Cougar*, a qual, mais tarde, transformar-se-ia no HTML4.

Em janeiro de 1997, o W3C endossou a HTML 3.2 como uma Recomendação oficial. Com essa versão, a HTML incorporou os elementos `table` e `applet`, bem como elementos para marcar subscritos, sobrescritos e texto ao redor de imagens.

Em julho de 1997, foi lançada a versão rascunho para a Cougar, depois denominada HTML4.

Em dezembro desse ano, o W3C endossou a HTML4 como uma Recomendação oficial.

Em dezembro de 1999, o W3C publicou as Recomendações para o HTML 4.01. Essa é a versão atual da HTML.

1.3 Criação da HTML5

Em maio de 2007, o W3C reconsiderou sua decisão de encerrar o desenvolvimento da HTML em favor da XHTML e tornou pública sua decisão de retomar os estudos para o desenvolvimento da HTML5, tomando como base o trabalho que já vinha sendo desenvolvido pelo WHATWG.

WHATWG é a sigla em inglês para *Web Hypertext Application Technology Working Group*, que, em português, significa Grupo de Trabalho para Tecnologias de Hipertexto em Aplicações para Web. O WHATWG foi criado em 2004 por desenvolvedores da Apple, da Fundação Mozilla e do navegador Opera, que, descontentes com os rumos adotados pelo W3C, propuseram-se a desenvolver as especificações para HTML5, Web Forms 2.0 e Web Controls 1.0. Atualmente, o foco único do Grupo de Trabalho é a HTML5, uma vez que a Web Forms 2.0 também foi assimilada pelo W3C e os estudos para Web Controls 1.0 foram interrompidos.

O WHATWG desenvolve a HTML5 em conjunto com o W3C e ambos mantêm em seus sites uma versão das especificações que diferem ligeiramente em pequenos detalhes. A versão do WHATWG é menos restritiva do que a versão do W3C. Por exemplo: em vários itens da especificação, apresenta exemplos ilustrativos e informações sobre suporte da funcionalidade descrita, nos navegadores modernos. Essas informações adicionais não constam da versão do W3C.

No dia 19 de janeiro de 2011, Ian Hickson, editor da HTML5, publicou no blog da WHATWG uma matéria informando que a especificação para a HTML5 continuaria a ser desenvolvida exclusivamente pelo W3C, ficando sob responsabilidade do WHATWG a continuidade do desenvolvimento de uma especificação para a HTML geral, isto é, sem sufixo designativo da versão.

- Estrutura da especificação para a HTML5

A especificação para a HTML5 está estruturada em 10 seções, a saber:

- 1. **Infraestrutura comum:** define terminologia, classes, algoritmos, sintaxes e partes comuns das especificações.
- 2. **Semântica, estrutura e APIs para documentos HTML:** definem as funcionalidades do DOM HTML e dos elementos HTML em geral.
- 3. **Elementos HTML:** explicam o significado de cada um dos elementos HTML. São estabelecidas regras de uso dos elementos na marcação, bem como diretrizes de manipulação deles pelos agentes de usuário.
- 4. **Microformatos:** a especificação para a HTML5 prevê um mecanismo para marcar informações sobre o documento, no formato nome/valor, para serem lidas por máquinas. Essa seção descreve esse mecanismo e os algoritmos capazes de converter documentos HTML em outros formatos. Adicionalmente, nessa seção definem-se alguns vocabulários para Microformatos: informações de contato, calendário de eventos e licenças.
- 5. **Carregamento de páginas web:** documentos HTML não aparecem do nada – essa seção define as muitas funcionalidades relacionadas ao tratamento de páginas web pelos diferentes dispositivos.
- 6. **APIs para aplicações web:** descrevem as funcionalidades básicas para desenvolvimento de scripts em aplicações HTML.
- 7. **Interação com o usuário:** descreve os diferentes mecanismos de interação do usuário com um documento HTML.
- 8. **APIs para comunicação:** tratam dos mecanismos de comunicação entre aplicações HTML rodando em domínios diferentes e no mesmo cliente.
- 9. **Sintaxe HTML:** descreve a sintaxe HTML.
- 10. **Sintaxe XHTML:** descreve a sintaxe XHTML.

A especificação contém ainda apêndices nos quais são estabelecidas regras de renderização para os navegadores, listadas as funcionalidades obsoletas e considerações da Internet Assigned Numbers Authority (IANA), órgão responsável pela coordenação geral de protocolos para a Internet, nomes de domínio (DNS) e endereço IP.

1.5 Princípios de desenvolvimento da HTML5

Em 26 de novembro de 2007, o Grupo de Trabalho da HTML pertencente ao W3C publicou uma nota contendo um conjunto de diretrizes a ser seguido pelo Grupo de Trabalho e que descreve os princípios de desenvolvimento da HTML5. Tal nota está hospedada no site do W3C: <http://www.w3.org/TR/html-design-principles/>.

Os princípios descritos visam a orientar o Grupo de Trabalho que desenvolve a HTML5 nas seguintes áreas:

- ▣ Compatibilidade
- ▣ Utilidade
- ▣ Interoperabilidade
- ▣ Acesso universal

Vejamos a seguir cada uma dessas áreas e seus princípios de desenvolvimento para a HTML5.

1.5.1 Compatibilidade

Compatibilidade é um termo que pode ser interpretado de várias maneiras. É frequente o uso dos termos “retrocompatibilidade” e “compatibilidade futura”, contudo, dependendo do contexto, nem sempre fica claro o que eles significam. Os princípios de compatibilidade previstos na Nota do W3C esclarecem o termo compatibilidade em diferentes contextos, conforme descritos a seguir.

1.5.1.1 Suporte para conteúdos existentes

Existem milhares de páginas web codificadas em desacordo com as diretrizes da HTML e que são renderizadas a contento e funcionam perfeitamente, pois o tratamento dado pelos navegadores aos erros de marcação é muito complacente.

Historicamente os fabricantes de navegadores cada vez mais implementaram mecanismos de tratamento de erros de marcação capazes de simplesmente ignorá-los. O nível de sofisticação dos mecanismos chegou a tal ponto que os navegadores praticamente “adivinhavam” o que o desenvolvedor deveria ter feito quando cometeu um erro de marcação e, na maioria dos casos, renderizam o conteúdo como se erros não existissem.

Entre a marcação de uma lista não ordenada com um nível de aninhamento:

```
<ul>
  <li>item 1</li>
    <ul>
      <li>item 1.1
        <li>item 1.2</li>
      </ul>
    </li>
  <li>item 2
    <ul>
      <li>item 2.1
        <li>item 2.2</li>
    </ul>
  </li>
```

A marcação em questão contém vários erros, mas, se constar de um documento enviado com o tipo de MIME `text/html`, e qualquer DOCTYPE ou mesmo sem DTD renderizada normalmente como uma lista de dois itens, cada um deles com subitens e respectivos marcadores-padrão.

Essa marcação XHTML servida com o tipo de MIME `application/xhtml+xml` o navegador acusará um erro fatal e o usuário será brindado com uma mensagem de erro.

Tudo sabemos, a tentativa de evoluir a HTML para XHTML fracassou e foi abandonada pelo W3C em favor da HTML5 e esta ao contrário da XHTML, segundo princípio da compatibilidade, deverá ser desenvolvida de modo a não “quebrar” marcação errada.

Muitos sites usam o elemento `u` para marcar textos sublinhados. Esse elemento foi declarado em desuso pela HTML4; contudo, a HTML5 deve prever suporte para ele.

Esses dois simples exemplos bem ilustram o princípio da compatibilidade, porém não é só marcação errada e elemento em desuso que ele contempla. APIs de especificações HTML antigas, funcionalidades e comportamentos não previstos em especificações e até mesmo funcionalidades proprietárias devem ser consideradas no desenvolvimento da HTML5.

Convém ainda ressaltar que renderizar corretamente marcação errada não implica documento válido.

Segundo esse princípio, a HTML5 está sendo desenvolvida de modo que a marcação mostrada anteriormente renderiza normalmente em HTML5, mas os autores devem evitá-la ao criar documentos novos. Elas serão suportadas em documentos antigos quando migrados para HTML5.

As perguntas que devem ser feitas para se chegar à conclusão sobre o suporte ou não a uma funcionalidade ou comportamento são:

- A quantidade de conteúdos dependente da funcionalidade ou comportamento é considerável?
- A funcionalidade ou comportamento ocorre em sites populares e de grande audiência?
- A funcionalidade ou comportamento é para consumo global ou localizado?
- A funcionalidade ou comportamento é para a web pública ou restrito a dispositivos particulares?
- A funcionalidade ou comportamento funciona em vários dispositivos ou está restrito a determinado dispositivo?

1.5.1.2 Degradação graciosa

Degradação graciosa é a tradução literal para “graceful degradation” cujo significado é o seguinte:

A criação de um conteúdo deve ser projetada de modo que usuários e agentes de usuários tenham acesso a ele independentemente do dispositivo e condições que estejam usando, ainda que os mais precários possíveis.

O conteúdo é criado implementando-se as mais modernas e avançadas tecnologias com o objetivo de estilizar, incrementar apresentação e usabilidade e tudo mais que achar conveniente.

Nessas condições, o princípio da degradação graciosa estabelece que dispositivos e usuários sem suporte para as tecnologias avançadas não ficarão privados da informação transmitida pelo conteúdo. Por exemplo: projetar um box com bordas arredondadas e sombras proporciona degradação graciosa para usuários com navegador sem suporte para CSS3 que verão o box com bordas não arredondadas e sem sombras, porém terão acesso ao conteúdo nele inserido. Ao projetar um menu dropdown com JavaScript, é preciso garantir que o menu será acessível e funcional para usuários sem suporte ou com JavaScript desabilitado.

Esses dois simples exemplos bem ilustram o princípio da degradação graciosa que, em última análise, significa o seguinte:

Ao acrescentar funcionalidades cujo suporte é previsto nos mais modernos navegadores, é preciso que os desenvolvedores garantam mecanismos capazes

se renderizar o conteúdo de forma a não perder a informação transmitida, quando o conteúdo é renderizado em agentes de usuário antigos nem para usuários que navegam com algumas funcionalidades desabilitadas.

Quando esse princípio, a HTML5 está sendo desenvolvida de modo a fornecer mecanismos capazes de proporcionar uma solução alternativa para a degradação do conteúdo. Em se tratando de novas funcionalidades, na maioria dos casos, as seguintes considerações são levadas em conta:

- Novos elementos ou atributos, quando não entendidos pelo agente de usuário, devem prever um mecanismo adicional de transmissão de semântica, sem perder sua funcionalidade.
- Novos métodos ou atributos devem ser capazes de ser testados via ECMAScript antes de serem usados.
- Novos elementos ou atributos devem ser semânticos e capazes de receberem estilização mínima.
- Novos elementos capazes de comportarem um mecanismo de renderização sofisticado devem prever uma alternativa de transmissão do seu conteúdo para agentes que não os suportem.

Contudo, esse princípio não é absoluto no desenvolvimento da HTML5. Em alguns casos, a nova funcionalidade simplesmente não irá funcionar em determinada classe de agentes de usuário ou ainda é impossível de se prover degradação graciosa. Por exemplo: as novas APIs que dependem de linguagem de script para funcionar só funcionarão em agentes de usuário sem suporte para o script.

Os exemplos a seguir ilustram o princípio da degradação graciosa aplicado ao desenvolvimento da HTML5:

- Os novos elementos `canvas` e `audio` destinados a implementar multimídia em um documento preveem um mecanismo de apresentação de conteúdo alternativo para navegadores que não suportam esses elementos, como mostrado na marcação a seguir:

```
<canvas>Insira aqui o conteúdo alternativo</canvas>
<audio>Insira aqui o conteúdo alternativo</audio>
```

- O novo método `getElementsByClassName()` é muito mais rápido do que seu equivalente tradicional desenvolvido com ECMAScript. Segundo o princípio da degradação graciosa, mecanismos de verificação de suporte para esse método podem ser usados para servir o método equivalente para agentes que não o suportam.

1.5.1.3 Não reinventar a roda

Segundo esse princípio, se uma tecnologia já está implementada e é largamente usada para atender a determinados casos, não há necessidade de se criar uma nova funcionalidade para atender àqueles casos. Mesmo porque é mais fácil ampliar funcionalidades já existentes e testadas do que partir do zero criando nova funcionalidade com o mesmo propósito.

O exemplo clássico da aplicação desse princípio é a adoção dos elementos `marquee` e do atributo `contenteditable` que já existem, não constam das especificações para a HTML, mas agora passam a fazer parte da HTML5.

1.5.1.4 Pavimentar os caminhos existentes

Segundo esse princípio, se uma prática é amplamente usada pelos desenvolvedores, é preferível adotá-la a proibi-la ou inventar algo novo para substituí-la.

É comum a prática de os autores usarem, em documentos HTML, a marcação `
` em detrimento de `
`. Não há nenhum dano em se permitir a continuidade dessa prática.

1.5.1.5 Evolução em lugar de revolução

Revoluções, por vezes, mudam o mundo para melhor. Contudo, na maioria dos casos, é melhor aperfeiçoar práticas existentes do que simplesmente descartá-las. Segundo esse princípio, se uma técnica é usada pelos desenvolvedores, é preferível aperfeiçoá-la e adotá-la em vez de obrigar os autores a estudar e aprender novas técnicas.

1.5.2 Utilidade

Utilidade visa a garantir que a marcação HTML possa ser usada de modo efetivo para todos os fins a que ela se destina.

1.5.2.1 Solucionar problemas reais

Alterações nas especificações devem ser feitas para solucionar problemas atuais reais. Abstrações e ilações teóricas sobre problemas existentes devem ser descartadas em favor de uma solução pragmática e efetiva. Todos os problemas atuais devem ser estudados e resolvidos, se possível.

1.5.2.2 Prioridades

Prioridades é um princípio que considera uma cadeia de interessados quando há conflitos em relação à implementação de uma funcionalidade na HTML5.

Nesses casos, ao se criar uma funcionalidade, os usuários serão considerados em primeiro lugar. Seguem-se a estes os implementadores das funcionalidades que são seguidos pelos que criam especificações; em último lugar, considera-se a pureza teórica da questão.

1.5.2.3 Segurança

Segurança é um princípio que visa a assegurar que as funcionalidades da HTML5 não ofereçam brechas que possibilitem violar as normas de segurança do modelo web. Preferencialmente as considerações sobre segurança devem ser tratadas diretamente nas especificações.

Por exemplo: a comunicação entre documentos de diferentes sites é muito útil; contudo, implementar tal funcionalidade sem restrições pode colocar em risco a segurança dos dados transmitidos. A funcionalidade para comunicação entre documentos está projetada na HTML5 paralelamente a mecanismos que impedem a violação das regras de segurança.

1.5.2.4 Separação de camadas

A HTML foi projetada para separar a marcação estrutural da apresentação. Nesse contexto, marcação que expresse estrutura é preferida sobre marcação para apresentação. Contudo, marcação estruturada é um meio para se alcançar um fim tal como ocorre quando consideramos o funcionamento da funcionalidade independentemente do tipo de mídia do usuário.

Esse princípio estabelece que não são necessárias considerações profundas e detalhamento refinado sobre questões relacionadas à semântica se o fim pode ser alcançado por uma via mais simples. Definir um padrão razoável de semântica para diferentes tipos de mídia é suficiente. Deve ser feito um balanço ponderado entre semântica refinada e uso prático antes de se chegar a uma conclusão. Nomes de elementos e atributos da marcação devem ser práticos e pragmáticos em lugar de verbosos e muito detalhados.

Por exemplo: o elemento `article` define um artigo individual sem detalhar a maneira como ele deve ser renderizado. Em determinado contexto, um artigo pode ser único na página e disposto em múltiplas colunas ao passo que em outro contexto possa coexistir com vários outros artigos multicolunares em uma mesma página.

Os elementos `b` e `i` são largamente usados. É muito melhor estabelecer novas regras de renderização para eles, nos diferentes tipos de mídia, do que simplesmente bani-los das especificações.

1.5.2.5 Consistência do DOM

O princípio da consistência do DOM, como o próprio nome sugere, visa a criar funcionalidades na HTML5 que permitam a produção de uma árvore do documento consistente e facilmente acessível às linguagens de script e programas em geral. Discrepâncias são admitidas desde que necessárias a contemplar dispositivos do passado, mas, nesses casos, as diferenças devem ser minimizadas.

1.5.3 Interoperabilidade

Interoperabilidade visa a aumentar as chances de uma implementação HTML funcionar nos mais variados dispositivos e sistemas.

1.5.3.1 Comportamentos definidos

Esse princípio visa a que as funcionalidades, quando for o caso, comportem-se de maneira bem definida em lugar de deixar livremente por conta dos agentes de usuário a tarefa de estabelecer como será o comportamento. Contudo, deve-se permitir que as implementações tenham certa liberdade para incrementar algum comportamento em áreas como de interface do usuário e da qualidade de renderização.

1.5.3.2 Evitar soluções complexas

Esse princípio dá preferência ao uso de soluções simples. Funcionalidades simples são mais fáceis de serem implementadas pelos agentes de usuário, além de facilitarem a interoperabilidade e o entendimento pelos autores. No entanto, simplicidade não deve ser desculpa para se ferir outros princípios de desenvolvimento da HTML5.

1.5.3.3 Tratamento de erros

Esse princípio estabelece que os erros de marcação devem ser tratados de modo a não frustrar o usuário apresentando-lhe uma mensagem de erro. Nos casos de erro é preferível um mecanismo de degradação graciosa a uma mensagem de erro fatal.

1.5.4 Acesso universal

Acesso universal visa a garantir o acesso às funcionalidades da HTML5 pelo maior número possível de dispositivos e sistemas segundo os princípios a seguir:

1.5.4.1 Independência de mídia

As funcionalidades da HTML5 devem, sempre que possível, funcionar nos mais variados tipos de plataformas, dispositivos e mídias. Isso não significa que determinada funcionalidade deva ser descartada porque não é suportada por essa ou aquela plataforma. Por exemplo: funcionalidades de interação não devem ser descartadas pelo fato de não serem suportadas em mídia impressa, como ocorre com o elemento `a` que simplesmente não funciona quando impresso.

1.5.4.2 Suporte multilíngua

Essa funcionalidade visa a facilitar a produção de documentos nos diferentes idiomas existentes no mundo.

1.5.4.3 Acessibilidade

Essa funcionalidade visa a facilitar o acesso ao conteúdo independentemente do dispositivo ou das necessidades especiais do usuário. Isso não significa que uma funcionalidade deva ser descartada se um grupo de usuários não tiver acesso a ela, mas se devem prever meios alternativos de acesso. É o caso do elemento `img` que não deve ser banido porque usuários cegos não conseguem ver imagens. Usa-se o atributo `alt` para fornecer um meio alternativo de acesso à imagem para aqueles usuários.

1.6 Diferenças entre HTML5 e HTML4

As principais diferenças entre a HTML5 e a HTML4 têm suas origens na tentativa de a HTML5 estar sendo desenvolvida com o propósito de substituir tanto a HTML4 criada nos anos 90 quanto a XHTML que foi uma tentativa frustrada de reformular a HTML4 como uma aplicação XML.

1.6.1 Introdução

A especificação para a HTML5 iniciada no ano de 2004 e incorporada pelo W3C no ano de 2007 tem como objetivo geral estudar e resolver problemas relacionados à implementação de um HTML contemporâneo e ao mesmo tempo compatível com conteúdos existentes. Para cumprir esse objetivo, considera os seguintes pontos:

- Definição de uma linguagem única denominada HTML5 que pode ser escrita tanto com a sintaxe HTML quanto com a sintaxe XML.
- Definição de modelos de processamento detalhados com vista a promover a interoperabilidade da linguagem.
- Aperfeiçoamento da marcação dos documentos.
- Criação de marcação e APIs para novas tecnologias, tais como as aplicações Web.

Nesse contexto, podemos agrupar as diferenças entre as duas linguagens nas seguintes áreas:

1.6.1.1 Retrocompatibilidade

A HTML5 está sendo desenvolvida em dois níveis de requisitos de conformidades. Requisitos para autores e requisitos para agentes de usuário. Considere, para ilustrar a diferença nessa área, os elementos `isindex` e `plaintext`. De acordo com a HTML4, esses elementos estão em desuso (deprecated) e seriam banidos das próximas especificações para a HTML. Segundo os princípios de desenvolvimento da HTML5, esses elementos perdem sua condição de “em desuso” e passam a ser considerados elementos obsoletos (obsolete). A diferença é que eles não serão banidos da especificação. Continuarão constando como requisitos para agentes de usuário e seu uso pelos autores é desaconselhado.

Na prática isso significa que os fabricantes de agentes de usuário devem implementar funcionalidades que permitam reconhecer e renderizar apropriadamente aqueles elementos nos dispositivos futuros, muito embora o uso deles, pelos autores, seja desaconselhado.

Na HTML5 não existem elementos em desuso (deprecated) com o sentido que a palavra tem na HTML4.

O mesmo acontece com outros elementos em desuso segundo a HTML4, como os elementos `font`, `u`, `s` etc.

1.6.1.2 Modelo de desenvolvimento

Nessa área a diferença é que a HTML5 não será considerada concluída até que haja pelo menos duas implementações completas das funcionalidades da especificação. Será criada uma suíte de testes para verificar o funcionamento de todas as funcionalidades nas duas implementações. O objetivo aqui é assegurar que a especificação pode ser implementada e usada pelos autores tão logo esteja finalizada.

Para as versões anteriores da linguagem HTML, a especificação final era aprovada por um comitê do W3C antes mesmo de ser totalmente implementada.

1.6.2 Sintaxe

A especificação para a HTML5 define uma sintaxe que é compatível tanto com a HTML quanto com a XHTML. Os documentos usando a sintaxe HTML com o tipo de MIME `text/html` são parseados segundo regras compatíveis com as atuais implementações populares.

Observe a seguir um exemplo de marcação HTML:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html lang="pt-br">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>Título da página</title>
    <link rel="stylesheet" type="text/css" href="/estilos/main.css">
  </head>
  <body>
    <h1>Minha página HTML4</h1>
  </body>
</html>
```

A seguir, a marcação HTML5 compatível com a sintaxe HTML:

```
<!DOCTYPE html>
<html lang="pt-br">
    <head>
        <meta charset="utf-8">
        <title> Título da página </title>
        <link rel="stylesheet" href="/estilos/main.css">
    </head>
    <body>
        <h1>Minha página HTML5</h1>
    </body>
</html>
```

1.6.2.1 DOCTYPE

A declaração DOCTYPE (tipo de documento) deve ser feita na primeira linha da marcação HTML para garantir renderização no modo standard. Para maiores informações, ver: <http://maujor.com/w3ctuto/qatips/doctype.html>. Nada deverá existir acima da declaração de DOCTYPE nem mesmo uma linha em branco.

Existem doze diferentes DOCTYPES para uso em documentos HTML e XHTML e todos eles são verbosos e difíceis de se guardar na memória. Todos eles fazem referência e contêm um link apontando para um documento com as regras de sintaxe para a versão HTML ou XHTML na qual o documento foi escrito.

Em HTML5 o DOCTYPE foi simplificado, não havendo mais necessidade do link, e o DOCTYPE existe com a única função de habilitar a renderização em modo standard para documentos escritos com a sintaxe HTML. Os navegadores já entendem a declaração DOCTYPE segundo a sintaxe HTML5.

1.6.2.2 Codificação de caracteres

A codificação de caracteres em um documento web pode ser feita de três maneiras distintas:

- Com uso do parâmetro charset no cabeçalho HTTP Content-Type.
- Uso de um caractere Unicode Byte Order Mark (BOM) no início do documento.
- Uso do elemento meta com o atributo charset.

Em HTML5 a declaração da codificação de caracteres com uso do elemento `meta` foi simplificada, pois não há mais a necessidade dos atributos `http-equiv` e `content`. Contudo, convém notar que a sintaxe HTML4 para esse elemento também é válida em HTML5 e, se você a preferir, pode usá-la sem problemas. Para maiores informações, ver: <http://www.w3.org/International/0-charset>.

1.6.2.3 Elemento `link`

Em HTML5 o atributo `type` para links a folhas de estilos e para scripts é dispensável, mas a declaração desse atributo pode ser usada, opcionalmente, em HTML5.

1.7 A HTML5 nos navegadores atuais

Desenvolver um projeto web em HTML5 para os navegadores atuais é uma opção que pode ser considerada; caso você decida por ela, é preciso levar em conta algumas restrições e mecanismos existentes para contorná-las.

A maioria, mas não todas as funcionalidades da HTML5, já é suportada por um ou mais navegadores atuais; contudo, existem mecanismos desenvolvidos com a linguagem JavaScript que permitem detectar suporte para as funcionalidades criando condições de o desenvolvedor oferecer um conteúdo alternativo para uma funcionalidade não suportada por esse ou aquele navegador. Vejamos a seguir algumas considerações sobre esse tema.

1.7.1 Os novos elementos e o Internet Explorer

A HTML5 criou vários elementos novos que não são estilizáveis com uso das CSS nos navegadores Internet Explorer nas versões anteriores à versão 9, pois esses navegadores, ao contrário dos navegadores standards, não aplicam regras CSS a elementos que não conhecem.

Em uma marcação HTML, se você resolver criar um novo elemento chamado `xpto` (ainda que ele não seja válido) e escrever uma regra CSS como mostrada a seguir, o conteúdo inserido no elemento será normalmente estilizado na cor vermelha com fundo na cor preta, exceto nos IE8 e anteriores.

```
xpto { color: red; background-color: black; }
```

Se você pretende servir seu documento HTML5 para aqueles navegadores, precisa inserir um mecanismo capaz de fazer com que eles reconheçam os novos elementos

da linguagem. Tal mecanismo é oferecido pelo método `createElement()` da JavaScript, conforme mostrado a seguir:

```
<!--[if lte IE 8]>
<script>
    document.createElement("xpto");
</script>
<![endif]-->
```

Existem aproximadamente trinta novos elementos na HTML5 e uma alternativa para o script, criando todos esses elementos individualmente da forma como mostrada anteriormente; foi desenvolvida por Remy Sharp que apresentou o script em uma matéria no seu blog, disponível para consulta em <http://remysharp.com/2009/01/07/html5-enabling-script/>.

O script de Remy Sharp encontra-se hospedado no Google em endereço público; podemos criar em nossos documentos um *hotlink* para o script, inserindo-o na seção head do documento, como mostrado a seguir:

```
<!--[if lte IE 8]>
<script src="http://html5shim.googlecode.com/svn/trunk/html5.js"></script>
<![endif]-->
```

1.7.2 Os novos elementos e o modelo de renderização

Outro aspecto importante a considerar, e esse válido para todos os navegadores, é que os novos elementos, todos eles, renderizam seus conteúdos como elementos inline, pois os modelos de conteúdo da HTML5 são prática e conceitualmente diferentes dos modelos das HTML anteriores.

Se quisermos que um ou mais dos novos elementos sejam renderizados segundo o conceito nível de bloco da HTML4, devemos declarar explicitamente essa condição em regra CSS, conforme mostrado a seguir:

```
xpto { display: block; }
```

1.7.3 Biblioteca JavaScript Modernizr

Modernizr é uma pequena biblioteca JavaScript que está sendo constantemente atualizada por seus autores, destinada a detectar suporte nativo pelos navegadores para as funcionalidades das novas tecnologias para a web. As funcionalidades contempladas pela biblioteca são aquelas que estão sendo implementadas pelas especificações para a HTML5 e para as CSS3.

A biblioteca foi desenvolvida por Faruk Ateş com a colaboração de Paul Irish e encontra-se disponível para download em www.modernizr.com.

O script da biblioteca cria um objeto JavaScript denominado Modernizr e várias propriedades para esse objeto. As propriedades e métodos criados têm nomes idênticos aos das novas funcionalidades a detectar. Observe os exemplos a seguir:

```
Modernizr.canvas;           // propriedade para verificar suporte ao novo elemento canvas  
Modernizr.audio;            // propriedade para verificar suporte ao novo elemento audio  
Modernizr.borderradius;     // propriedade para verificar suporte a bordas arredondadas das CSS3  
Modernizr.multiplebgs;      // propriedade para verificar suporte a múltiplos backgrounds das CSS3
```

Uma vez linkada a biblioteca a um documento web, podemos testar o suporte a uma nova funcionalidade, conforme exemplo a seguir, em que testamos o suporte para o elemento `canvas`.

```
if (Modernizr.canvas) {  
    // script para uso de canvas  
} else {  
    alert("Lamento, seu navegador não suporta canvas");  
}
```

A biblioteca cria e adiciona classes ao elemento `html`, oferecendo ao desenvolvedor a possibilidade de estilizar diferenciadamente um ou mais elementos na página conforme o navegador ofereça ou não suporte a determinada funcionalidade.

Suponha o seguinte cenário para exemplificar o uso dessas classes:

Se o navegador oferecer suporte para múltiplos backgrounds, os parágrafos do documento serão na cor vermelha; se não suportar, serão na cor azul.

A biblioteca identifica o suporte pelo navegador e acrescenta a classe com o nome `multiplebgs` ao elemento `html`. Não havendo suporte, o nome da classe acrescentada é `no-multiplebgs` e a folha de estilos para resolver o problema proposto no cenário criado é:

```
<style>  
    .multiplebgs p { color: red; }  
    .no-multiplebgs p { color: blue; }  
</style>
```

Notar que o acréscimo da classe, como mostrado, possibilita uma autêntica condicional na folha de estilos, pois as regras mostradas cumprem a mesma finalidade da condicional mostrada no script a seguir.

```
<script>
    var para = document.getElementsByTagName("p");
    if (Modernizr.multiplebgs) {
        for (var i=0; i<para.length; i++) {
            para[i].style.color = "red";
        }
    } else {
        for (var i=0; i<para.length; i++) {
            para[i].style.color = "blue";
        }
    }
</script>
```

Outra funcionalidade importante da biblioteca é que ela agregou o script desenvolvido por Remy Sharp, mostrado no item 1.7.1, que faz com que o navegador Internet Explorer, nas versões 8 e anteriores, reconheça, para fins de aplicação de estilização, os novos elementos da HTML5. Assim, o uso da biblioteca dispensa o uso do script de Remy Sharp.

Atualmente não existe um endereço público de hospedagem da biblioteca para incorporá-la aos nossos documentos com uso de um *hotlink*. Você deverá fazer o download da biblioteca, hospedá-la no seu servidor e linká-la às suas páginas, conforme mostrado a seguir:

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    ...
    <script src="path/modernizr-v.n.min.js"></script>
</head>
<body>
    ...

```

Maiores detalhes sobre a biblioteca, incluindo a relação de todos os objetos, propriedades e classes por ela criadas encontram-se em <http://www.modernizr.com/docs/>.

1.8 Templates HTML5

A sintaxe HTML5 é bastante flexível, pois a marcação pode ser escrita segundo as regras da HTML4 que admitem, por exemplo, que valores de atributo sejam ou não marcados com aspas ou apóstrofes, que algumas tags, como `p` e `li`, não sejam fechadas e uma série de outras regras que dão liberdade de escolha da marcação conforme

preferência pessoal do autor. Escrever HTML5 com base nessas regras significa produzir um documento HTML5, ou em conformidade com a sintaxe HTML.

Por outro lado, a HTML5 pode também seguir a sintaxe XML e ser escrita usando suas regras que são bem rígidas, pois não admitem, por exemplo, valores de atributos sem aspas ou apóstrofes e tags sem fechamento. Se você desenvolve HTML, deve estar bem familiarizado com essa sintaxe. Escrever HTML5 com base nas regras da XML significa produzir um documento XHTML5, ou em conformidade com a sintaxe XML.

Nessa seção apresentaremos as formas de escrever HTML5 segundo as diversas sintaxes permitidas pela linguagem, desenvolvendo alguns templates ilustrativos.

Antes de passarmos aos templates, façamos algumas considerações sobre a marcação XHTML.

Seja você um desenvolvedor experiente, seja um iniciante que cria conteúdos com uso de marcação HTML, é quase certo que, assim como eu, desenvolve seus documentos segundo as regras da XHTML.

Existem milhões de páginas na Internet desenvolvidas com essa marcação; isso porque quem as criou estava se preparando para o futuro, pois até o início de 2007 o W3C alardeava que o futuro da HTML era a XHTML e que não haveria novas versões da HTML.

O objetivo do W3C ao criar a XHTML foi proporcionar meios aos desenvolvedores de incluir as funcionalidades da XML nos documentos para a web. Em março de 2007, Tim Berners-Lee publicou um press release afirmando o seguinte:

Após a publicação do HTML4 e seguindo as conclusões do Workshop 1998, o W3C focou na transformação do HTML em um formato com base em XML chamado XHTML devido aos benefícios próprios do formato XML. A primeira recomendação XHTML completa foi publicada no início de 2000. Contudo, devido ao maciço legado do conteúdo Web, com base em variações do HTML, os fabricantes de navegadores começaram vagarosamente a adotar o XHTML. Isso resultou em pouca motivação para os desenvolvedores de conteúdos adotar o XHTML em seu ambiente tradicional de desenvolvimento. Exponentes das comunidades de desenvolvedores e de design apelaram ao W3C para a necessidade urgente de uma revisão dos seus compromissos com o HTML, acrescentando novas funcionalidades (a começar com a Norma para HTML4) em uma maneira que seja consistente com as práticas da comunidade e ao mesmo tempo retrocompatíveis. O W3C garantirá interoperabilidade por meio da implementação de suítes de testes robustas e de serviços de validação para as futuras tecnologias e que estarão à disposição da comunidade.

O W3C com o forte suporte dos seus Membros e com mais recursos (tanto em pessoal quanto em hardware) tem o prazer de retomar os trabalhos para o HTML. O W3C modificou os Estatutos do Grupo de Trabalho do HTML com o propósito de permitir a efetiva participação dos fabricantes de navegadores, projetistas de aplicações e desenvolvedores de conteúdos, pois depende do trabalho conjunto dessas pessoas o sucesso do HTML do futuro.

Estava decretada a morte da XHTML em favor da HTML5. Contudo, o legado da XHTML permanece e você que já se acostumou com a sintaxe rígida da marcação não terá de rever o que aprendeu, pois ela continua perfeitamente válida na HTML5, assim como é válida também a sintaxe HTML4 muito menos rígida e mais complacente do que a da XHTML.

Um documento XHTML para ser puro e poder se beneficiar das funcionalidades da XML – e foi para isso que a XHTML foi criada – deve ser servido com o tipo de MIME `application/xml` ou `application/xhtml+xml`. Contudo, a maioria dos documentos XHTML na web é servida com o tipo MIME para HTML `text/html`, o que significa que não são conformes com a XML.

A tentativa de o W3C evoluir a HTML para conformidade com XML fracassou, porque páginas XHTML servidas como XML que contenham qualquer violação com a sintaxe XML causarão um erro fatal de parseamento e a página simplesmente não será renderizada.

Por outro lado, apesar da rigidez com o tratamento de erros, a sintaxe XHTML foi bem recebida pelos desenvolvedores e é amplamente usada.

Observe a marcação XHTML a seguir:

```
<?php header('Content-Type: application/xhtml+xml'); ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <title>Página XHTML</title>
  <style type="text/css" media="all">
    /* regras de estilo */
  </style>
</head>
<body>
  <h1>Página com marcação XHTML</h1>
  <p>Essa página contém marcação XHTML <strong class="destaque">sem erros</strong></p>
```

```
<p>Foi servida com tipo de MIME <code>application/xhtml+xml</code></p>
</body>
</html>
```

Para demonstrar o exemplo que desenvolvemos, inserimos na primeira linha da página contendo a marcação XHTML a função `header()` do PHP, com a finalidade de servir a página como aplicação XML. O documento está de acordo com a sintaxe XML e será renderizado normalmente em navegadores que suportam XML.

Vamos supor que o desenvolvedor do exemplo tenha se esquecido de usar aspas no valor destaque da classe para o elemento `strong` inserido no primeiro parágrafo, como mostrado a seguir.

```
<p>Essa página contém marcação XHTML <strong class=destaque>sem erros</strong></p>
```

Se o documento for servido com o tipo de MIME `text/html`, o erro de marcação será ignorado pelo navegador e a renderização do documento será normal. Contudo, servindo o documento como XML com o tipo de MIME `application/xml` ou `application/xhtml+xml` o erro será fatal e não haverá renderização, conforme figuras 1.1 e 1.2.

Na figura 1.1 mostramos a não renderização da página nos navegadores Firefox e Safari.

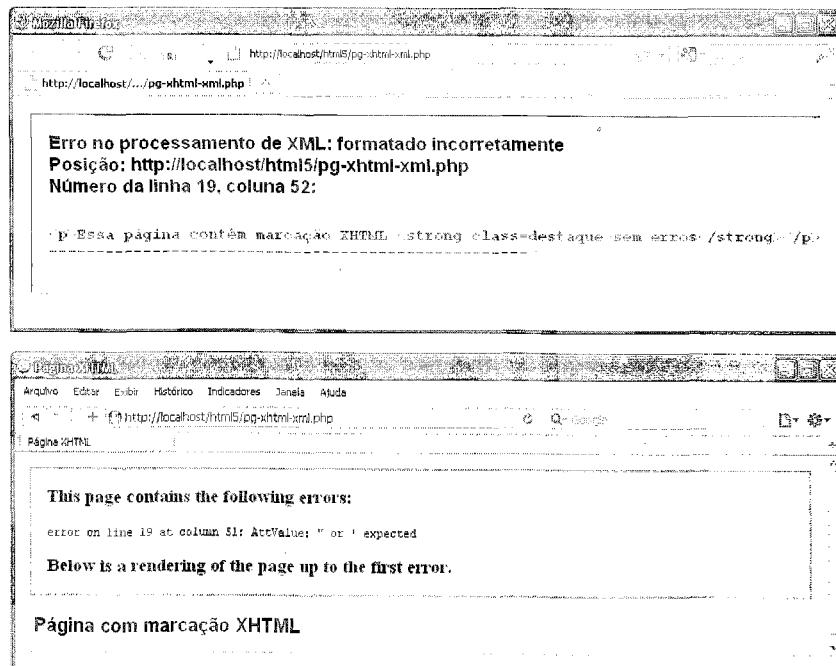


Figura 1.1 – Erro em documento XHTML em navegador standard.

Notar que não há um padrão de apresentação da mensagem de erro, a maneira de apresentá-lo ao usuário difere de um navegador para outro. O navegador Opera, ao apresentar a mensagem de erro, oferece adicionalmente um link para a renderização da página como se o erro não existisse. Por outro lado, o navegador Internet Explorer, que não oferece suporte para XML, comporta-se apresentando ao usuário uma caixa de diálogo com opção de abrir o arquivo com outro programa ou salvá-lo (Figura 1.2).

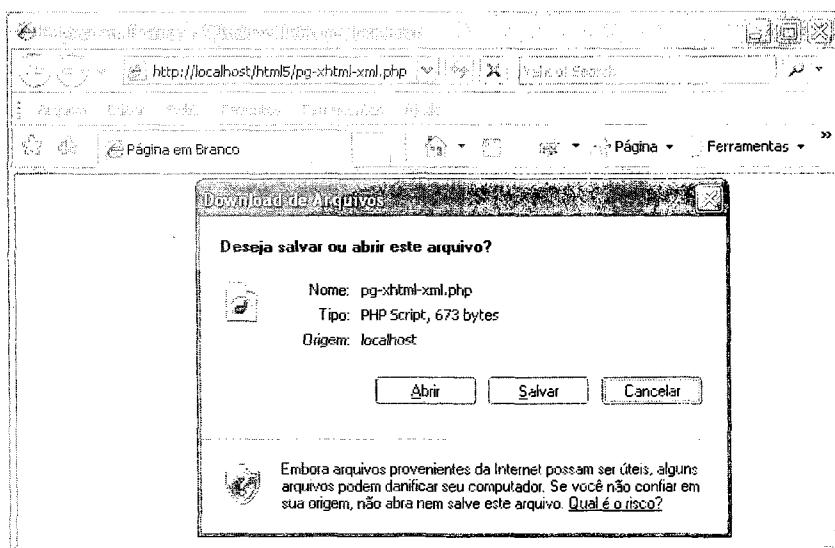


Figura 1.2 – Erro em documento XHTML no IE.

Essa rigidez no tratamento de erros contribuiu muito para a não adoção da XHTML como XML pelos desenvolvedores. Estima-se que mais de 95% das páginas web contenham pelo menos um erro de marcação. Quem se arriscaria a ter seu site “quebrado” por conta de uma distração ou um mínimo engano involuntário?

1.8.1 Template mínimo

Em HTML5, um documento para ser válido, além do DOCTYPE, deve conter o elemento title e nada mais. Assim, um template mínimo da HTML5 pode ser estruturado da seguinte forma:

```
<!DOCTYPE html>
<title>Template</title>
<!-- Aqui os conteudos da pagina -->
```

A marcação mostrada anteriormente valida como HTML5, contudo a marcação a seguir não valida.

```
<!DOCTYPE html>
<title>Template</title>
<!-- Aqui os conteúdos da página -->
```

Notar que na primeira marcação o comentário HTML inserido na última linha do código mostrado contém um texto não acentuado e na segunda marcação o texto foi acentuado.

O uso de caracteres não ASCII (tais como acentuação) na marcação HTML5, mesmo em comentários, obriga que se declare explicitamente a codificação de caracteres. Se isso não for feito, a renderização dos caracteres não ASCII será truncada e o documento não passará no validador. Assim, chegamos ao template mínimo, conforme mostrado a seguir:

```
<!DOCTYPE html>
<title>Template</title>
<meta charset=utf-8>
<!-- Aqui os conteúdos da página -->
```

É evidente que esse template mínimo não deve ser usado em um site real, mas você poderá usá-lo para testes e experiências com a linguagem HTML5.

A sintaxe para DOCTYPE é case insensitive (não distingue minúsculas de maiúsculas), o que significa que qualquer uma das sintaxes seguintes é perfeitamente válida.

```
<!DOCTYPE html>
<!doctype html>
<!DOCTYPE HTML>
<!doctype HTML>
<!Doctype HTML>
```

Apesar da flexibilidade com a sintaxe, aconselho a usar a primeira ou a segunda forma mostrada, que são aquelas preferidas pela maioria dos desenvolvedores. Escolha entre as duas a que mais lhe agrada e siga com ela.

É muito provável que você esteja familiarizado com uma sintaxe HTML para declaração de caracteres no documento com o uso do elemento `meta`, conforme mostrado a seguir.

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

Na sintaxe HTML5, essa declaração foi abreviada para:

```
<meta charset=utf-8>
```

Pelo princípio da compatibilidade, a HTML5 continua a oferecer suporte para a sintaxe da HTML e você, se preferir, pode usá-la nos documentos novos ou deixá-la como está nos documentos existentes migrados para a HTML5.

Tal como na HTML, em HTML5 o uso de aspas ou apóstrofes nos valores dos atributos é facultativo; assim, as duas sintaxes mostradas a seguir são válidas.

```
<meta charset=utf-8>  
<meta charset="utf-8">
```

Como já dissemos, a sintaxe HTML5 é compatível tanto com a HTML quanto com a XHTML; portanto, se você está criando um documento XHTML, terá de fechar as tags e usar aspas em nomes de atributo. Deverá escrever como mostrado a seguir:

```
<meta charset="utf-8" />
```

1.8.2 Template HTML5 – versão 1

A seguir, vamos examinar e comentar um template HTML5 com base em sintaxe HTML4.

A seção `head` de um documento HTML5 deve conter obrigatoriamente o elemento `title` e opcionalmente são admitidos os seguintes elementos naquela seção: `base`, `command`, `link`, `meta`, `noscript`, `script`, `style`. Desses sete elementos opcionais apenas o elemento `command` não é previsto nas versões anteriores da linguagem HTML, pois se trata de um novo elemento da HTML5.



Em aplicações que já possuem um título intríseco, por exemplo, um e-mail escrito em HTML que já tem um campo destinado ao assunto, o elemento `title` pode ser dispensado.

Observe a marcação a seguir:

```
<!DOCTYPE html>  
<html lang="pt-br">  
<head>  
  <meta charset="utf-8">  
  <title>Template</title>  
  <meta name="description" content="Template HTML5 do livro do Maujor.">  
  <meta name="keywords" content="lista de palavras-chave">
```

```
<meta name="author" content="Mauricio Samy Silva">
<meta name="generator" content="HTML-Kit 292">
<meta name="robots" content="all">
<link rel="stylesheet" href="mais.css">
<style>
/* estilos incorporados */
</style>
<script src="path/modernizr-1.5.min.js"></script>
<script src="scripts.js"></script>
</head>
<body>
<!-- Aqui os conteúdos da página --&gt;
&lt;/body&gt;
&lt;/html&gt;</pre>
```

O elemento `meta` destina-se a fornecer metadados, ou seja, informações adicionais sobre a página. Informações essas que não podem ser transmitidas com uso dos demais elementos permitidos na seção `head` do documento.

Os atributos previstos nas especificações da HTML5 para o elemento `meta` são: `name`, `http-equiv`, `content` e `charset`, além dos atributos globais. Sempre que o atributo `name` ou `http-equiv` for definido, deve-se definir também o atributo `content`. Nesses casos, o metadado é do tipo par nome/valor no qual o valor do atributo `name` é o nome no par e o valor do atributo `content` é o valor no par.

A quantidade de elementos `meta` a inserir na seção `head` do documento e o tipo de metadado a codificar são funções de cada projeto; no exemplo mostrado, os elementos foram inseridos apenas a título de ilustração, assim aquela seção não pretende ser um guia rígido de metadados.

No exemplo, inserimos na seção `head`, também a título de ilustração, os elementos `link`, `style` e `script`.

Notar que todos os valores de atributos na marcação mostrada estão entre aspas.

1.8.3 Template HTML5 – versão 2

Vamos examinar e comentar a seguir um template HTML5 idêntico ao template mostrado no item anterior, também com base em sintaxe HTML4.

Observe a marcação a seguir:

```
<!DOCTYPE html>
<html lang="pt-br">
```

```

<head>
  <meta charset=utf-8>
  <title>Template</title>
  <meta name=description content="Template HTML5 do livro do Maujor.">
  <meta name=keywords content="lista de palavras-chave">
  <meta name=author content="Mauricio Samy Silva">
  <meta name=generator content="HTML-Kit 292">
  <meta name=robots content=all>
  <link rel=stylesheet href=mais.css>
  <style>
    /* estilos incorporados */
  </style>
  <script src="path/modernizr-1.5.min.js"></script>
  <script src=scripts.js></script>
</head>
<body>
  <!-- Aqui os conteúdos da página -->
</body>
</html>

```

Nessa versão mostramos a sintaxe alternativa que admite valores de atributos sem aspas.

Mas por que conservamos as aspas em alguns valores do atributo `content`? Notar que para os casos em que isso ocorreu o valor do atributo é uma frase ou conjunto de palavras separadas por espaços e, se tirarmos as aspas, o valor do atributo deixa de ser único, o que não é válido. Com aspas, o parser HTML interpreta o valor do atributo como uma só string. Notar que, para o elemento `meta` que fornece informações para os robôs que visitam a página, o valor do atributo `content` está sem aspas, pois tal valor é uma só palavra.

1.8.4 Template XHTML5 – versão 1

Vamos examinar e comentar a seguir um template XHTML5 com base em sintaxe XML e destinado a ser servido com o tipo de MIME `text/html`.

Observe a marcação a seguir:

```

<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="utf-8" />
    <title>Template</title>

```

```
<meta name="description" content="Template HTML5 do livro do Maujor." />
<meta name="keywords" content="lista de palavras-chave" />
<meta name="author" content="Mauricio Samy Silva" />
<meta name="generator" content="HTML-Kit 292" />
<meta name="robots" content="all" />
<link rel="stylesheet" href="mais.css">
<style>
/* estilos incorporados */
</style>
<script src="path/modernizr-1.5.min.js"></script>
<script src="scripts.js"></script>
</head>
<body>
<!-- Aqui os conteúdos da página --&gt;
&lt;/body&gt;
&lt;/html&gt;</pre>
```

Nessa versão o template é compatível com a sintaxe XML, pois valores de atributo estão entre aspas e elementos vazios estão fechados. Convém ressaltar que, embora escrito com a sintaxe XML, o template não está apto a se valer das funcionalidades da XML e deverá ser servido com o tipo de MIME `text/html`.

1.8.5 Template XHTML5 – versão 2

Vamos examinar e comentar um template XHTML5 com base em sintaxe XML, destinado a ser servido com o tipo de MIME `application/xml` ou `application/xhtml+xml`.

Observe a marcação a seguir:

```
<!DOCTYPE html>
<html lang="pt-br">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="pt-br">
<head>
<meta charset="utf-8" />
<title>Template</title>
<meta name="description" content="Template HTML5 do livro do Maujor." />
<meta name="keywords" content="lista de palavras-chave" />
<meta name="author" content="Mauricio Samy Silva" />
<meta name="generator" content="HTML-Kit 292" />
<meta name="robots" content="all" />
<link rel="stylesheet" href="mais.css" />
<style>
/* estilos incorporados */
</style>
```

```

<script src="path/modernizr-1.5.min.js"></script>
<script src="scripts.js"></script>
</head>
<body>
    <!-- Aqui os conteúdos da página -->
</body>
</html>

```

Nessa versão o template é compatível com a sintaxe XML e apto a valer-se das funcionalidades da XML devendo ser servido com o tipo de MIME `application/xml` ou `application/xhtml+xml`.

Em documentos XML, não há necessidade da declaração de DOCTYPE, pois por padrão a renderização acontece em modo standard. Também não há necessidade de declaração da codificação de caracteres, pois por padrão ela é em UTF-8. Contudo, é indispensável que se declare o atributo `xmlns` no elemento-raiz do documento apontando para o arquivo que contém o namespace para XML. Notar ainda que o atributo para definição do principal idioma no qual os conteúdos do documento foram escritos é `xml:lang` e não `lang` como na sintaxe HTML.

1.8.6 Template XHTML5 – versão 3

Com a chegada da HTML5, o Consórcio W3C criou o conceito de marcação poliglota e publicou especificações e diretrizes para a criação de tal marcação.

As especificações definem marcação poliglota assim:

Documento que usa marcação poliglota é um documento HTML5, que é, ao mesmo tempo, um documento HTML e XML escrito conforme um conjunto de regras definidas. Marcação poliglota é processada e compatível com a HTML e com a XHTML, segundo as regras das especificações para a HTML5.

Nessa versão o template usa uma marcação poliglota, conforme segue:

```

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" lang="pt-br" xml:lang="pt-br">
<head>
    <meta charset="utf-8" />
    <title>Template</title>
    <meta name="description" content="Template HTML5 do livro do Maujor." />
    <meta name="keywords" content="lista de palavras-chave" />
    <meta name="author" content="Mauricio Samy Silva" />
    <meta name="generator" content="HTML-Kit 292" />

```

```
<meta name="robots" content="all" />
<link rel="stylesheet" href="mais.css" />
<style>
/* estilos incorporados */
</style>
<script src="path/modernizr-1.5.min.js"></script>
<script src="scripts.js"></script>
</head>
<body>
<!-- Aqui os conteúdos da página --&gt;
&lt;/body&gt;
&lt;/html&gt;</pre>
```

A marcação poliglota deve satisfazer aos seguintes requisitos:

O DOCTYPE, como prevê a XML, deve ser declarado em maiúsculas e o html que se segue ao DOCTYPE deve ser em minúsculas. Lembre-se de que em HTML a sintaxe é insensível ao tamanho de caixa, mas em marcação poliglota é rígida.

Os elementos `html`, `head`, `title` e `body` devem obrigatoriamente constar da marcação.

O elemento `html` deve obrigatoriamente conter os atributos `xmlns`, `lang` e `xml:lang`.

Os namespaces XML permitidos em marcação poliglota são aqueles para as tecnologias SVG e MathML e devem ser declarados nos elementos `svg` e `math` respectivamente, conforme sintaxe mostrada a seguir:

```
<svg xmlns = "http://www.w3.org/2000/svg">...</svg>
<math xmlns="http://www.w3.org/1998/Math/MathML">...</math>
```

O outro namespace permitido é para `xlink` que pode ser declarado tanto no elemento `html` quanto no elemento `svg` que contenha essa funcionalidade, como mostrado a seguir:

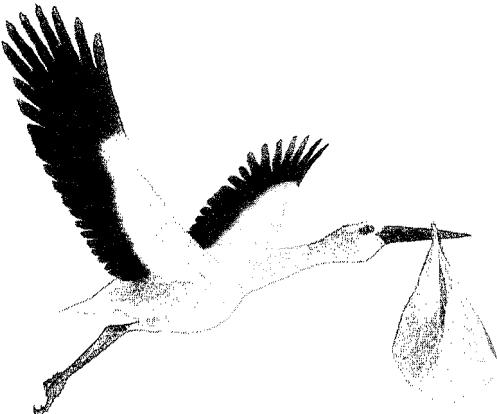
```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:xlink="http://www.w3.org/1999/xlink" lang="pt-br" xml:lang="pt-br">
```

ou

```
<svg xmlns:xlink="http://www.w3.org/1999/xlink">...</svg>
```

A declaração de caracteres deve constar da seção `head` do documento.

A sintaxe da marcação deve seguir as regras da XML. Para maiores informações sobre sintaxe XML, consulte: http://www.maujor.com/w3c/xhtml10_2ed.html e <http://www.maujor.com/w3ctuto/xhtmlfaq.html>.



CAPÍTULO 2

Novidades na HTML5

Neste capítulo teremos uma visão geral das novidades introduzidas pelas especificações da HTML5 mostrando o novo modelo de conteúdo da linguagem. Abordaremos os tópicos que dizem respeito às diferenças entre as linguagens HTML4 e HTML5. Tais diferenças são relativas principalmente à criação de novos elementos e atributos, à redefinição de alguns elementos e atributos existentes, à atualização de elementos e atributos considerados obsoletos, à criação de novas APIs e à extensão das interfaces `HTMLDocument` e `HTMLElement`. Relacionaremos os atributos globais definindo e exemplificando cada um deles. Estudaremos com detalhes os novos elementos da linguagem, definindo-os, esclarecendo as regras de emprego e mostrando exemplos de uso.

2.1 Introdução

A maioria dos elementos e atributos da HTML4 continua válida na HTML5 e, em consequência, nada do que você aprendeu até agora sobre marcação HTML terá sido inútil. Os seus velhos e conhecidos elementos `h1`, `p`, `div`, `span`, `blockquote` bem como os atributos `class`, `title`, `accesskey`, `name`, `value` e tantos outros estão previstos na HTML5 com as mesmas finalidades e valor semântico que tinham na HTML4.

2.2 Modelo de conteúdo

A especificação para a HTML5 estabelece um Modelo de Conteúdo que vem a ser a descrição do tipo de conteúdo que se espera ser inserido em cada elemento. Os conteúdos de cada elemento HTML deverão estar de acordo com o que estabelece o modelo de conteúdo para o elemento.

Cada elemento pertence a uma ou mais categorias que agrupam elementos segundo seu modelo de conteúdo. As categorias de conteúdos são:

- Metadados
- Fluxo
- Seção
- Cabeçalhos
- Frase
- Incorporado
- Interativo
- Formulário

A especificação para a HTML5 mostra uma ilustração de como as categorias de conteúdos se relacionam entre si. Transcrevemos na figura 2.1 a ilustração conforme consta do site <http://www.whatwg.org>.

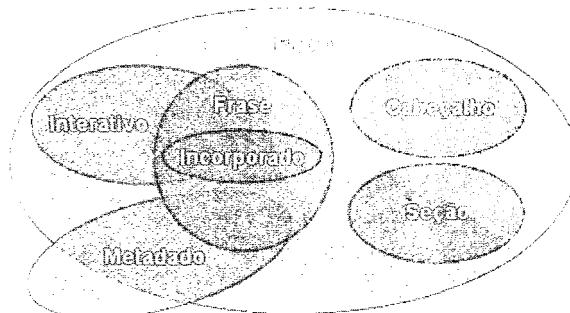


Figura 2.1 – Modelo de conteúdo.

Os elementos de formulários, além de poderem pertencer às categorias de *Fluxo*, de *Frase* e/ou *Interativa*, pertencem também a uma categoria específica denominada *Formulário* que se desdobra nas seguintes subcategorias:

- Listas
- Rótulos
- Envio
- Limpeza

Acredito ser desnecessário definir o tipo de conteúdo de cada categoria, pois o nome da categoria já passa a ideia exata para o tipo.

2.3 Atributos globais

Atributos globais são aqueles que podem ser usados com todos os elementos da HTML5, ou seja, não somente com os elementos novos, mas também com os elementos da HTML4 que agora fazem parte da HTML5.

No item 2.4 apresentaremos os elementos da HTML5 e faremos referências aos atributos que eles suportam. Quando se tratar de elemento que suporte os atributos globais, apenas citaremos seu suporte a eles sem mais considerações, pois, para não repeti-los em cada um dos elementos, vamos examiná-los separadamente e em detalhes nesta seção.

2.3.1 accesskey

(Atributo tecla de acesso) Destina-se a permitir que o usuário dê o foco ao elemento com uso do teclado, ou seja, atribui uma tecla de acesso ao elemento. O valor desse atributo é um caractere ou uma lista de caracteres do teclado, separados por espaço. Observe, nos exemplos a seguir, a sintaxe para uso desse atributo em duas situações diferentes de um documento.

```
<ul>
    <li><a href="/site-a.html" accesskey="a">Site A</a></li>
    <li><a href="/site-b.html" accesskey="b">Site B</a></li>
    <li><a href="/site-c.html" accesskey="c">Site C</a></li>
</ul>
```

Nessa primeira situação, o agente de usuário deve prover meios de dar o foco aos links para os sites A, B e C quando o usuário pressiona as teclas **Ctrl+Shift+a**, **Ctrl+Shift+b**, **Ctrl+Shift+c** respectivamente.

Vejamos outro exemplo:

```
<form action="/busca.php">
    <label>Buscar: <input type="busca" name="q" accesskey="b 0"></label>
    <input type="submit">
</form>
```

Nessa situação, o foco deve ser dado ao campo de busca quando o usuário pressiona as teclas **Ctrl+Shift+b**. Se o usuário estiver usando um dispositivo móvel ou outro similar que possua somente teclado numérico, o foco deve ser dado ao se pressionar a tecla **0**.

O atributo `accesskey` teve seu uso redefinido na HTML5, pois, em versões anteriores da HTML, ele não era um atributo global. Estava restrito aos elementos `a`, `area`, `button`, `input`, `label`, `legend` e `textare`a.

2.3.2 class

(Atributo classe) Destina-se a atribuir uma classe identificadora para o elemento. Trata-se de um atributo identificador e a diferença para o atributo identificador único `id` estudado adiante é que em um mesmo documento o nome da classe pode ser usado para identificar várias instâncias de um mesmo elemento, bem como para identificar elementos diferentes. Observe, nos exemplos a seguir, a sintaxe para uso desse atributo em diferentes elementos de um documento.

```
<p class="destaque">Texto do parágrafo marcado com a classe destaque</p>
<h1 class="destaque corum">Cabeçalho de nível 1 marcado com a classe destaque</h1>
```

Para os elementos mostrados, uma folha de estilo é capaz de identificar os diferentes elementos identificados com a classe `destaque` e estilizar todos eles com a mesma cor, por exemplo. É permitido definir mais de um valor para esse atributo e nesse caso as regras de estilos definidas para ambos os valores serão aplicadas ao elemento. A ordem de declaração dos valores é irrelevante e, havendo conflito de regras de estilo aplicadas aos valores, prevalece o efeito cascata das folhas de estilos.

2.3.3.contenteditable

(Atributo conteúdo editável) Atributo novo, criado pela HTML5. Esse atributo admite os valores `true` e `false` e se destina a permitir que o usuário edite os conteúdos do elemento no navegador. Trata-se de um atributo inventado pela Microsoft e suportado pelos seus navegadores há tempos e que foi introduzido na especificação para a HTML5. É um atributo herdado, pois, quando definido com o valor `true` para um elemento container, faz com que seus elementos-descendentes sejam editáveis.

Observe, a seguir, a sintaxe para uso desse atributo.

```
<div contenteditable="true">
  <h1> Cabeçalho do nível </h1>
  <p>Parágrafo <span>um</span></p>
  <p contenteditable="false">Parágrafo dois</p>
  
</div>
```

Nesse exemplo todos os conteúdos do `div` são editáveis. Para o parágrafo dois não é possível editar o seu conteúdo textual, contudo é possível editá-lo como um todo, apagando-o por exemplo.

2.3.4 contextmenu

(Atributo menu de contexto) Atributo novo, criado pela HTML5 não é suportado por nenhum navegador atual. Esse atributo cria um menu de contexto personalizado para o elemento em que foi inserido. Menu de contexto é aquele que se abre quando o usuário clica com o botão direito do mouse sobre o elemento para o qual ele foi projetado. Se você clicar com o botão direito do mouse em qualquer lugar da barra de ferramentas de seu sistema operacional, ou sobre uma figura em um site da web, abre-se um menu de contexto.

O valor desse atributo deve ser igual ao valor do atributo `id` do container do menu de contexto personalizado. Observe, a seguir, a sintaxe para uso desse atributo.

```
<form name="exemplo">
    <label>Nome base: <input name="nbase" type="text" contextmenu="menu_nome"></label>
    <menu type="context" id="menu_nome">
        <command label="Escolher randomicamente um nome"
            onclick="document.forms.exemplo.elements.nbase.value = escolherNome()">
        <command label="Preencher demais campos"
            onclick="preencherCampos(document.forms.exemplo.elements.nbase.value)">
    </menu>
</form>
```

Nesse exemplo criamos um menu de contexto que se abre quando o usuário entra em um campo de texto de um formulário rotulado como “Nome base”. O evento que dispara a abertura do menu de contexto não é necessariamente o clique com o botão direito do mouse, mas sim depende do elemento em que o menu for atrelado. No nosso caso, o menu está atrelado a um controle `input` do tipo `text` e o menu abrirá quando o usuário der o foco àquele controle.

O menu de contexto deve ser construído com uso do novo elemento `menu` da HTML5.

No exemplo o menu de contexto oferece duas opções ao usuário. Ambas as opções quando escolhidas disparam o funcionamento de uma função JavaScript que preenche o controle `input` do formulário.

2.3.5 dir

(Atributo direção do texto) Destina-se a especificar a direção de escrita do texto. Esse atributo admite dois valores, a saber: `ltr`, sigla em inglês para *left to right*, que traduzimos por “da esquerda para a direita”, que vem a ser a direção da escrita ocidental (por exemplo: o português), e `rtl`, sigla em inglês para *right to left*, que traduzimos por “da direita para a esquerda”, que vem a ser a direção da escrita oriental (por exemplo: o japonês). Observe, nos exemplos a seguir, a sintaxe para uso desse atributo.

```
<p dir="ltr">Texto com atributo de direção ocidental</p>
<p dir="rtl">Texto com atributo de direção oriental</p>
```

Existe a propriedade CSS `direction` que cumpre a mesma finalidade desse atributo e admite os mesmos valores além do valor `inherit`, como mostrado a seguir com estilização in-line.

```
<p style="direction:ltr;">Texto com estilização de direção ocidental</p>
<p style="direction:rtl;">Texto com atributo de direção oriental</p>
```

A especificação para a HTML5 recomenda expressamente o uso do atributo `dir` em lugar da estilização com a propriedade `direction`, com a finalidade de manter a informação sobre a direção do texto acessível para agentes de usuário com CSS desabilitada, como é o caso de mecanismos de busca.

2.3.6 draggable

(Atributo de arraste) Atributo novo, criado pela HTML5. Esse atributo torna arrastável o elemento em que foi inserido. Admite os valores `true`, `false` e `auto`. O valores `true` e `false` fazem o elemento tornar-se arrastável ou não respectivamente e o valor `auto` define o arraste-padrão para essa funcionalidade, conforme implementada no navegador, isto é, se por padrão o elemento é arrastável ele permanece arrastável quando o valor `auto` for definido. Observe, a seguir, a sintaxe para uso desse atributo.

```
<div id="drag" draggable="true"></div>
```

Nesse exemplo o `div#drag` é arrastável na página, mas isso não significa que simplesmente definindo o atributo você possa arrastar livremente o elemento. É preciso estabelecer, com uso de JavaScript, as condições de arraste. A especificação para a HTML5 criou vários atributos e métodos que visam a criar as facilidades de arraste.

2.3.7 hidden

(Atributo de relevância) Atributo novo, criado pela HTML5; não é suportado por nenhum navegador atual. Esse atributo torna irrelevante o elemento em que foi inserido. É um atributo booleano para o qual não há necessidade de se definir valores. Quando presente torna o elemento irrelevante. Os agentes de usuário não devem renderizar elementos marcados com esse atributo.

Esse atributo não deve ser usado meramente para esconder elementos. Sua finalidade é desconsiderar conteúdos irrelevantes para determinado contexto ou não relacionados ao conteúdo atual.

Observe, a seguir, a sintaxe para uso desse atributo.

```
<div id="tela" hidden></div>
```

2.3.8 id

(Atributo identificador único) Destina-se a atribuir um nome identificador para o elemento. Em um mesmo documento, o nome usado para identificar um elemento deve ser único, isto é, não é válido atribuir o mesmo nome de `id` para mais de um elemento. Observe, nos exemplos a seguir, a sintaxe para uso desse atributo em várias instâncias de ocorrência do elemento parágrafo em um documento.

```
<p id="cor-um">Texto do parágrafo marcado com o identificador cor-um</p>
<p id="diferente">Texto do parágrafo marcado com o identificador diferente</p>
<p id="abc3">Texto do parágrafo marcado com o identificador abc3</p>
<p id="cor-dois">Texto do parágrafo marcado com o identificador cor-dois</p>
```

O valor do atributo `id` é um nome que você pode escolher à vontade. Nas versões anteriores da HTML, existiam restrições quanto ao primeiro caractere do nome escolhido, como: não é válido começar o nome de um atributo com um número ou hífen. Por outro lado, o caractere underscore (`_`) é válido. A HTML5, porém, estabelece unicamente que valores de `id` devem ter, pelo menos, um caractere e quando mais de um não pode haver espaço em branco entre eles.

Mas, afinal, qual é a utilidade de se identificar um elemento no documento? Programas, scripts e folhas de estilo têm a capacidade de se comunicar com a marcação HTML e conseguem selecionar elementos com base em seu identificador. Por exemplo: o parágrafo mostrado anteriormente, com `id` cujo valor é `cor-um`, é facilmente identificado por um arquivo de folhas de estilo e estas poderão estilizá-lo com uma cor diferente da adotada para os demais parágrafos do documento.

2.3.9 item*

(Atributo para Microdados) A HTML5 criou o *Modelo Microdados* que é uma extensão de *Microformats* e se destina a fornecer um mecanismo complementar de adição de semântica aos elementos HTML, destinada à leitura por máquinas, como robôs de indexação. Nesse modelo, Microdados é a denominação para um grupo de pares item/propriedades. Itens e propriedades são inseridos como atributos/valores em elementos da marcação. No modelo Microdados, um item relaciona-se com suas propriedades com uso dos atributos `itemscope`, `itemprop`, `itemref`, `itemtype`, `itemid`.

Para maiores detalhes sobre esses novos atributos da HTML5, consulte o capítulo 11.

2.3.10 lang

(Atributo de idioma) Destina-se a definir o idioma do conteúdo de um elemento. Se esse atributo não estiver presente no elemento, o idioma é o mesmo definido no seu elemento-pai. Todos os navegadores suportam esse atributo e é de boa prática sempre declará-lo no elemento-raiz `html` com a finalidade de informar aos agentes de usuário o idioma principal no qual a página foi escrita. Os valores para esse atributo são as abreviaturas dos idiomas do mundo inteiro relacionadas em norma internacional. Observe, nos exemplos a seguir, a sintaxe para uso desse atributo no elemento-raiz de um documento escrito em português do Brasil e em um elemento dentro desse documento marcando um termo escrito em inglês.

```
<html lang="pt-br">
...
<p>O termo inglês <span lang="en">welcome</span> significa bem-vindo</p>
```

2.3.11 spellcheck

(Atributo de correção) Atributo novo, criado pela HTML5; não é suportado por nenhum navegador atual. Esse atributo se destina a disponibilizar a funcionalidade de correção ortográfica e gramatical para os conteúdos textuais inseridos nos elementos para os quais o atributo foi definido. Funciona criando um corretor semelhante à ferramenta corretora de alguns editores de textos, por exemplo, o Word da plataforma Windows. A especificação não define a interface de usuário para a correção. Os valores possíveis para esse atributo são `true` e `false` indicando correção habilitada ou não, respectivamente. Observe, no exemplo a seguir, a sintaxe para uso desse atributo em um campo de texto do tipo `textare`a. No exemplo tudo o que for digitado no campo será submetido ao corretor ortográfico.

```
<textarea rows="8" cols="40" spellcheck="true"></textarea>
```

2.3.12 style

(Atributo para estilização) Destina-se a definir regras de estilo para o elemento em que é aplicado. Com o uso desse atributo, você pode, por exemplo, aplicar cores, mudar o tamanho das letras, posicionamentos e praticamente tudo que diga respeito à apresentação visual do conteúdo do elemento em que o atributo for aplicado. Veja, a seguir, um exemplo de aplicação desse atributo.

```
<p style="color:red;">Um parágrafo estilizado na cor vermelha</p>
```

2.3.13 tabindex

(Atributo de tabulação) Destina-se a definir uma ordem de tabulação, ou seja, a sequência em que os elementos receberão o foco quando o usuário navegar com uso do teclado. Veja, a seguir, um exemplo de aplicação desse atributo.

```
<p tabindex="5">Parágrafo</p>
<p tabindex="1">Parágrafo</p>
<a href="#" tabindex="4">link</a>

<h4 tabindex="3">Título 4</h4>
```

2.3.14 title

(Atributo título) Destina-se a definir um título para o elemento em que é aplicado. Leitores de tela normalmente leem o valor desse atributo. Veja, a seguir, um exemplo de aplicação do atributo.

```
<p title="Conteúdo do título">Um parágrafo para exemplificar o atributo title</p>
```

2.4 Novos elementos

A especificação para a HTML5 introduziu novos elementos de marcação com a finalidade de oferecer aos desenvolvedores melhores opções para estruturar seus documentos. Examinaremos a seguir os novos elementos de marcação, sua apresentação e finalidades, categorias, interface a que pertencem e atributos para eles previstos.

2.4.1 article

Destina-se a marcar um conteúdo autossuficiente em uma página, documento ou aplicação. O conteúdo marcado é, a princípio, reusável e pode ser distribuído de

forma independente, por exemplo: via RSS. Alguns exemplos de conteúdos para serem marcados com esse elemento são: um post em um fórum, um artigo de uma revista ou jornal, uma matéria publicada em um blog, um comentário postado por um visitante, um *widget* ou *gadget* interativo ou qualquer outro conteúdo independente.

Elementos `article` podem ser aninhados e nesses casos o conteúdo aninhado deve estar relacionado com o conteúdo no qual foi aninhado. Por exemplo: uma matéria publicada em um blog que admite comentários dos leitores é apropriada para ser marcada com esse elemento contendo aninhados outros elementos `article` para marcar os comentários, como mostrado a seguir:

```
<article>
  // Marcação HTML para uma matéria do blog
  <article>
    // Comentário 1 sobre a matéria
  </article>
  <article>
    // Comentário 2 sobre a matéria
  </article>
  ...
</article>
```

Atributos para este elemento: atributos globais.

2.4.2 aside

Destina-se a marcar um conteúdo separado, mas tangencialmente relacionado com o conteúdo em volta dele. Em tipografia são os blocos de conteúdos nas barras-laterais. Esse elemento pode ser usado para obter o efeito tipográfico que ele confere ao conteúdo. Mas, cuidado, usar `aside` para marcar conteúdos simplesmente porque estão posicionados à esquerda ou à direita do conteúdo principal nem sempre é uma escolha acertada. A posição do conteúdo candidato a ser marcado com `aside` não deve ser considerada, mas sim considere-se o tipo de conteúdo.

Como regra geral e não absoluta e definitiva, uma boa indicação de conteúdo para `aside` é aquele que, embora relacionado ao conteúdo principal, não prejudicará sua compreensão se for retirado.

Alguns exemplos de conteúdos para serem marcados com esse elemento são: *fullquotes* ou barras-laterais, banners e textos com propaganda, agrupamento de mecanismos de navegação e outros conteúdos que possam ser considerados separados do conteúdo principal. Observe o exemplo a seguir que demonstra o uso

desse elemento para marcar a coluna de um blog contendo links para as categorias de *posts* e *blogroll*.

```
<body>
// Marcação HTML para o topo do blog
<article>
    // Post 1
</article>
<article>
    // Post 2
</article>

...
<aside>
    <nav>
        <h1>Categorias</h1>
        <ul>
            <li><a href="c1">categoria 1</a></li>
            <li><a href="c2">categoria 2</a></li>
        </ul>
        <h1>Blogroll</h1>
        <ul>
            <li><a href="http://blog1.com">Blog 1</a></li>
            <li><a href="http://blog2.com">Blog 2</a></li>
        </ul>
    </nav>
</aside>
...
```

Atributos para este elemento: atributos globais.

2.4.3 audio

Esse elemento será estudado com detalhes no capítulo 3.

2.4.4 canvas

Esse elemento será estudado com detalhes no capítulo 4.

2.4.5 command

Esse elemento destina-se a marcar um comando a ser evocado pelo usuário. Um comando pode fazer parte de um menu de contexto ou barra de ferramenta criada com o elemento `menu` (ver 2.4.6) ou ainda ser um comando isolado na página, por

exemplo, definindo um atalho de teclado. Esse elemento para ser renderizado deve ser usado como elemento-filho do elemento `menu`. Atualmente nenhum navegador oferece suporte para esse elemento.

O elemento `command` admite os seguintes atributos: atributos globais, `type`, `label`, `icon`, `disabled`, `checked`, `radiogroup` e `title`.

O atributo `type` define o tipo de comando, podendo ser do tipo comando associado a uma ação, um comando de escolha de uma opção ou ainda uma seleção. Os valores possíveis para esse atributo são: `command`, `checkbox` e `radio`. O valor-padrão é `command` que cria um comando normal associado a uma ação. O valor `checkbox` cria um comando do tipo caixa de seleção de opções e o valor `radio` um comando tipo seleção do item de uma lista.

O atributo `label` destina-se a definir um nome para o comando e deve ser uma string não vazia.

O atributo `icon` destina-se a definir um elemento gráfico para representar o comando.

O atributo `disabled` é booleano e, quando presente, destina-se a desabilitar comando.

O atributo `checked` é booleano e, quando presente, destina-se a selecionar uma opção do comando. Não deve ser usado se o comando é do tipo `command`.

O atributo `radiogroup` destina-se a agrupar comandos do tipo `radio`. Só deve ser usado se o comando é do tipo `radio`.

O atributo `title` destina-se a fornecer ao usuário uma dica descritiva do comando.

2.4.6 menu

Esse elemento destina-se a marcar uma lista de comandos e é usado como elemento container para elementos `command`. Atualmente nenhum navegador oferece suporte para esse elemento.

O elemento `menu` admite os seguintes atributos: atributos globais, `type` e `label`.

O atributo `type` define o tipo de menu que está sendo criado. Os valores possíveis para esse atributo são: `context`, `toolbar` e `list`. O valor-padrão é `list` que simplesmente cria uma lista de comandos sem qualquer ação. O valor `context` cria um menu do tipo contexto e o valor `toolbar` um menu tipo barra de ferramenta.

Observe a seguir dois exemplos de uso dos elementos `command` e `menu`.

Exemplo 1: Nesse exemplo criamos um menu do tipo barra de ferramentas com três opções para alinhamento: à *esquerda*, no *centro* e à *direita*. Cada opção poderá ser acionada por clique em um ícone que irá disparar a execução de uma função JavaScript. O exemplo a seguir esclarece o uso dos atributos estudados anteriormente.

```
<menu type="toolbar">
    <command type="radio" radiogroup="alinhamento" checked="checked"
        label="Esquerda" icon="icons/alL.png" onclick="setAlign('esq')">
    <command type="radio" radiogroup="alinhamento"
        label="Centro" icon="icons/alC.png" onclick="setAlign('centro')">
    <command type="radio" radiogroup="alinhamento"
        label="Direita" icon="icons/alR.png" onclick="setAlign('dir')">
</menu>
```

Exemplo 2: Nesse exemplo criamos um menu do tipo barra de ferramentas com três itens: *Arquivo*, *Editiar* e *Ajuda*. Cada item contém uma lista de opções de escolha para realizar uma ação mediante disparo de uma função JavaScript. Para o item *Arquivo* a ação é seguir um link.

```
<menu type="toolbar">
    <li>
        <menu label="Arquivo">
            <button type="button" onclick="fnovo()">Novo...</button>
            <button type="button" onclick="fabrir()">Abrir...</button>
            <button type="button" onclick="fsalvar()">Salvar</button>
            <button type="button" onclick="fsalvarcomo()">Salvar como...</button>
        </menu>
    </li>
    <li>
        <menu label="Editiar">
            <button type="button" onclick="ecopiar()">Copiar</button>
            <button type="button" onclick="ecortar()">Recortar</button>
            <button type="button" onclick="ecolar()">Colar</button>
            <button type="button" onclick="eexcluir()">Excluir</button>
        </menu>
    </li>
    <li>
        <menu label="Ajuda">
            <li><a href="help.html">Ajuda</a></li>
            <li><a href="about.html">Sobre</a></li>
        </menu>
    </li>
</menu>
```

Observe na figura 2.2 uma opção de renderização do menu correspondente à marcação do exemplo 2.

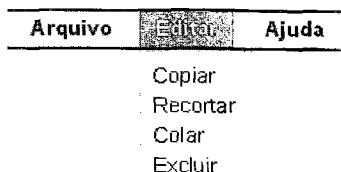


Figura 2.2 – Elementos menu e command.

2.4.7 datalist

Esse elemento foi criado para ser usado em conjunto com o elemento `input`. Destina-se a fornecer uma lista de sugestões para preenchimento do campo `input` e permite que o usuário selecione uma das opções de uma lista de opções predefinida como sugestão pelo desenvolvedor ou então entre no campo um valor diferente daqueles constantes da lista de opções sugeridas. Navegadores que não oferecem suporte para esse elemento renderizam normalmente o campo `input`, já que se trata de um elemento previsto nas especificações anteriores da HTML e largamente suportado pelos navegadores. Atualmente somente os navegadores Opera e Firefox oferece suporte para esse elemento.

Para o elemento `input` foi criado pela HTML5 um atributo novo. Ele permite relacionar o campo com a lista de opções sugeridas. Trata-se do atributo `list`. Define-se um valor qualquer para o atributo `list` do campo `input` e um `id` com o mesmo valor para o elemento `datalist` criando um vínculo entre o campo e a lista de opções sugeridas. A marcação mostrada no exemplo a seguir esclarece a criação desse vínculo.

O exemplo mostrado a seguir cria uma lista de opções sugeridas, para preenchimento de um campo destinado a coletar o nome de uma fruta. Esse exemplo está disponível no site do livro e atualmente deverá ser carregado no navegador Opera 11+ ou Firefox 4+ para se visualizar seu efeito.

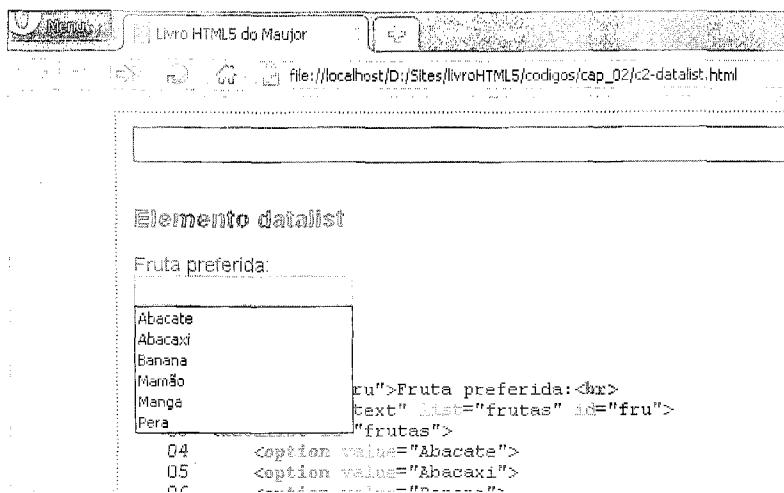
```
<label for="fru">Fruta preferida:</label><br>
<input type="text" list="frutas" id="fru">
<datalist id="frutas">
  <option value="Abacate">
  <option value="Abacaxi">
  <option value="Banana">
  <option value="Mamão">
```

```
<option value="Manga">
<option value="Pera">
</datalist>
```



[c2-datalist.html]

Ao dar-se o foco ao campo aparece a lista de sugestões. Caso a opção do usuário não conste da lista, ele poderá entrar normalmente o nome da fruta de sua preferência no campo de texto. Observe na figura 2.3 a renderização desse elemento no navegador Opera.

Figura 2.3 – Elemento *datalist*.

No navegador Firefox o comportamento é diferente. As sugestões só aparecem quando o usuário digita a primeira letra.

2.4.8 details

Esse elemento destina-se a fornecer informações textuais ou controles de formulário adicionais sobre determinada página ou sobre um conteúdo específico da página. Por padrão os conteúdos fornecidos por esse elemento não são colocados à vista do usuário e o acesso a eles se dá mediante ação do usuário, normalmente por clique em um elemento. Ao usuário é facultada a opção de mostrar ou esconder os conteúdos.

O elemento *details* admite os seguintes atributos: atributos globais e *open*. O atributo *open* quando presente faz com que os conteúdos adicionais sejam colocados à vista quando o elemento é renderizado. Esse elemento poderá ser usado em conjunto com o elemento *summary* (ver 2.4.9, a seguir).

2.4.9 summary

Esse elemento foi criado para uso em conjunto com o elemento `details` e destina-se a fornecer um rótulo ou breve sumário das informações adicionais fornecidas por `details`. Trata-se de um elemento primeiro-filho do elemento `details` e que admite apenas os atributos globais.

A especificação prevê que se esse elemento não for usado como elemento-filho de `details` o agente de usuário deverá fornecê-lo nativamente com o conteúdo textual “*Details*”. Experimente fazer uma cópia do exemplo fornecido no site retirando o elemento `summary` da marcação para visualizar essa funcionalidade. Experimente também usar o atributo `open` no elemento `details`.

O exemplo a seguir esclarece o uso dos elementos `details` e `summary`, mas nenhum navegador atual oferece suporte para esses elementos. No entanto, existe um plugin jQuery desenvolvido por Mathias Bynens (<http://github.com/mathiasbynens/noSelect-jQuery-Plugin>) que simula o comportamento do navegador para esses elementos.

Na página de exemplo disponível no site do livro usamos o plugin de Mathias e você poderá visualizar o comportamento do navegador para o exemplo em qualquer navegador recente.

```
<details>
  <summary>Livros do Maujor</summary> // clicando esse elemento mostra/oculta o conteúdo adicional
  <dl>
    <dt>Construindo sites com CSS e (X)HTML</dt>
      <dd>Ano: 2007</dd>
      <dd>Páginas: 438</dd>
    <dt>jQuery - A biblioteca do programador JavaScript</dt>
      <dd>Ano: 2008</dd>
      <dd>Páginas: 543</dd>
    <dt>AJAX com jQuery - Requisições AJAX com a simplicidade de jQuery</dt>
      <dd>Ano: 2009</dd>
      <dd>Páginas: 327</dd>
    <dt>JavaScript - Guia do programador</dt>
      <dd>Ano: 2010</dd>
      <dd>Páginas: 604</dd>
  </dl>
</details>
```



[c2-details.html]

2.4.10 figure

Esse elemento destina-se a servir de container para conteúdos independentes relacionados a um conteúdo específico no fluxo do documento. Use esse elemento para marcar conteúdos, como imagens, ilustrações, diagramas, fotos, gráficos, vídeos, linhas de códigos e similares. Os conteúdos desse elemento, embora relacionados ao conteúdo principal, podem ser retirados do fluxo principal e serem posicionados tangencialmente a ele, ou mesmo movidos para uma página específica ou um apêndice.

Esse elemento admite apenas os atributos globais e poderá ser usado em conjunto com o elemento `figcaption` (ver 2.4.11).

2.4.11 figcaption

Esse elemento destina-se a marcar uma legenda para o conteúdo inserido com uso do elemento `figure` e deve constar da marcação como elemento-filho do elemento `figure`. Admite apenas os atributos globais.

Os exemplos a seguir esclarecem alguns usos desses elementos.

Exemplo 1: marcar uma imagem

```
<figure>
  
  <figcaption>Copacabana a eterna princesinha do mar.</figcaption>
</figure>
```

Exemplo 2: marcar um vídeo

```
<figure>
  <video src="matrix.mov"></video>
  <figcaption>Trailer do filme Matrix.</figcaption>
</figure>
```

Exemplo 3: marcar parte de um poema

```
<figure>
  <p>Meu canto de morte,<br>
    Guerreiros, ouvi: <br>
    Sou filho das selvas, <br>
    Nas selvas cresci; <br>
    Guerreiros, descendo<br>
    Da tribo tupi.</p>
```

```
<figcaption>
  <cite>I-Juca-Pirama</cite> 1<sup>o.</sup> verso do IV canto do poema épico de Gonçalves Dias.
</figcaption>
</figure>
```

Exemplo 4: marcar um trecho de código

```
<figure>
  <figcaption>Regra CSS para estilização do elemento H1.</figcaption>
  <pre><code>h1 {
    font: italic 22px sans-serif;
    color: red;
    text-align: center;
  }
</code></pre>
</figure>
```

2.4.12 footer

Esse elemento destina-se a marcar o rodapé de uma seção ou da página como um todo e deve conter informações sobre o conteúdo da seção ou página, como o seu autor, links para documentos relacionados, direitos autorais e similares. Admite apenas os atributos globais.

É importante notar que a HTML5 altera radicalmente a semântica e uso desse elemento em relação às versões anteriores da HTML. O conceito tradicional de rodapé é que cada página tenha seu rodapé único normalmente marcado com um `:div#rodapé` (ou `div#footer`).

Em HTML5 uma página poderá conter vários rodapés, pois é válido que cada seção da página tenha seu próprio rodapé, bem como a página pode também conter seu rodapé à maneira como estamos acostumados a marcar em (X)HTML.

O elemento `footer` em si não cria na marcação uma nova seção tal como criam os elementos `article`, `aside`, `nav` e `section`.

Outra mudança no uso desse elemento diz respeito ao posicionamento do rodapé em relação ao conteúdo da seção no qual ele está inserido, pois nada impede que o rodapé seja marcado no início da seção, não havendo obrigatoriedade de ele ser posicionado no final, muito embora essa seja a posição mais comum para o rodapé.

Convém ressaltar que as informações de contato sobre o autor ou editor de uma seção ou página devem ser marcadas com uso do elemento `address` e este, por sua vez, poderá estar contido no elemento `footer`.



O elemento **address** destina-se a fornecer informações de contato do autor ou editor, portanto não deve ser usado para marcar endereços em geral. Para marcar endereços que não sejam do autor ou editor use o elemento **p** (parágrafo).

Os exemplos a seguir esclarecem alguns usos para o elemento **footer**.

Exemplo 1:

```
<article>
    <!-- conteúdo do elemento article -->
    <footer>
        <!-- informações sobre o conteúdo de article -->
    </footer>
</article>
```

Exemplo 2:

```
<nav>
    <!-- conteúdo do elemento nav -->
    <footer>
        <!-- informações sobre o conteúdo de nav -->
    </footer>
</nav>
```

Exemplo 3:

```
<body>
    <!-- conteúdos da página -->
    <section>
        <article>
            <!-- conteúdo do elemento article -->
            <footer>
                <!-- informações sobre o conteúdo de article -->
            </footer>
        </article>
        <footer>
            <!-- informações sobre o conteúdo de section -->
        </footer>
    </section>
    <footer>
        <!-- informações sobre o conteúdo da página -->
    </footer>
</body>
```

Exemplo 4:

```
<body>
<footer><a href="#">Ir para a Home Page</a></footer>
<hgroup>
  <h1>Lorem ipsum</h1>
  <h2>The ipsum of all lorem</h2>
</hgroup>
<p>A dolor sit amet, consectetur ...</p>
<footer><a href="#"> Ir para a Home Page</a></footer>
</body>
```

2.4.13 header

Esse elemento destina-se a marcar o cabeçalho de uma seção ou da página como um todo e deve conter os elementos h1-h6 (não obrigatoriamente) bem como informações sobre o conteúdo da seção. Também pode ser usado para marcar uma tabela de conteúdos para a seção, um formulário de busca ou logotipos importantes para a página. Admite apenas os atributos globais.

Uma página poderá conter vários elementos header e estes, por sua vez, os elementos h1-h6. Assim, é válido marcarmos na mesma página vários elementos h1-h6.

O elemento header em si não cria na marcação uma nova seção tal como criam os elementos article, aside, nav e section.

Os exemplos a seguir esclarecem alguns usos para o elemento header.

Exemplo 1:

```
<body>
<header>
  <h1>Título principal da página</h1>
</header>
// conteúdo da página
</body>
```

Exemplo 2:

```
<article>
<header>
  <h1>Título da seção article</h1>
  <ul>
    <li>Data:...</li>
    <li>Hora:...</li>
```

```

        <li>Categoria:...</li>
    </ul>
</header>
// conteúdo do elemento article
</article>
```

Exemplo 3:

```

<body>
<header>
    <h1>Jogos online</h1>
    <nav>
        <ul>
            <li><a href="/jogos">Jogos</a>
            <li><a href="/forum">Forum</a>
            <li><a href="/download">Download</a>
        </ul>
    </nav>
    <h2>Notícia importante</h2> <!-- aqui começa a segunda subseção -->
    <!-- conteúdo da subseção "Notícia importante" -->
    <p>Atualize seu software para o jogo de hoje.</p>
    <h2>Jogos</h2> <!-- aqui começa a terceira subseção -->
</header>
<p>Você tem três jogos ativos:</p>
<!-- conteúdo da subseção "Jogos" -->
```

2.4.14 hgroup

Esse elemento destina-se a agrupar os elementos h1-h6, quando existir mais de um deles como cabeçalho de uma seção ou da página. Admite apenas os atributos globais.

Uma página poderá conter vários elementos hgroup.

Os exemplos a seguir esclarecem o uso para esse elemento.

Exemplo 1:

```

<hgroup>
    <h1>Capítulo 2</h1>
    <h2>Novidades na HTML5</h2>
</hgroup>
```

Exemplo 2:

```
<article>
  <header>
    <hgroup>
      <h2>Título do artigo</h2>
      <h3>Subtítulo do artigo</h3>
    <hgroup>
      <p><time>18 dezembro de 2010</time></p>
  </header>
  <!-- conteúdo de article -->
</article>
```

2.4.15 keygen

Esse elemento destina-se a manipular um par de chaves criptografadas, públicas e privadas para autenticação da comunicação com o servidor por ocasião do envio de formulários.

O elemento keygen admite os seguintes atributos: atributos globais, autofocus, challenge, disabled, form, keytype e name.

Atualmente não existe um consenso sobre como implementar esse elemento e nem mesmo o reconhecimento por parte dos fabricantes de software sobre sua relevância. Assim, não entraremos em maiores detalhes deixando registrado apenas sua existência nas especificações para a HTML5.

2.4.16 mark

Esse elemento destina-se a marcar uma palavra ou trecho de texto que deva ser destacado com o propósito de servir como referência. Admite apenas os atributos globais.

Por exemplo: em uma citação usamos esse elemento para marcar determinada palavra que se tornou relevante para o contexto em que a citação foi inserida.

Não confundir esse elemento com o elemento strong. O elemento strong destina-se a marcar um trecho importante ao qual se pretende dar forte ênfase e o elemento mark um trecho relevante para o contexto no qual está inserido e para o qual se pretende dar destaque.

Atualmente os navegadores renderizam o trecho de texto marcado com esse elemento com um fundo na cor amarela, destacando-o visualmente do restante do texto.

2.4.17 meter

Esse elemento destina-se a marcar uma medida escalar compreendida entre determinados limites conhecidos.

Esse elemento não deve ser usado para marcar uma medida representativa do andamento de uma tarefa, por exemplo, a porcentagem de arquivo sendo baixada para o disco local em uma operação de download. Para isso existe o elemento `progress` que estudaremos adiante. Também não é apropriado o uso desse elemento para marcar uma medida representativa de uma porção da qual não se conhecem os limites superiores e inferiores, por exemplo, a altura de uma pessoa, já que não existe limite superior e inferior para a altura de pessoas.

Alguns exemplos de uso adequado para o elemento `meter` seriam: para marcar a porcentagem de uso do espaço em disco, o número de votos obtidos por um candidato em uma eleição na qual a quantidade total de votantes é conhecida ou a quilometragem atingida por um corredor de maratona em determinado instante.

O elemento `meter` admite os seguintes atributos: atributos globais, `value`, `min`, `max`, `low`, `high` e `optimum`.

O atributo `value` define o valor da medida marcada e é um atributo de uso obrigatório.

O atributo `min` define o valor mínimo da escala à qual a medida marcada pertence e se for omitido será considerado igual a 0.

O atributo `max` define o valor máximo da escala à qual a medida marcada pertence e se for omitido será considerado igual a 1.

O atributo `low` define um valor dentro da faixa à qual a medida pertence e abaixo do qual se considera um valor baixo.

O atributo `high` define um valor dentro da faixa à qual a medida marcada pertence e acima do qual se considera um valor alto.

O atributo `optimum` define um valor considerado ótimo dentro da faixa à qual a medida marcada pertence.

Se o valor dos atributos `min` e `max` não for especificado, eles serão, por padrão, iguais a 0 e 1 respectivamente.

Recomenda-se que os autores forneçam uma alternativa textual a ser inserida como conteúdo desse elemento para garantir o acesso à informação transmitida por esse elemento em agentes de usuário que não suportam o elemento `meter`.

O atributo global `title` deve ser usado para indicar a unidade de medida do valor representado pelo elemento.

Os exemplos a seguir esclarecem o uso para esse elemento.

Exemplos:

```
<meter value="0.7"></meter>      // 0,7 em um faixa de 0 a 1 ou seja 70%
<meter value="8" min="2" max="20"></meter>    // 8 em uma escala que vai de 2 a 20
<meter value="9" min="0" max="10" low="4" high="8" optimum="10"></meter>
// 9 em uma escala que vai de 0 a 10 e para a qual valores entre 0 e 4 são considerados baixos,
// entre 8 e 10 altos e o valor ótimo é 10
<meter value="7" min="0" max="10">7 pontos em 10 possíveis</meter> // com texto alternativo
<meter value="80" min="0" max="120" title="Km/h">Velocidade de 80Km/h</meter> // com atributo title
```

Atualmente os únicos navegadores que suportam esse elemento é o Chrome e o Opera que renderizam uma barra retangular na cor cinza preenchida em cor sólida em uma porção correspondente ao valor representado.

2.4.18 nav

Esse elemento destina-se a marcar uma seção da página que contenha links para outras páginas ou para outras partes dentro da própria página, ou seja, uma seção contendo links para navegação externa ou interna. Admite apenas os atributos globais.

Nem todos os grupamentos de links em uma página são candidatos a serem marcados com esse elemento. Somente os maiores grupos de link devem ser com ele marcados. Por exemplo: é comum no rodapé do site a inserção de um pequeno grupo de links para algumas páginas do site, como as de termos de serviço, direitos autorais, política de privacidade, home page etc. Para esses grupamentos o elemento footer é suficiente e não devemos usar o elemento nav.

O exemplo a seguir esclarece o uso desse elemento em duas seções de uma página.

```
<body>
<h1>A linguagem de marcação HTML5</h1>
<nav>
  <ul>
    <li><a href="/">Home</a></li>
    <li><a href="/codigos">Códigos dos exemplos</a></li>
    ...mais links internos ao site...
  </ul>
</nav>
<article>
```

```
<header>
  <h1>Elemento Canvas</h1>
  <p>A API para gráficos criados com JavaScript.</p>
</header>
<nav>
  <ul>
    <li><a href="#metodos">Métodos da API</a></li>
    <li><a href="#atributos">Atributos da API</a></li>
    ...mais links internos à página...
  </ul>
</nav>
<div>
  <section id="metodos">
    <h1>Métodos para formas complexas</h1>
    <p>...mais conteúdo...</p>
  </section>
  <section id="atributos">
    <h1>Atributos para estilização</h1>
    <p>... mais conteúdo...</p>
  </section>
  ... mais conteúdo...
</div>
<footer> // não usado o elemento nav
  <p><a href="?editar">Editar</a> | <a href="?apagar">Apagar</a> |
    <a href="?renomear">Renomear</a></p>
</footer>
</article>
<footer>
  <p><small>© copyright 2011 Livro HTML5 do Maujor</small></p>
</footer>
</body>
```

2.4.19 output

Esse elemento destina-se a servir de container para o resultado de um cálculo. Para esclarecer sua finalidade, vamos apresentar um exemplo prático de uso bastante comum em sites de compra online.

Considere um formulário no qual o usuário define, em um de seus campos, o número de itens de determinada mercadoria que deseja comprar e cujo preço unitário é conhecido no site.

Ao preencher o campo “quantidade”, imediatamente aparece em um campo “valor total” o resultado da multiplicação da quantidade pelo valor unitário do produto a adquirir. Na HTML atual não existe um elemento específico para marcar o resultado da multiplicação, e o que se faz é usar um campo de texto (elemento `input type="text"`) para apresentar o valor total.

A HTML5 criou o elemento `output` para marcar resultados de cálculos, que é o indicado para o “valor total” do nosso exemplo. Observe a marcação a seguir que esclarece o uso do elemento `output`.

► HTML

```
<p>Preço unitário do produto: R$320,00</p>
<form name="formulario" method="get" action="">
    <p><label>Quantidade: <input type="text" name="qde"></label></p>
    <p>Valor total: R$<output name="vtotal"></output></p>
</form>
```

► JavaScript

```
function calculaValorTotal() {
    var qdeItens = document.forms['formulario']['qde'].value;
    var valorTotal = (qdeItens * 320).toFixed(2);
    document.forms['formulario']['vtotal'].value = valorTotal;
}
window.onload = function() {
    document.forms['formulario']['qde'].onblur = calculaValorTotal;
}
```

O elemento `output` admite os seguintes atributos: atributos globais, `for`, `form` e `name`.

O atributo `for` destina-se a associar o elemento `output` aos controles do formulário ao qual ele está referenciado. O valor desse atributo deve ser uma lista de valores igual aos valores do atributo `id` de cada controle do formulário, como mostrado a seguir.

```
<form name="formulario" method="get" action="">
    <p><label>Quantidade: <input type="text" name="qde" id="quantidade"></label></p>
    <p>Valor total: R$<output name="vtotal" for="quantidade"></output></p>
</form>
```

O atributo `form` destina-se a associar de forma única o elemento `output` ao formulário ao qual ele pertence. O valor desse atributo deve ser igual ao valor do atributo `id` do formulário, como mostrado a seguir.

```
<form name="formulario" method="get" action="" id="form-um">
  ...
  <p><output form="form-um" name="vtotal"></output></p>
</form>
```

O atributo `name` destina-se a declarar um nome identificador para o elemento `output`.

2.4.20 progress

Esse elemento destina-se a marcar o andamento (ou progresso) de uma tarefa. A medida do andamento pode ser feita de duas formas distintas.

Forma indeterminada: quando a medição informa o progresso da tarefa, mas não há uma forma clara de quantificar o quanto falta para a sua conclusão, por exemplo, em tarefas que dependem de uma resposta do servidor.

Forma determinada: quando a medição é feita com um valor numérico compreendido entre 0 (zero) e um valor máximo, sendo assim possível determinar a porcentagem realizada e faltante para a conclusão da tarefa.

O elemento `progress` admite os seguintes atributos: atributos globais, `value`, `max` e `form`.

O atributo `value` destina-se a definir o quanto de tarefa já foi realizada.

O atributo `max` destina-se a definir o quanto é necessário para realizar a tarefa.

O atributo `form` destina-se a associar de forma única o elemento `progress` ao formulário ao qual ele pertence, quando for o caso. O valor desse atributo deve ser igual ao valor do atributo `id` do formulário.

Observe a marcação a seguir que esclarece o uso do elemento `progress`.

```
<section>
  <h2>Andamento da tarefa</h2>
  <p>Progresso: <progress id="p" max="100"><span>0</span>%</progress></p>
  <script>
    var barraAndamento = document.getElementById('p');
    function updateProgress(valorAtual) {
      barraAndamento.value = valorAtual;
      barraAndamento.getElementsByTagName('span')[0].textContent = valorAtual;
    }
  </script>
</section>
```

2.4.21 ruby

Esse elemento destina-se a marcar pequenas anotações sobre textos, contendo informações sobre a maneira correta de pronúncia do texto. Trata-se de uma forma tipográfica usada em escrita oriental sem similares na escrita ocidental e que aqui consta apenas a título de informação. Não faremos maiores considerações sobre esse elemento e seus elementos relacionados `rp` e `rt`. Para maiores informações sobre esses elementos, consulte os itens 4.6.19, 4.6.20 e 4.6.21 das especificações para a HTML5.

2.4.22 section

Esse elemento destina-se a marcar, genericamente, uma seção na página. Entende-se como seção um grupamento de conteúdos tratando de um mesmo tema e tipicamente contendo um cabeçalho. Admite apenas os atributos globais.

São exemplos de seções: capítulos de um livro, abas de uma caixa de diálogo, seções numeradas de uma tese de doutorado ou ainda as seções introdução, informações de contato e novidades de uma homepage.

Não se deve considerar o elemento `section` como um container genérico. Quando precisarmos de um elemento container genérico a ser usado para fins de estilização ou para ser manipulado por script, devemos empregar o elemento `div`.

O exemplo a seguir esclarece o uso desse elemento. Nele mostramos um artigo sobre a região geográfica sul do Brasil onde usamos três vezes o elemento `section`.

```
<article>
  <hgroup>
    <h1>Região sul do Brasil</h1>
    <h2>A região com o maior índice de alfabetização do Brasil</h2>
  </hgroup>
  <p>A região sul do Brasil compreende os estados do Rio Grande do Sul, Santa Catarina e Paraná.</p>
  <section>
    <h1>Rio Grande do Sul</h1>
    <p>O maior e mais populoso estado da região sul possui clima temperado...</p>
  </section>

  <section>
    <h1>Santa Catarina</h1>
    <p>Santa Catarina possui as mais lindas praias do litoral brasileiro...</p>
  </section>
```

```
<section>
  <h1>Paraná</h1>
  <p>No Paraná encontramos a mata de araucária uma das mais importantes florestas
  subtropicais do mundo...</p>
</section>
</article>
```

Notar no exemplo mostrado que cada `section` tem seu elemento `h1`. Usar mais de um cabeçalho do nível 1 ou de qualquer outro dos seis níveis possíveis em uma mesma página é perfeitamente válido em HTML5.

2.4.23 source

Esse elemento permite que se definam múltiplas alternativas para arquivos de mídia. Devem ser usados como elementos-filho dos elementos `audio` e `video` apontando para diferentes arquivos alternativos para uma das duas mídias. Esse elemento será estudado com detalhes no capítulo 3.

2.4.24 time

Esse elemento destina-se a marcar tanto um horário na faixa 0-24h quanto uma data do calendário gregoriano. Opcionalmente pode marcar, também, a diferença do horário local para o horário GMT (time-zone offset). A finalidade desse elemento é fornecer uma codificação para data-hora de modo que máquinas possam ler o código e processar, por exemplo, agendando eventos em um calendário ou lembrando datas de aniversário.

O elemento `time` admite os seguintes atributos: atributos globais, `datetime` e `pubdate`.

O atributo `datetime`, quando for especificado, destina-se a definir o horário e/ou data que está sendo marcado. Se não for especificado esse atributo, o horário e/ou data deverá ser inserido como conteúdo do elemento. O valor desse atributo deve ser uma string representativa do grupo data-hora em formato válido para a linguagem HTML.

O atributo `pubdate` é booleano e quando for especificado informa que o horário e/ou data marcados se referem ao instante em que o conteúdo foi publicado. Uma página poderá conter somente um elemento `time` com o atributo `pubdate` que se refere ao instante de publicação da página. Contudo, dentro de uma mesma página, elementos `article` diferentes podem conter diferentes elementos `time` com datas de publicação diferentes.

Os exemplos a seguir esclarecem o uso desse elemento.

Exemplo1: com uso do atributo pubdate

```
<article>
    <h1>Tarefas</h1>
    <footer>Publicado em: <time pubdate>19-12-2010</time>.</footer>
    <p>Trocar a lâmpada da sala.</p>
</article>
```

Exemplo2: com uso dos atributos pubdate e datetime. Nesse exemplo navegadores que não suportam HTML5 renderizam “Hoje” e os que suportam renderizam a data em formato local.

```
<article>
    <h1>Tarefas</h1>
    <footer>Publicado em: <time pubdate datetime="2010-12-19">Hoje</time>.</footer>
    <p>Trocar a lâmpada da sala.</p>
</article>
```

Exemplo3: com uso dos atributos pubdate e datetime. Nesse exemplo navegadores que não suportam HTML5 nada renderizam e os que suportam renderizam a data em formato local.

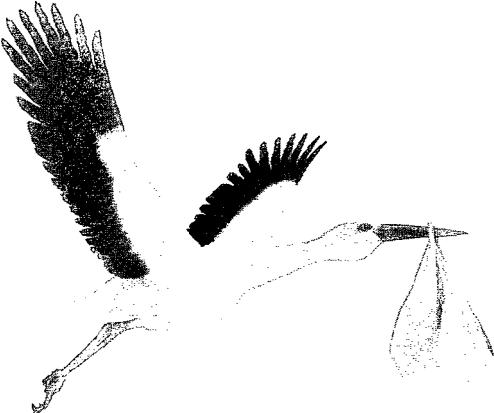
```
<article>
    <h1>Tarefas</h1>
    <footer>Publicado em: <time pubdate datetime="2010-12-19T09:15Z"></time>.</footer>
    <p>Trocar a lâmpada da sala.</p>
</article>
```

2.4.25 track

Esse elemento será estudado com detalhes no capítulo 3.

2.4.26 video

Esse elemento será estudado com detalhes no capítulo 3.



CAPÍTULO 3

Áudio e vídeo

Neste capítulo, serão apresentados os elementos `audio` e `video`. Estudaremos como inserir mídia em um documento HTML5 mostrando as opções para os tipos de arquivos-padrão, para inserção de mídia. Analisaremos como fazer a conversão de arquivos de vídeo com uso de ferramentas gratuitas disponíveis na Internet. Serão mostradas as funcionalidades da API de mídia, descrevendo-se e exemplificando os atributos, métodos e eventos da API. Serão mostrados vários exemplos, disponíveis no site do livro, que esclarecem a sintaxe e uso dos elementos de mídia da HTML5.

3.1 audio

Esse elemento destina-se a incorporar um som ou um *stream* de áudio.

Observe o exemplo a seguir que demonstra o uso desse elemento.

```
<audio src="som.mp3">  
</audio>
```

Somente isso, simples e direto, sem necessidade de plugins, Flash, QuickTime, Windows Media Player, Real Audio e toda a parafernália a ser considerada quando se pretende incorporar som em uma página web.

Basta declarar o caminho para o arquivo de som no atributo `src` do elemento. O som é reproduzido com as funcionalidades nativas do navegador.

Navegadores que não dão suporte ao elemento `audio` renderizam normalmente o conteúdo desse elemento. Assim, para servir o som para esses navegadores, definimos a incorporação do som de maneira como fazemos em (X)HTML, por exemplo: com uso de Flash. Observe o exemplo a seguir.

```
<audio src="som.mp3">
    <!-- Código (X)HTML para inserção de som.mp3 com Flash -->
</audio>
```

Nesse exemplo, navegadores que suportam HTML5 ignoram o código para inserção do som com Flash e os que não suportam executam o código Flash.

Ainda não há um consenso sobre qual *codec* adotar para ser implementado nativamente. A tendência atual é deixar por conta do fabricante a implementação de MP3 ou Ogg Vorbis. Isso nos obriga a servir marcação para ambos os formatos se pretendemos criar uma implementação crossbrowser.

Felizmente o elemento `audio` admite como elemento-filho o elemento `source` cuja finalidade é substituir o atributo `src` e permitir formatos alternativos para o som a ser servido.

O elemento `source` destina-se a permitir que se definam arquivos diferentes para incorporar as mídias `audio` e `video` em uma página.

O elemento `source` admite os seguintes atributos: atributos globais, `src`, `type`, e `media`.

O atributo `src` aponta para o endereço do arquivo de mídia a incorporar na página. É um atributo obrigatório cujo valor é um URL não vazio.

O atributo `type` destina-se a informar ao navegador o tipo de mídia a ser incorporado de modo que ele possa decidir o que fazer antes de baixar o arquivo. O valor desse atributo deve ser um *MIME type* válido.

O atributo `type` admite ainda o parâmetro `codecs` que determinados *MIME type* definem e deve ser informado para que o navegador saiba como o arquivo foi codificado.

O atributo `media` destina-se a informar ao navegador o tipo de mídia. Os valores possíveis para esse atributo são aqueles listados nas especificações para Media Queries (<http://dev.w3.org/csswg/css3-mediaqueries/>) e o valor-padrão é `all`.

Observe o exemplo a seguir que esclarece o uso do elemento `source` e seus atributos.

```
<audio>
    <source src="som.ogg" type="audio/ogg; codecs='vorbis'" media="screen">
    <source src="som.mp3" type="audio/mpeg; codecs='mp3'" media="screen">
    <!-- Código (X)HTML para inserção de som.mp3 com Flash -->
</audio>
```

Nesse exemplo escrevemos um código crossbrowser para servir som com HTML5.

Alternativamente podemos inserir marcação (X)HTML no elemento `audio` com a finalidade de servir navegadores que não suportam qualquer marcação para inserção de som, como mostrado a seguir.

```
<audio>
  <source src="som.ogg" type="audio/ogg">
  <source src="som.mp3" type="audio/mpeg">
  <!-- Código (X)HTML para inserção de som.mp3 com Flash -->
  <p>Seu navegador não suporta o elemento audio da HTML5.
  <br>Faça <a href="som.mp3">download de som.mp3</a></p>
</audio>
```

Mas, cuidado. Marcação (X)HTML inserida no elemento `audio` como mostrado anteriormente não cumpre finalidade de oferecer alternativa textual para o som, pois ela é renderizada somente em navegadores que não suportam o elemento `audio`. Assim, por exemplo, um usuário com necessidade auditiva usando um navegador-padrão não tem acesso à marcação (X)HTML.

Os atributos desse elemento são: atributos globais, `src`, `preload`, `autoplay`, `loop`, `controls`.

O atributo `src` destina-se a indicar o caminho para o arquivo de som a incorporar na página.

O atributo `preload` destina-se a indicar como o autor espera que seja o pré-carregamento do som na página, tendo em vista otimizar a experiência do usuário. Esse atributo admite os valores `none`, `metadata`, `auto`. O valor `none` instrui o navegador a não fazer o pré-carregamento do som e talvez seja uma boa decisão usar esse valor em uma página contendo vários diferentes sons incorporados. O atributo `metadata` instrui o navegador a fazer o pré-carregamento somente dos metadados do som, como dimensões, duração, controles etc. O valor `auto` define que o som será pré-carregado.

Os atributos `autoplay` e `loop` destinam-se respectivamente a iniciar o som automaticamente, tão logo a página seja carregada e a repetir o som indefinidamente. Observe o exemplo a seguir.

```
<audio autoplay loop>
  <source src="som.ogg" type="audio/ogg">
  <source src="som.mp3" type="audio/mpeg">
  <!-- Código (X)HTML para inserção de som.mp3 com Flash -->
  <p>Seu navegador não suporta o elemento audio da HTML5.
  <br>Faça <a href="som.mp3">download de som.mp3</a></p>
</audio>
```



A sintaxe XHTML5 por ser compatível com XML requer que seja declarado o valor dos atributos. Assim, para atributos booleanos, como `autoplay`, `loop` e `controls`, devemos declarar o valor do atributo igual seu nome, ou seja, `autoplay="autoplay"`, `loop="loop"` e `controls="controls"`.



Não confundir atributo booleano com valor booleano. Um atributo booleano é aquele que pode estar ou não presente na tag de abertura de determinado elemento e valor booleano para atributo são os valores `true` e `false`. Assim, é sintaticamente incorreto declarar, por exemplo, `controls="true"`.

A presença do atributo `controls` faz com que o navegador renderize uma barra de controle nativa contendo botões do tipo *play* e *pause* bem como controle de volume. O uso desse atributo é conforme mostrado a seguir.

► HTML

`<audio controls>`

```
<source src="som.ogg" type="audio/ogg">
<source src="som.mp3" type="audio/mpeg">
<source src="som.wav" type="audio/wave">
<p>Seu navegador não suporta o elemento audio da HTML5.
<br>Faça <a href="som.mp3">download de som.mp3</a></p>
</audio>
```



[c3-audio-ex1.html]

Na figura 3.1 mostramos a renderização nativa dos controles de som nos navegadores Opera, Firefox e Safari.



Figura 3.1 – Controles de som.

É possível personalizar os controles nativos de som da HTML5 com uso de JavaScript. A API de som prevê os métodos `play()` e `pause()` e o atributo `volume` que permitem a personalização. Observe o exemplo mostrado a seguir.

▷ JavaScript

```

1.      <script type="text/javascript">
2.      function id(id) {
3.          return document.getElementById(id)
4.      }
5.      function init () {
6.          var container = id('fundo-controles');
7.          var ctr = id('som');
8.          var play = id('play');
9.          var pause = document.getElementById('pause');
10.         var vmais = document.getElementById('vmais');
11.         var vmenos = document.getElementById('vmenos');
12.         if (!Modernizr.audio) {
13.             container.style.display = 'none';
14.         } else {
15.             play.addEventListener('click', function() {
16.                 ctr.play();
17.                 ctr.volume = 1;
18.             }, false);
19.             pause.addEventListener('click', function() {
20.                 ctr.pause();
21.             }, false);
22.             vmais.addEventListener('click', function() {
23.                 ctr.play();
24.                 ctr.volume += 0.1;
25.             }, false);
26.             vmenos.addEventListener('click', function() {
27.                 ctr.play();
28.                 ctr.volume -= 0.1;
29.             }, false);
30.         }
31.     }
32.window.onload = init;
33.</script>
```

Código comentado:

Linha(s)	Descrição
Linha 2 a 4	Cria uma função denominada <code>id()</code> que simplifica a sintaxe para <code>document.getElementById()</code> .
Linhas 5 a 31	Cria uma função denominada <code>init()</code> que contém o script a ser executado depois que a página é carregada. Na linha 32 chama-se essa função no evento <code>onload</code> .

Linha(s)	Descrição (cont.)
Linhas 6 a 11	Cria variáveis para armazenar os elementos do DOM que serão manipulados pelo script.
Linhas 12 a 30	Usa o objeto Modernizr para criar uma condicional (notar que a biblioteca Modernizr foi incorporada à página na seção head do documento).
Linha 13	Se o navegador não oferece suporte para o elemento <code>audio</code> os controles personalizados não serão mostrados na página, pois definimos <code>display: none;</code> para seu container.
Linha 14	Caso o navegador ofereça suporte para o elemento <code>audio</code> os controles personalizados serão mostrados na página e segue o script de personalização.
Linha 15	Usa o método <code>addEventListener()</code> para atrelar o evento <code>click</code> no elemento do DOM que representa o botão play.
Linha 16	Usa o método <code>play()</code> da API de <code>audio</code> para iniciar a reprodução do som.
Linha 17	Usa o atributo <code>volume</code> da API de <code>audio</code> para aumentar o volume do som ao máximo. Essa linha faz com que o som volte ao seu volume máximo caso o usuário acione o botão play após ter baixado o volume e poderá ou não existir dependendo de cada projeto.
Linha 20	Usa o método <code>pause()</code> da API de <code>audio</code> para iniciar pausar a reprodução do som.
Linha 24	Cada clique no botão para aumentar o volume incrementa o volume do som de 0.1. A escala de volume vai de 0 a 1.
Linha 28	Cada clique no botão para diminuir o volume decrementa o volume do som de 0.1. A escala de volume vai de 0 a 1.

► HTML

```
<audio id="som">
  <source src="som.ogg" type="audio/ogg">
  <source src="som.mp3" type="audio/mpeg">
  <source src="som.wav" type="audio/wave">
  <p>Seu navegador não suporta o elemento audio da HTML5.
  <br>Faça <a href="som.mp3">download de som.mp3</a></p>
</audio>
<div id="fundo-controles">
  <input type="button" src="play.png" value="Play" id="play">
  <input type="button" src="pause.png" value="Pause" id="pause">
  <input type="button" src="vmais.png" value="Volume(+)" id="vmais">
  <input type="button" src="vmenos.png" value="Volume(-)" id="vmenos">
</div>
```



[c3-audio-ex2.html]

No exemplo mostrado anteriormente personalizamos os controles com uso de botões nativos criados com `input` do tipo `button`. É possível personalizar com `input` do tipo `image`, criando botões de qualquer forma, cor, tamanho ou efeito. O exemplo mostrado a seguir é idêntico ao anterior, no qual alteramos a marcação HTML e acrescentamos regras de estilo.

► JavaScript

Idêntico ao do exemplo anterior.

► CSS

```
<style>
#fundo-controles {
    width:163px;
    height:43px;
    display:table-cell;
    vertical-align:middle;
    text-align:center;
    background:url(fundo-controles.png) no-repeat;
}
</style>
```

► HTML

```
<audio id="som">
    <source src="som.ogg" type="audio/ogg">
    <source src="som.mp3" type="audio/mpeg">
    <source src="som.wav" type="audio/wave">
    <p>Seu navegador não suporta o elemento audio da HTML5.
    <br>Faça <a href="som.mp3">download de som.mp3</a></p>
</audio>
<div id="fundo-controles">
    <input type="image" src="play.png" alt="Play" id="play">
    <input type="image" src="pause.png" alt="Pause" id="pause">
    <input type="image" src="vmais.png" alt="Volume(+)" id="vmais">
    <input type="image" src="vmenos.png" alt="Volume(-)" id="vmenos">
</div>
```



[c3-audio-ex3.html]



Nos dois últimos exemplos mostrados, se JavaScript estiver desabilitado, o usuário não terá os controles de som-padrão. Por isso, é preferível que se use o atributo `controls` na marcação HTML e se retire esse atributo com JavaScript declarando `document.getElementById('audio')[0].removeAttribute('controls');`

3.2 vídeo

Esse elemento destina-se a incorporar um vídeo ou filme.

Observe o exemplo a seguir que demonstra o uso desse elemento.

```
<video src="video.ogv">  
</video>
```



[c3-video-ex1.html]

Esse exemplo está disponível no site do livro e pode ser visualizado em navegadores que suportam o formato .ogv para vídeos. Ao visualizar o exemplo, como não foram definidos os controles para o vídeo, em plataforma Windows, clique com o botão direito do mouse sobre a imagem do vídeo e no menu de contexto que aparece escolha “Exibir vídeo”.

Inserir vídeo, na prática, como mostrado no exemplo, não é uma boa opção, pois esperar que o usuário abra um menu de contexto para iniciar o vídeo está fora de cogitação. O exemplo simplesmente demonstra a forma mais simples de inserção. É necessário o uso de alguns atributos no elemento `video` antes de publicá-lo para consumo do usuário.

Contudo, é somente isso que precisamos marcar para incorporar um vídeo à página. Simples e direto sem necessidade de plugins, Flash, QuickTime, Windows Media Player e toda a parafernália a ser considerada quando se pretende incorporar vídeo em uma página web.

Basta declarar o caminho para o arquivo de vídeo no atributo `src` do elemento. O vídeo é reproduzido com as funcionalidades nativas do navegador.

Navegadores que não dão suporte ao elemento `video` renderizam normalmente o conteúdo desse elemento. Assim, para servir o vídeo para esses navegadores, definimos a incorporação do vídeo de maneira como fazemos em (X)HTML, por exemplo: com uso de Flash. Observe o exemplo a seguir.

```
<video src="video.ogv">  
  <!-- Código (X)HTML para inserção do vídeo com Flash -->  
</video>
```

Nesse exemplo, navegadores que suportam HTML5 ignoram o código para inserção do vídeo com Flash e os que não suportam executam o código Flash.

Infelizmente existem outros fatores que independem da vontade do grupo de trabalho para a HTML5 em relação à implementação das funcionalidades para vídeo. Os interesses dos fabricantes de softwares não permitiram que houvesse um consenso sobre qual formato de vídeo adotar para ser implementado nativamente. Para melhor entender as consequências de um não consenso, principalmente quanto à maneira verbosa de como criar a marcação para incorporar vídeo, mostraremos a seguir a terminologia e fixaremos alguns conceitos básicos sobre vídeos na web.

Existem dois tipos de arquivos de vídeo: o arquivo para manipulação local por softwares de edição usados na fase de projeto e o arquivo para publicação, que é o resultado do tratamento do arquivo finalizado na fase do projeto. O primeiro é um arquivo em alta resolução e ocupando muitos Kb de memória ao passo que o segundo é um arquivo otimizado para publicação.

A otimização de um arquivo para publicação, a princípio, deve-se basear em três critérios:

- tamanho do arquivo;
- resolução/qualidade;
- compatibilidade com os dispositivos de apresentação.

A otimização de um arquivo-vídeo compreende duas ações: codificar e comprimir. Vamos começar examinando como é feita a compressão do arquivo.

Sabemos que arquivos têm suas extensões (.doc, .html, .xls, .php etc.) e também que é possível identificar o tipo de arquivo pela sua extensão, pois .doc é um arquivo Word, .xls é Excel e assim por diante. Com arquivos de vídeo não é diferente e suas extensões, como .avi, .mov, .mp4, .wmv, .flv etc., informam-nos não somente que se trata de arquivos de vídeo, mas também como foi feita a compressão do arquivo.

Uma extensão indicativa de um arquivo de vídeo é análoga às extensões .zip, .rar, .tar, .GZIP e tantas outras extensões de arquivos comprimidos com as quais estamos bastante familiarizados. Um arquivo .zip pode conter vários arquivos de diferentes tipos. Por exemplo: se “ziparmos” os arquivos de um site, ele provavelmente conterá arquivos do tipo .html, .jpg, .png, .php, .inc, .xml, .ico, .css e muitos outros. Escolhendo o nome “site” para o arquivo “zipado”, o nosso produto final será site.zip, que nada mais é do que um container para diversos arquivos.

Um vídeo é constituído por uma série de informações necessárias para apresentá-lo em determinada mídia. Informações, como trilhas de áudio, stream de

metadados, título, subtítulos, parâmetros de sincronização etc. estão para a pressão do vídeo assim como os diferentes tipos de arquivo estão para uma pressão “zipada”.

Assim, o arquivo *matrix.mp4* é um container de compressão para as informações de vídeo denominado *matrix* e a compressão usada foi *mp4*.

Os containers de compressão relevantes para a HTML5 são:

- Ogg para a extensão *.ogv*.
- MPEG4 para as extensões *.mp4* e *.m4v*.
- WebM para a extensão *.webm*.
- Flash Video para a extensão *.flv*.

Mas isso não é tudo. Além da compressão, os arquivos de vídeo são também codificados. E, para aumentar ainda mais a confusão causada pela falta de padronização, existem diferentes formas de codificação, que, por sua vez, contribuem para mais confusão, uma vez que codificar igual em diferentes containers de compressão pode resultar em resultados diversos.

As diferentes formas de codificação de vídeo são conhecidas por *codecs*, abreviação para as palavras codificação/decodificação.

A existência de diferentes containers e *codecs* para vídeos é resultado das diferentes implementações dos fabricantes de softwares em seus produtos para visualização de vídeos.

A HTML5, em um primeiro momento, tentou unificar o formato para vídeos optando por um software livre; contudo, a Nokia e a Apple não concordaram e não houve consenso.

Os *codecs* relevantes para a HTML5 são:

- H.264 para MPEG
- MPEG4 para *mp4*
- Theora para *ogv*
- VP8 para *WebM*

Para maiores detalhes sobre compressão e codificação de vídeos, consulte os seguintes links:

- <http://www.animenmusicvideos.org/guides/avtech31/>
- <http://pt.wikipedia.org/wiki/H.264>
- <http://pt.wikipedia.org/wiki/H.Theora>
- <http://pt.wikipedia.org/wiki/VP8>
- http://en.wikipedia.org/wiki/Comparison_of_container_formats

A tendência atual é deixar por conta do fabricante a implementação do tipo de compressão e *codecs*. Isso nos obriga a servir marcação para vários formatos se pretendemos criar uma implementação crossbrowser.

Felizmente o elemento `video` admite como elemento-filho o elemento `source` cuja finalidade é substituir o atributo `src` e permitir formatos alternativos para o vídeo a ser servido.

O elemento `source` destina-se a permitir que se definam arquivos diferentes para incorporar as mídias `audio` e `video` em uma página.

O elemento `source` admite os seguintes atributos: atributos globais, `src`, `type`, e `media`.

O atributo `src` aponta para o endereço do arquivo de mídia a incorporar na página. É um atributo obrigatório cujo valor é um URL não vazio.

O atributo `type` destina-se a informar ao navegador o tipo de mídia a ser incorporado de modo que ele possa decidir o que fazer antes de baixar o arquivo. O valor desse atributo deve ser um *MIME type* válido.

O atributo `type` admite ainda o parâmetro `codecs` que determinados *MIME type* definem e deve ser informado para que o navegador saiba como o arquivo foi codificado.

O atributo `media` destina-se a informar ao navegador o tipo de mídia. Os valores possíveis para esse atributo são aqueles listados nas especificações para Media Queries (<http://dev.w3.org/csswg/css3-mediaqueries/>) e o valor-padrão é `all`.

Observe o exemplo a seguir que esclarece o uso do elemento `source` e seus atributos.

```
<video>
  <source src="video.ogv" type="video/ogg; codecs='theora, vorbis'" media="screen">
  <source src="video.mp4" type="video/mp4; codecs='mp4v.20.8, mp4a.40.2'" media="screen">
  <source src="video.webm" type="video/webm">
    <!-- Código (X)HTML para inserção de video.mp4 com Flash -->
</video>
```

Notar que o parâmetro `codecs` do atributo `type` especifica a codificação tanto do vídeo quanto do áudio.

Nesse exemplo escrevemos um código crossbrowser para servir vídeo com HTML5.

Alternativamente podemos inserir marcação (X)HTML no elemento `video` com a finalidade de servir navegadores que não suportam qualquer marcação para inserção de som, como mostrado a seguir.

```
<video>
  <source src="video.ogv" type="video/ogg">
  <source src=" video.mp4" type=" video/mp4">
  <source src="video.webm" type="video/webm">
  <!-- Código (X)HTML para inserção de vídeo com Flash -->
  <p>Seu navegador não suporta o elemento video da HTML5.
  <br>Faça <a href="video.mp4">download do vídeo</a></p>
</video>
```

Mas, cuidado. Marcação (X)HTML inserida no elemento `video` como mostrado anteriormente não cumpre finalidade de oferecer alternativa textual para o vídeo, pois ela é renderizada somente em navegadores que não suportam o elemento `video`. Assim, por exemplo, um usuário com necessidade visual usando um navegador-padrão não tem acesso à marcação (X)HTML.

Os atributos desse elemento são: atributos globais, `src`, `preload`, `autoplay`, `loop`, `controls`, `poster`, `audio`, `width` e `height`.

O atributo `src` destina-se a indicar o caminho para o arquivo de vídeo a incorporar na página.

O atributo `preload` destina-se a indicar como o autor espera que seja o pré-carregamento do vídeo na página, tendo em vista otimizar a experiência do usuário. Esse atributo admite os valores `none`, `metadata`, `auto`. O valor `none` instrui o navegador a não fazer o pré-carregamento do vídeo e talvez seja uma boa decisão usar esse valor em uma página contendo diferentes vídeos incorporados. O atributo `metadata` instrui o navegador a fazer o pré-carregamento somente dos metadados do vídeo, como dimensões, duração, controles etc. O valor `auto` define que o vídeo será pré-carregado.

Os atributos `autoplay` e `loop` destinam-se respectivamente a iniciar o vídeo automaticamente, tão logo a página seja carregada e a repetir o vídeo indefinidamente. Observe o exemplo a seguir.

```
<video autoplay loop>
  <source src="video.ogv" type="video/ogg">
```

```
<source src="video.mp4" type="video/mp4">
<source src="video.webm" type="video/webm">
<!-- Código (X)HTML para inserção de video com Flash -->
<p>Seu navegador não suporta o elemento video da HTML5.
<br>Faça <a href="video.mp4">download do video</a></p>
</video>
```



[c3-video-ex2.html]



A sintaxe XHTML5 por ser compatível com XML requer que seja declarado o valor dos atributos. Assim, para atributos booleanos, como `autoplay`, `loop` e `controls`, devemos declarar o valor do atributo igual seu nome, ou seja, `autoplay="autoplay"`, `loop="loop"` e `controls="controls"`.



Não confundir atributo booleano com valor booleano. Um atributo booleano é aquele que pode estar ou não presente na tag de abertura de determinado elemento e valor booleano são os valores `true` e `false`. Assim, é sintaticamente incorreto declarar, por exemplo, `controls="true"`.

A presença do atributo `controls` faz com que o navegador renderize uma barra de controle nativa contendo botões do tipo *play* e *pause* bem como controle de volume. O uso desse atributo é conforme mostrado a seguir.

► HTML

<video controls>

```
<source src="video.ogv" type="video/ogg">
<source src="video.mp4" type="video/mp4">
<source src="video.webm" type="video/webm">
<!-- Código (X)HTML para inserção de video com Flash -->
<p>Seu navegador não suporta o elemento video da HTML5.
<br>Faça <a href="video.mp4">download do video</a></p>
</video>
```



[c3-video-ex3.html]

Na figura 3.2 mostramos a renderização nativa dos controles de som nos navegadores Chrome, Firefox, Opera e Safari.

O atributo `poster` destina-se a definir uma imagem a ser renderizada inicialmente e enquanto as informações sobre o vídeo não estiverem totalmente carregadas ou o usuário não acionar o início do vídeo. Essa imagem funciona como uma “capa” de um livro e normalmente contém o título do vídeo e transmite visualmente informações sobre o vídeo. Quando esse atributo não é definido, a renderização inicial apresenta o primeiro quadro do vídeo. O valor desse atributo deve ser o URL de uma imagem.

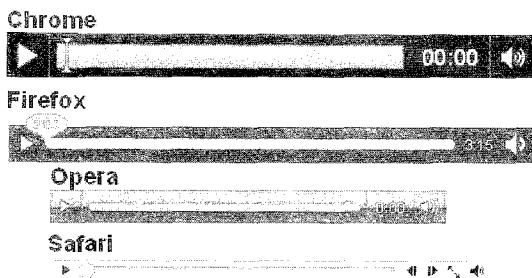


Figura 3.2 – Controles de vídeo.

O uso desse atributo é conforme mostrado a seguir.

► HTML

```
<video poster="capa.jpg" controls>
  <source src="video.ogv" type="video/ogg">
  <source src="video.mp4" type="video/mp4">
  <source src="video.webm" type="video/webm">
  <!-- Código (X)HTML para inserção de vídeo com Flash -->
  <p>Seu navegador não suporta o elemento video da HTML5.
  <br>Faça <a href="video.mp4">download do vídeo</a></p>
</video>
```



[c3-video-ex4.html]

O atributo audio admite o valor muted e, quando presente, faz com que o som do vídeo (se o vídeo for sonoro) seja por padrão colocado no estado mudo. Há estudos para a introdução de outros valores para esse atributo nas especificações para a HTML5. Valores para definir, por exemplo, os padrões para o volume e trilha sonora iniciais. O uso desse atributo é conforme mostrado a seguir.

► HTML

```
<video poster="capa.jpg" controls muted>
  <source src="video.ogv" type="video/ogg">
  <source src="video.mp4" type="video/mp4">
  <source src="video.webm" type="video/webm">
  <!-- Código (X)HTML para inserção de vídeo com Flash -->
  <p>Seu navegador não suporta o elemento video da HTML5.
  <br>Faça <a href="video.mp4">download do vídeo</a></p>
</video>
```



[c3-video-ex5.html]

Nos exemplos mostrados omitimos o parâmetro `codecs` do atributo `type`. Essa não é uma boa prática, pois ao especificarmos vários formatos de vídeo com uso do elemento `source` o navegador percorre os formatos na ordem em que eles aparecem no código e ao encontrar o primeiro formato que suporta inicia o carregamento do arquivo. Com o objetivo de evitar que o navegador tenha de verificar o suporte examinando o formato do vídeo, o que leva mais tempo, devemos sempre usar o atributo `type` e seu parâmetro `codecs`, pois para o navegador verificar o suporte ao MIME `type` é uma operação muito mais rápida do que verificar o suporte ao arquivo.

Os atributos `width` e `height` definem, respectivamente, a largura e altura (dimensões) da área de apresentação do vídeo. Se esses atributos não estiverem presentes, as dimensões serão iguais àquelas para as quais o vídeo foi codificado, ou seja, as dimensões-padrão do vídeo.

Quando um vídeo é criado, definem-se suas dimensões e com a definição das dimensões fica implícito o `aspect ratio` que é a razão entre a largura e altura. Caso não sejam definidas as dimensões do vídeo, o navegador usa as dimensões intrínsecas do vídeo e finalmente se estas não estiverem disponíveis a largura adotada é 300px.

Se definirmos apenas uma das dimensões, a outra será tal que o `aspect ratio` seja preservado. Se definirmos as duas dimensões sem respeitar o `aspect ratio` a área de apresentação será a definida; contudo, o vídeo não sofrerá deformação, pois suas dimensões serão tais que o `aspect ratio` seja preservado e teremos a área definida maior do que a área do vídeo. O exemplo a seguir disponível no site do livro é interativo, permitindo que você altere os atributos `width` e `height` e observe a renderização das áreas de apresentação.

▷ HTML

```
<video controls width="escolha" height="escolha">
    <source src="video.ogv" type="video/ogg">
    <source src="video.mp4" type="video/mp4">
    <source src="video.webm" type="video/webm">
    <!-- Código (X)HTML para inserção de video com Flash -->
    <p>Seu navegador não suporta o elemento video da HTML5.
    <br>Faça <a href="video.mp4">download do video</a></p>
</video>
```



[c3-video-ex6.html]

Para ver o script da interatividade, consulte o código-fonte do exemplo.

3.2.1 track

Esse elemento destina-se a definir trilhas externas para os elementos `video` e `audio`. Com eles o desenvolvedor pode personalizar trilhas que são gravadas em arquivos externos e inseri-las no vídeo. Os atributos específicos desse elemento definem as características da trilha e são: `kind`, `src`, `charset`, `srclang` e `label`. O elemento `track` admite ainda os atributos globais.

O atributo `kind` destina-se a definir o tipo de trilha a ser inserida e os valores possíveis são: `subtitles`, `captions`, `descriptions`, `chapters` e `metadata` que se destinam respectivamente a inserir subtítulos, legendas, descrições, capítulos e metadados.

O atributo `src` aponta para um arquivo do tipo texto puro contendo basicamente os textos a serem inseridos e o tempo (momento) de inserção.

No estágio atual das especificações ainda não há um consenso sobre o formato de arquivo a adotar entre os mais de cinquenta disponíveis para essa tarefa. Contudo, a especificação para a HTML5 prevê o formato *WebSRT* com a extensão `.srt` (`exemplo.srt`).

Observe no exemplo a seguir um arquivo nesse formato.

```
1 // numeração, sequencial
00:00:00,000 --> 00:00:05,000 // intervalo de inserção h:m:s:ms
0 livro HTML5 do Maujor apresenta: // texto a inserir
2
00:00:05,010 --> 00:00:12,000
0 elemento track para inserção de legendas nas mídias audio e video.
3
00:00:12,010 --> 00:00:14,800
Exemplo de arquivo WebSRT.
4
00:00:26,100 --> 00:00:28,206
0 elemento track é fantástico para a acessibilidade.
```

Um arquivo *WebSRT* tem o formato e sintaxe conforme mostrado no exemplo e é constituído de grupamentos de três linhas. A primeira linha contém uma numeração sequencial; a segunda, o intervalo de inserção com o instante de início e fim separados pelo sinal `-->` e, na terceira linha, o texto a inserir.

Arquivos *WebSRT* devem ser codificados por padrão em *UTF-8*. O atributo `charset` foi criado por razões históricas e, quando presente, destina-se a definir uma codificação diferente.

O atributo `srclang` destina-se a definir o idioma em que foram escritos os textos do arquivo `WebSRT`. O valor desse atributo é uma string com o código do idioma, como `pt-br`, `en`, `de` etc. Esse atributo é de uso obrigatório quando o valor do atributo `kind` for `subtitles`.

O atributo `label` destina-se a definir um rótulo para a trilha a ser inserida.

O exemplo a seguir esclarece o uso desse elemento.

```
<video autoplay controls>
  <source src="video.ogv" type="video/ogg">
  <source src="video.mp4" type="video/mp4">
  <source src="video.webm" type="video/webm">
  <!-- Código (X)HTML para inserção de video com Flash -->
  <p>Seu navegador não suporta o elemento video da HTML5.
  <br>Faça <a href="video.mp4">download do video</a></p>
  <track kind="subtitles" label="Inglês" srclang="en" src="legendas_en.srt"></track>
  <track kind="subtitles" label="Português" srclang="pt-br" src="legendas_pt-br.srt">
</track>
</video>
```

Nesse exemplo criamos subtítulos (legendas) para o vídeo em inglês e português. A escolha do idioma pelo usuário se faz com uso de script, não mostrado no exemplo.

Atualmente nenhum navegador suporta esse elemento; contudo, você pode ter uma ideia de como se pretende que essa facilidade funcione visitando a página http://dev.mennerich.name/showroom/html5_video/. Nela você verá um exemplo que usa um player de vídeo denominado LeanBack Player, desenvolvido em JavaScript e se destina a inserir legendas em vídeos.

Outra página que demonstra a inserção de legendas está em <http://people.opera.com/brucel/demo/video/multilingual-synergy.html>; nessa há opção de escolha dos idiomas inglês e japonês para as legendas.

Para maiores detalhes sobre arquivos `WebSRT` visite <http://www.delphiki.com/websrt/>.



A especificação para a HTML5 prevê ainda o formato WebVTT para arquivos externos de trilhas.

3.2.2 Codificação de vídeos

É muito provável que o equipamento que você usa para criar seus vídeos gere arquivos em formatos para edição, o que significa arquivos de qualidade superior e ocupando muitos MB de memória, ou ainda, arquivos que necessitam de tratamento e compressão para transformá-los em formatos destinados à publicação.

Assim, é improvável que você extraia da sua filmadora arquivos nas extensões .*avi*, .*mp4* e .*webm* prontos para publicar na sua página HTML5. Existem softwares especializados em edição que fazem a conversão dos arquivos; contudo, você não precisa instalar um desses sofisticados softwares somente para converter arquivos de vídeo, pois existem várias ferramentas de conversão gratuitas para download. Mostraremos a seguir as ferramentas Firefogg para codificar vídeos em formato Ogg e WebM e Handbrake para formatos MPEG4. Ambas as ferramentas são licenciadas sobre a bandeira do software livre, o que significa que você pode baixar e usar gratuitamente.

As ferramentas serão mostradas nos itens 3.2.2.1 e 3.2.2.2, nos quais fizemos uso de várias imagens para explicar passo a passo como formatar (comprimir) arquivos de vídeo. A finalidade é simplesmente apresentar as ferramentas para aqueles que não estão familiarizados com edição e gravação de vídeos. Assim, mostramos somente o uso básico das ferramentas sem entrar em detalhes avançados. Se você optar pelo uso dessas ferramentas, explore-as mais a fundo fazendo testes práticos e inspecionando todas as opções que elas oferecem. Contudo, com o conhecimento aqui adquirido, você será capaz de formatar seus vídeos de modo que eles possam ser inseridos em páginas HTML5 com uso do elemento *video*. Se você é um expert em edição de vídeo, sinta-se à vontade para pular os itens 3.2.2.1 e 3.2.2.2.

3.2.2.1 Firefogg

Firefogg é uma extensão para o navegador Firefox nas suas versões 3.5 e mais recentes. É uma ferramenta de código aberto e licença General Public License (GPL), destinada a codificar vídeos nos containers Ogg e WebM com os codecs Theora (vídeo) Vorbis (áudio) e VP8 (vídeo-áudio), respectivamente, formatos suportados nativamente pelos navegadores Firefox e Chrome.

Visite o site do Firefogg em <http://www.firefogg.org> e você encontrará um link para instalação da extensão no Firefox (Figura 3.3).

Clique o link para instalar e o Firefox apresentará uma mensagem solicitando que você permita a instalação da extensão. Clique o botão **Permitir** (Figura 3.4).

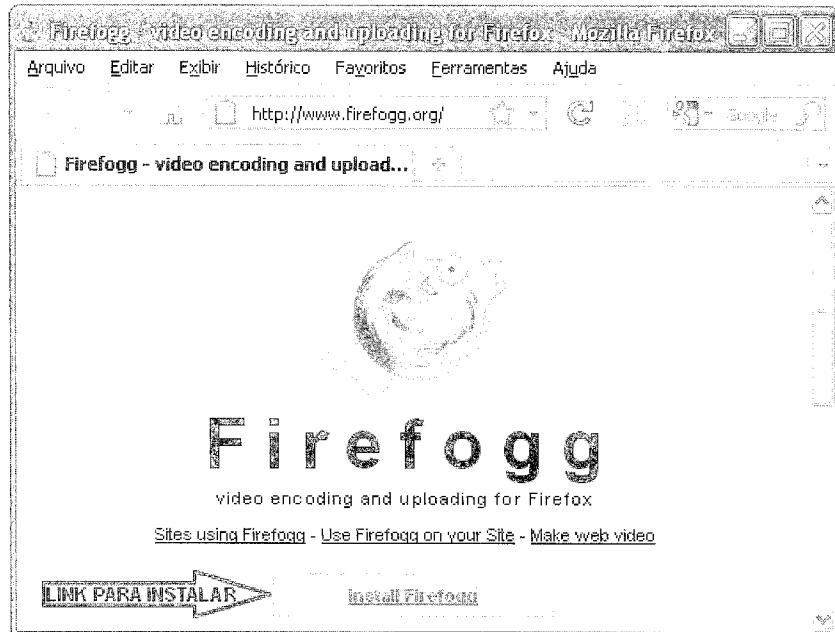


Figura 3.3 – Site do Firefogg.



Figura 3.4 – Permitir instalação.

Clicando o botão, abre-se uma caixa de diálogo-padrão para instalação de extensões e temas no Firefox. Clique o botão **Instalar agora** para iniciar a instalação (Figura 3.5).

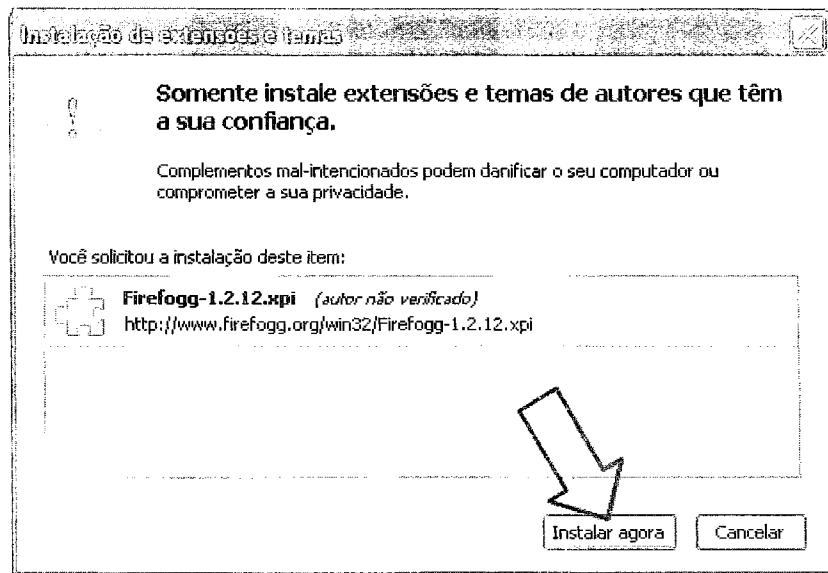


Figura 3.5 – Iniciar instalação.

Terminada a instalação, o Firefox solicitará que você reinicie o navegador para que se complete o processo de instalação (Figura 3.6).

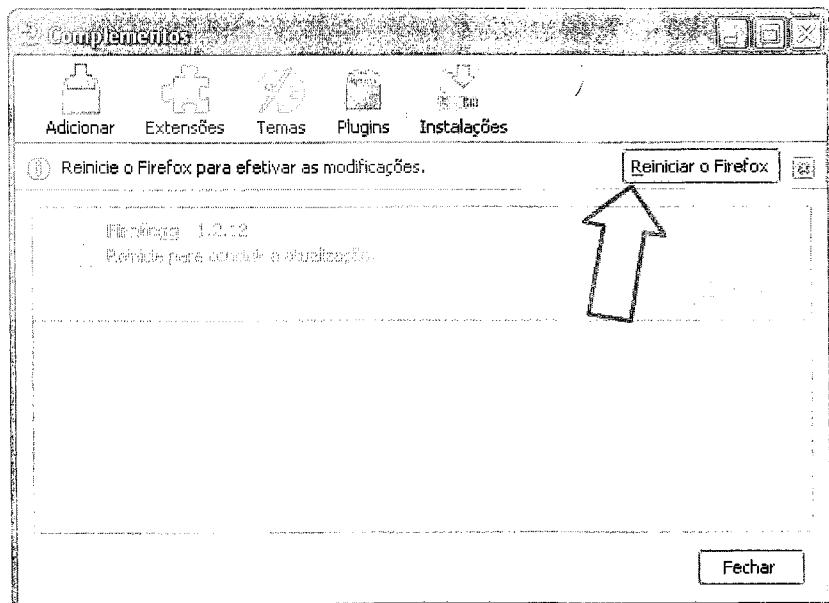


Figura 3.6 – Reiniciar o Firefox.

O Firefox é reiniciado e apresenta uma tela informando que o Firefogg está instalado (Figura 3.7).

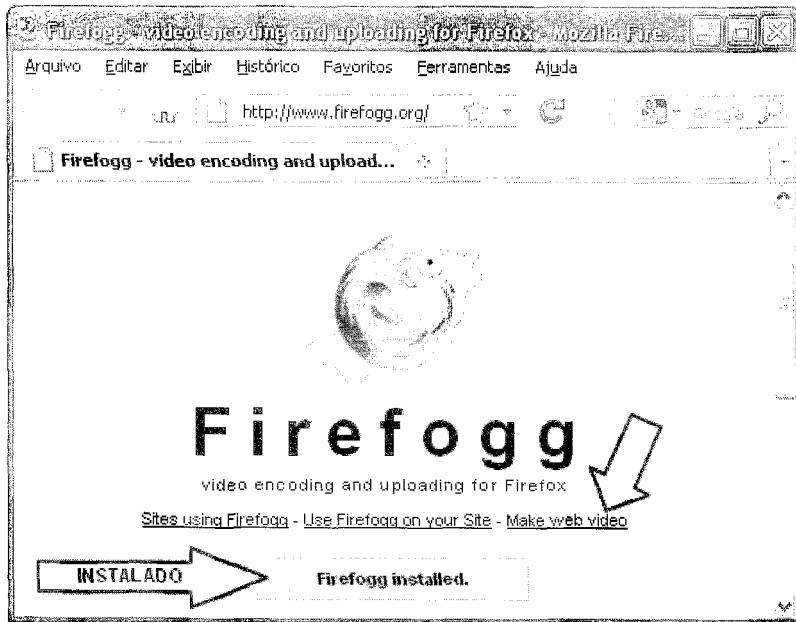


Figura 3.7 – Firefogg instalado.

Feche a tela inicial e abra o menu **Ferramentas** na barra superior do Firefox acionando o submenu **Make Web Video** (Figura 3.8).

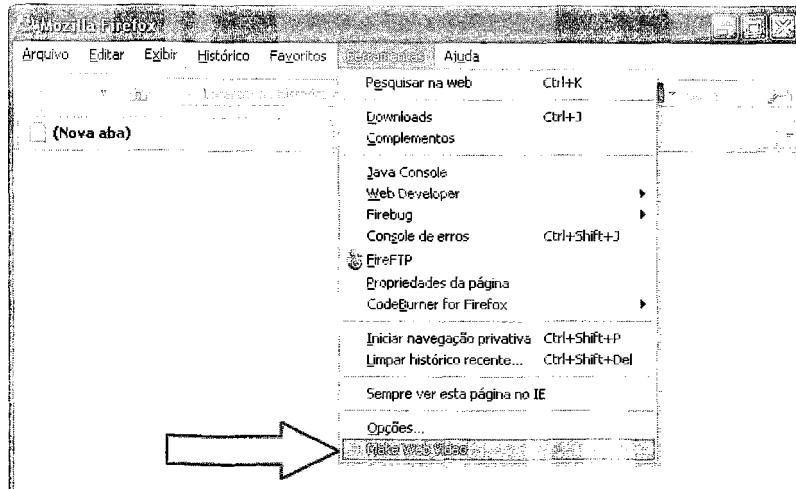


Figura 3.8 – Abrir o Firefogg.

Abre-se a tela inicial do Firefogg apresentando ao usuário a opção de escolha do idioma que mais lhe agrada (Figura 3.9).



Figura 3.9 – Escolha do idioma no Firefogg.

Nessa mesma tela existe um botão que ao ser clicado permite que você escolha no seu disco local um arquivo de vídeo a ser comprimido. Clique o botão, faça sua escolha e será apresentada a tela seguinte, conforme mostrado na figura 3.10. Essa tela permite que você configure os parâmetros de compressão e escolha o formato do arquivo comprimido. Notar que, no campo indicado na figura com o número 1, aparece o arquivo que você escolheu para compressão.

O botão destacado na figura com o número 2 quando clicado inicia o processo de compressão. Em um primeiro momento, experimente simplesmente escolher o arquivo e clicar o botão de compressão, para ter um contato inicial com o funcionamento da ferramenta.

Vamos analisar a seguir os seis itens do menu de configurações para compressão de arquivos. O menu é do tipo acordeão e o primeiro item é mostrado na figura 3.11. Esse item do menu permite escolher entre os containers Theora e WebM. Para Theora há três opções de compressão; para WebM, duas opções. Escolha a compressão que mais se adequa ao seu projeto. Se você não tem certeza, faça experiências com as diferentes compressões.

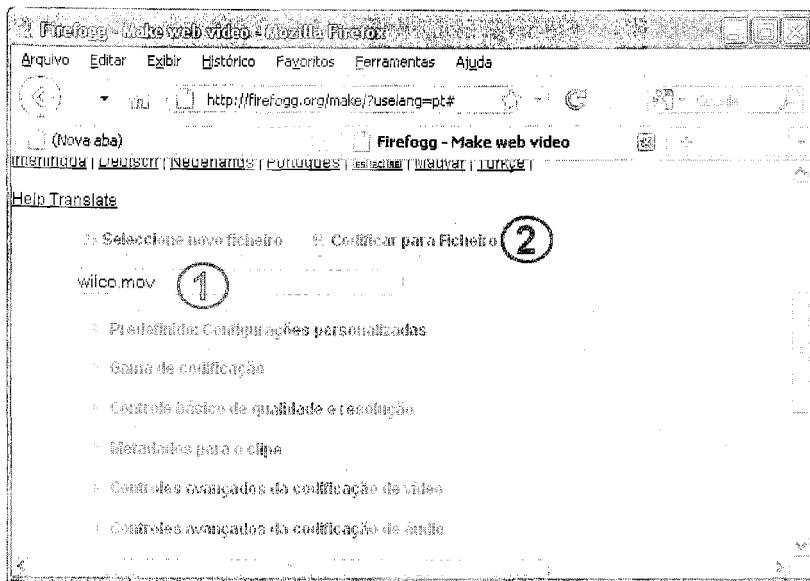


Figura 3.10 – Menu de configurações do Firefogg.

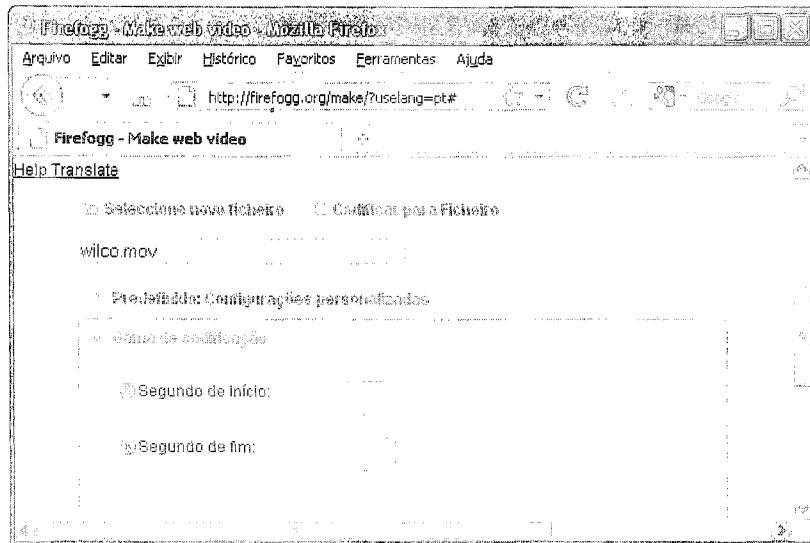


Figura 3.11 – Menu de configurações do Firefogg – item 1.

O segundo item do menu permite que você defina o instante de início e fim da compressão do vídeo expressos em segundos. Por exemplo: para um vídeo com 30 segundos de duração, escolha 0 (zero) e 30 respectivamente para compressão do vídeo todo, ou 15 e 30 se você pretende que a compressão (e consequentemente o resultado final) mostre o vídeo a partir da metade. A figura 3.12 mostra esse item do menu.

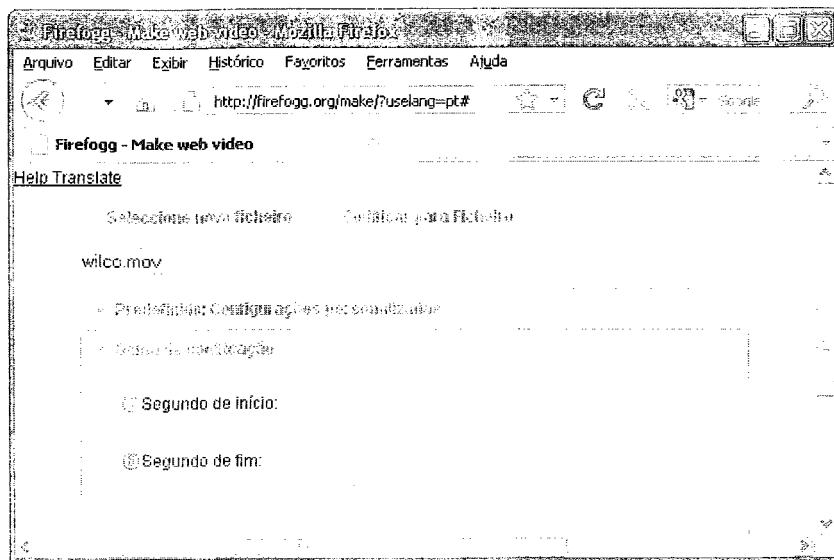


Figura 3.12 – Menu de configurações do Firefogg – item 2.

O terceiro item do menu permite definir os parâmetros de qualidade e resolução e escolher o container de compressão, conforme mostrado na figura 3.13. Nesse menu você poderá escolher somente os codecs de vídeo e áudio e deixar os demais campos com seus valores default.

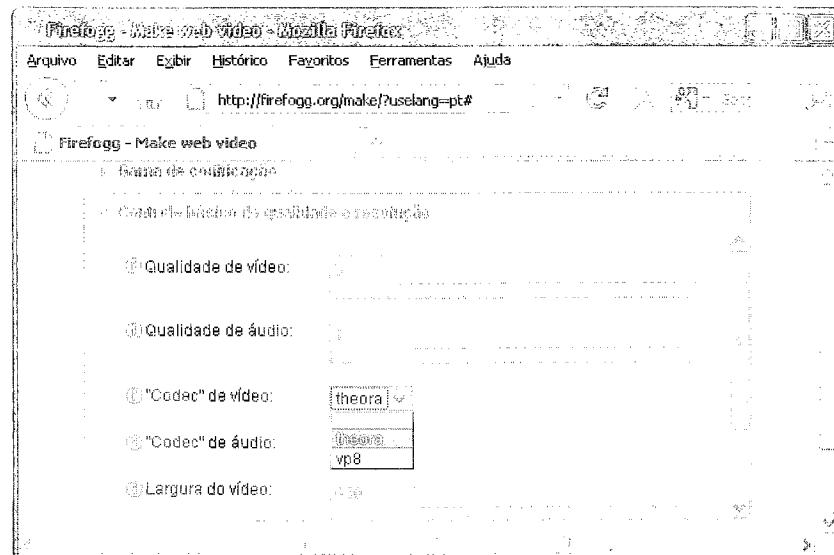


Figura 3.13 – Menu de configurações do Firefogg – item 3.

O quarto item do menu permite definir os metadados para o vídeo. Os metadados a definir devem ser escritos em campos de texto e são: título, nome do criador do vídeo, data e local de criação, nome da organização e direitos autorais.

O quinto item do menu permite configurações avançadas para o vídeo e pode ser deixado com suas configurações default. Adicionalmente existe um pequeno ícone azul com a letra i que, ao se colocar o mouse sobre ele, fornece informações sobre cada parâmetro a configurar.

O sexto item do menu permite que você escolha não incluir na compressão o áudio do vídeo. Para isso marque a caixa checkbox do menu.

Uma vez definidos os parâmetros de configuração, é hora de iniciar a compressão do vídeo. Para tanto pressione o botão **Codificar para arquivo** mostrado em destaque na figura 3.10. Abre-se uma janela de diálogo solicitando que você escolha um local no seu disco local para armazenar o arquivo no formato escolhido. Faça a escolha e dê continuidade à compressão pressionando o botão correspondente da caixa de diálogo aberta. Com isso o processo de compressão é iniciado e uma barra de progresso informa o andamento da tarefa ao mesmo tempo que uma visão prévia do vídeo é apresentada em um thumbnail abaixo da barra, conforme mostrado na figura 3.14.

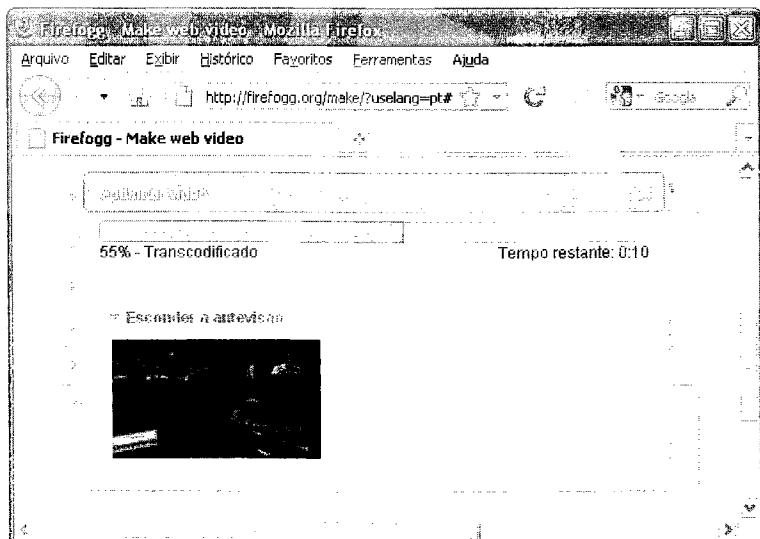


Figura 3.14 – Compressão em andamento no Firefogg.

Terminada a compressão, você poderá rodar e visualizar o vídeo no próprio thumbnail, iniciar nova compressão, ou verificar o resultado final armazenado no seu disco local.

3.2.2.2 HandBrake

HandBrake é uma ferramenta para compressão de vídeos disponível para download em <http://handbrake.fr/downloads.php>. É uma ferramenta de código aberto e licença General Public License (GPL), destinada a codificar vídeos no container MPEG4 codecs H.264 para vídeo e AAC para áudio, formato suportado nativamente pelo navegador Safari, por Flash da Adobe, iPhone e Google Android. A Microsoft já anunciou que o navegador Internet Explorer 9 virá com suporte para esse formato.

Visite a área de download do site da HandBrake, conforme mostrado na figura 3.15, no endereço <http://handbrake.fr/downloads.php>, faça o download e instale a ferramenta em seu computador.

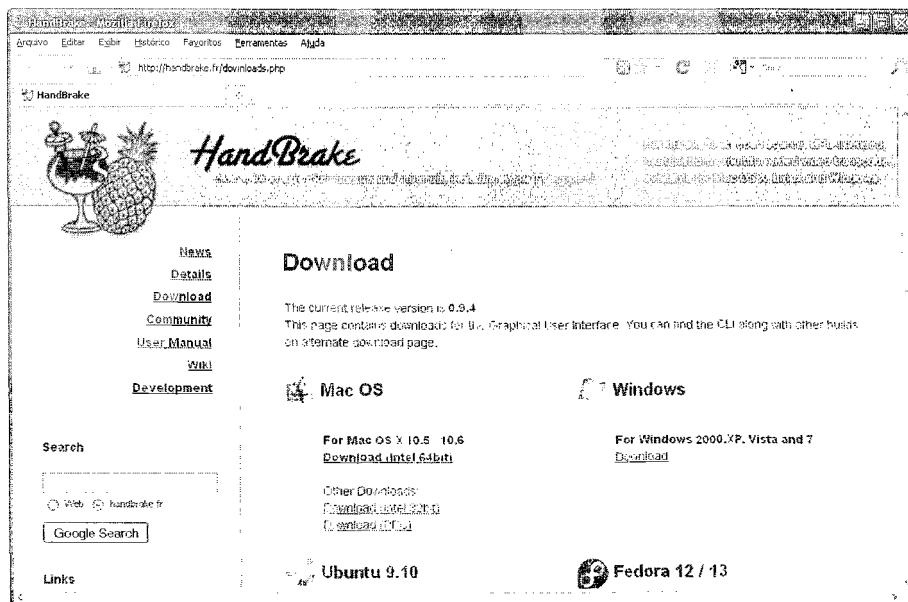


Figura 3.15 – HandBrake – área de download.

Abra a ferramenta recém-instalada no seu computador. A tela inicial é conforme a figura 3.16. Nessa tela abra o menu **Source** e clique a opção **Video File** para selecionar o arquivo de vídeo a formatar. Notar que a opção MP4 para o container já vem selecionada por padrão.

Clicando a opção **Vídeo File** abre-se a janela de diálogo para escolha do arquivo. Escolha o arquivo no seu disco rígido e prossiga.

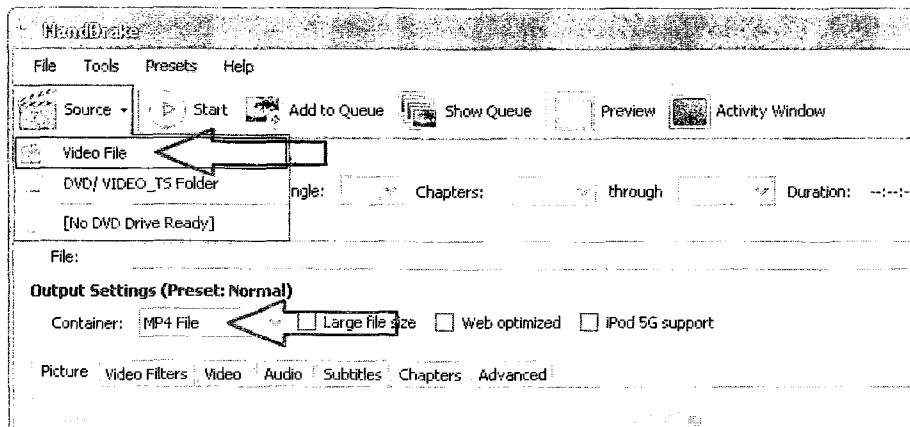


Figura 3.16 – HandBrake – tela inicial (parte).

Escolhido o arquivo, a ferramenta após alguns segundos apresentará uma janela de alerta, conforme mostrado na figura 3.17. O alerta convida o usuário a definir uma pasta-padrão no seu disco rígido para salvar os arquivos formatados. Clique OK e ignore o alerta se você preferir não designar uma pasta-padrão para salvar os arquivos.

Caso você prefira a pasta-padrão, configure-a no menu da ferramenta em **Tools > Options > aba General > Default Path**.

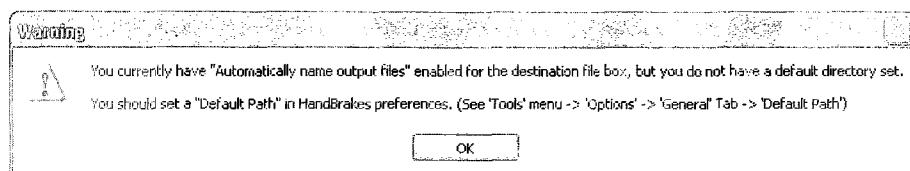


Figura 3.17 – HandBrake – alerta para definir diretório-padrão.

O próximo passo é configurar as opções de compressão do arquivo. Observe na figura 3.18 as configurações básicas da aba **Picture** descritas a seguir de acordo com a numeração destacada na figura.

1. No menu à direita da interface, escolha compressão para iPhone & iPod Touch, que é a opção que nos dará maior flexibilidade de escolha.

2. Clique o botão **Browse** e escolha no seu disco rígido um destino e o nome do arquivo formatado. A extensão-padrão é **.m4v** que é usada para iPhone e iPod e similar à extensão **.mp4**.
3. Marque o checkbox para a opção **Web optimized**.
4. A escolha default do container é **MP4 File**.
5. Marque o checkbox **Keep Aspect Ratio** para garantir que a compressão não distorcerá seu vídeo e escolha uma largura ou altura para o vídeo. Escolhendo uma delas a outra será automaticamente escolhida para manter o “aspect ratio”.

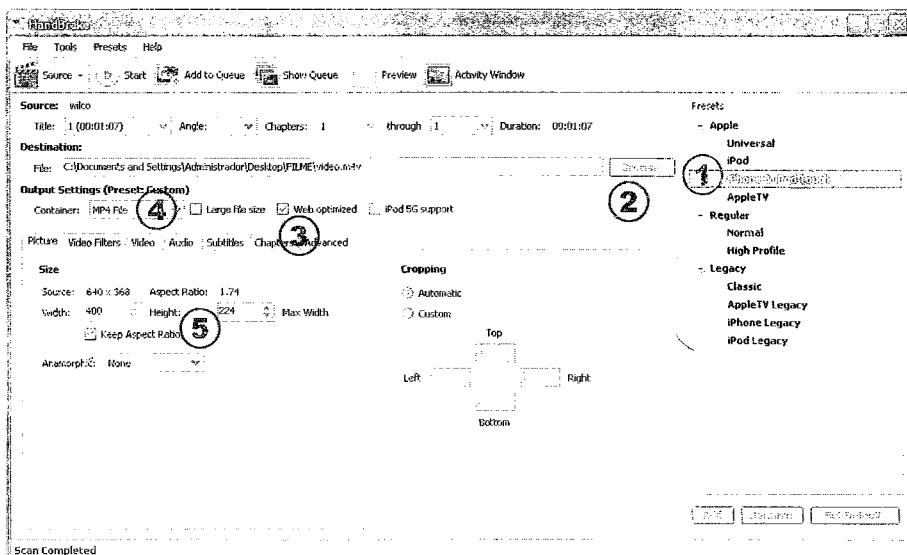


Figura 3.18 – HandBrake – configurações aba Picture.

Passe para a aba **Vídeo** e configure como mostrado na figura 3.19. Notar que existem três opções para a configuração da qualidade do vídeo. Na primeira opção, **Target Size (MB)**, você define um tamanho máximo para seu arquivo formatado. A não ser que você saiba exatamente o que está fazendo, é melhor não optar por essa forma de definição da qualidade.

A segunda opção, **Avg Bitrate (kbps)**, permite que você defina a qualidade do vídeo escolhendo uma taxa média de kilobytes por segundo para a compressão. Aqui também, a não ser que você saiba exatamente o que está fazendo, é melhor não optar por essa forma de definição da qualidade.

A terceira opção permite que se defina a qualidade em porcentagem. O valor default é 60,78% e você pode adotar esse valor.

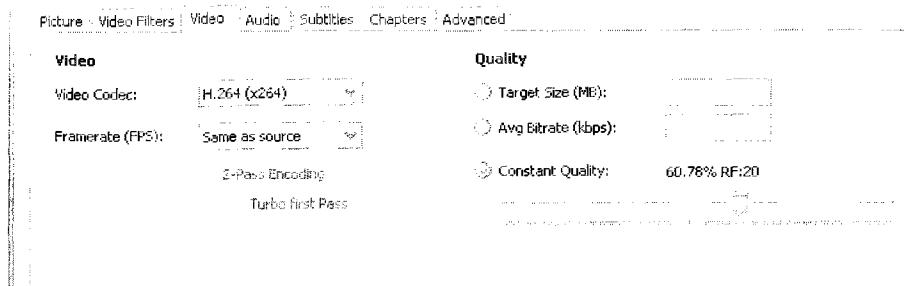


Figura 3.19 – HandBrake – configurações aba Video.

As demais abas de configuração podem ser deixadas como estão. Finalmente clique o botão **Start** no menu da ferramenta, conforme mostrado na figura 3.20, e o processo de compressão iniciará.

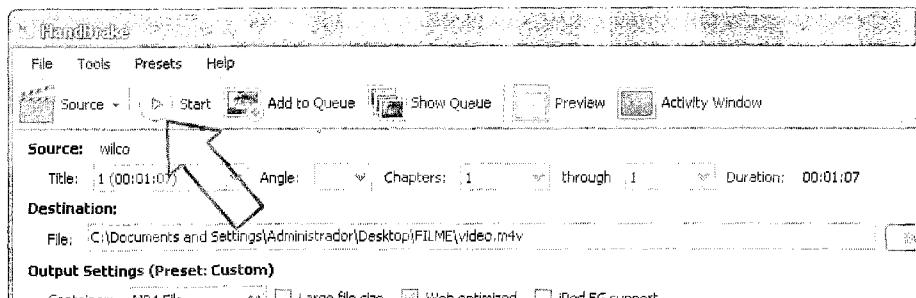


Figura 3.20 – HandBrake – iniciar compressão.

Durante a compressão, será apresentada uma janela pop-up, como mostrada na figura 3.21, informando o andamento da compressão. Terminada a compressão, a janela desaparece e o arquivo formatado estará na pasta designada no início do processo de compressão.

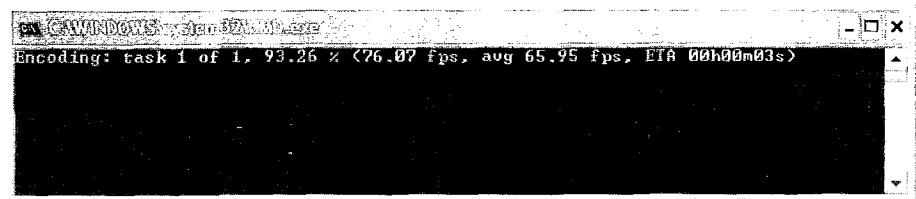


Figura 3.21 – HandBrake – comprimindo.

3.2.2.3 Outras ferramentas

Além das duas ferramentas mostradas anteriormente, existem centenas de outras na Internet e a escolha de uma delas cabe somente ao desenvolvedor. Apresentamos a seguir, a título de informação, uma lista de algumas delas.

- Media-Converter – <http://media-convert.com/conversor/> (uso online sem necessidade de download)
- Super – <http://www.erightssoft.com/SUPER.html> (plataforma Windows)
- Format Factory – <http://www.formatoz.com/> (plataforma Windows)
- Media Coder – <http://mediacoder.sourceforge.net/> (todas as plataformas)
- FFmpeg – <http://ffmpeg.org/> (todas as plataformas)
- Switch – <http://www.nch.com.au/switch/> (Windows e Macintosh)

3.3 API para áudio e vídeo

Os atributos, métodos e eventos previstos na API para as mídias de `audio` e `video` da HTML5 são os mesmos. No item 3.1 mostramos dois scripts de personalização dos controles de `audio` que fizeram uso dos métodos `play()` e `pause()` e do atributo `volume` previstos na API das mídias. Esses métodos e atributos e os demais métodos e atributos da API são comuns aos elementos `audio` e `video` e constantes da API para elementos de mídia da HTML5. A única diferença é que o elemento `video` possui os atributos `poster`, `width` e `height` não previstos para o elemento `audio`.

Assim, é possível personalizar os controles de `video` tal como fizemos com os controles de `audio`. O exemplo a seguir usa o mesmo script do exemplo [c3-audio-ex3.html] no qual mostramos a personalização dos controles para `audio`. Naquele exemplo fizemos menção à necessidade de se incluir o atributo `controls` na marcação e retirá-lo com script para garantir sua presença caso JavaScript esteja desabilitado no navegador. Nesse exemplo mostramos o código para essa técnica, incluindo os controles e retirando-os.

Observe o exemplo para personalização dos controles de `video` com uso de JavaScript.

► JavaScript

```

1.      <script type="text/javascript">
2.      function id(id) {
3.          return document.getElementById(id);
4.      }
5.      function init () {
6.          var container = id('fundo-controles');
7.          var ctr = id('video');
8.          var play = id('play');
9.          var pause = document.getElementById('pause');
10.         ctr.removeAttribute('controls'); // retira os controles nativos
11.         var vmais = document.getElementById('vmais');
12.         var vmenos = document.getElementById('vmenos');
13.         if (!Modernizr.video) {
14.             container.style.display = 'none';
15.         } else {
16.             // código idêntico ao do arquivo c3-audio-ex3.html
17.         }
18.     </script>

```

► HTML

```

<video poster="capa.jpg" controls id="video">
    <source src="video.ogv" type="video/ogg">
    <source src="video.mp4" type="video/mp4">
    <source src="video.mp4" type="video/mp4">
    <!-- Código (X)HTML para inserção de vídeo com Flash -->
    <p>Seu navegador não suporta o elemento video da HTML5.
    <br>Faça <a href="video.mp4">download do vídeo</a></p>
</video>
<div id="fundo-controles">
    <input type="button" src="playv.png" value="Play" id="play">
    <input type="button" src="pausev.png" value="Pause" id="pause">
    <input type="button" src="vmais.png" value="Volume(+)" id="vmais">
    <input type="button" src="vmenos.png" value="Volume(-)" id="vmenos">
</div>

```



[c3-video-ex7.html]

Não é comum controles de vídeo apresentarem ao mesmo tempo dois botões para play e pause. O que se encontra geralmente é uma troca dos botões de acordo com o estado do vídeo. Se o vídeo está parado, o usuário vê um botão play; se está rodando, um botão pause. Pode-se conseguir esse efeito com script, como veremos adiante.

3.3.1 Atributos

Vamos examinar a seguir os atributos constantes da API para as mídias de áudio e vídeo. Os exemplos mostrados nesse item foram criados para a mídia video, mas, na sua maioria, são aplicáveis à mídia audio.

error

Esse atributo retorna um objeto erro para a mídia e possui a propriedade code que retorna um número código para o erro com o significado mostrado a seguir:

- MEDIA_ERR_ABORTED (1) – O usuário abortou a exibição do vídeo.
- MEDIA_ERR_NETWORK (2) – Um erro de conexão causou a interrupção ou não exibição do vídeo.
- MEDIA_ERR_DECODE (3) – Um erro de codificação impediu a exibição do vídeo.
- MEDIA_ERR_SRC_NOT_SUPPORTED (4) – O endereço do vídeo definido no atributo src não aponta para um arquivo válido.

O exemplo a seguir é interativo e foi desenvolvido definindo-se propositalmente um endereço inexistente para o vídeo causando um erro que poderá ser visualizado ao se clicar um botão. Atualmente somente o navegador Firefox suporta essa propriedade.

► JavaScript

```
function id(id) {  
    return document.getElementById(id)  
}  
  
function init() {  
    var video = id('video');  
    var valor = id('valor');  
    var btn = id('btn');  
    var erro = video.error;  
    var tipoErro = erro.code;  
    btn.onclick = function() {  
        var texto = 'Objeto erro retornado: ' + erro + '  
Código do erro: ' + tipoErro;  
        valor.innerHTML = (texto);  
    }  
}  
  
window.onload = init;
```

► HTML

```
<video poster="capa.jpg" controls id="video">
  <source src="xpto/video.ogv" type="video/ogg">
  <source src="xpto/video.mp4" type="video/mp4">
  <source src="xpto/video.webm" type="video/webm">
  <!-- Código (X)HTML para inserção de video com Flash -->
  <p>Seu navegador não suporta o elemento video da HTML5.
  <br>Faça <a href="video.mp4">download do video</a></p>
</video>
```



[c3-video-ex8.html]

currentSrc

Esse atributo retorna o valor do atributo `src` da mídia, ou seja, o endereço do arquivo de mídia e sua sintaxe, como mostrado a seguir.

```
var video = document.getElementById('video')
alert(video.currentSrc); // alerta o endereço do arquivo de vídeo
```



[c3-video-ex9.html]

networkState

Esse atributo monitora o estado em que encontra o vídeo em relação à atividade da rede. Retorna um número código para o estado com o significado mostrado a seguir.

- `NETWORK_EMPTY` (0) – O elemento de mídia não foi inicializado. Todos os atributos estão no seu estado inicial.
- `NETWORK_IDL`(1) – O algoritmo de seleção do endereço do arquivo de mídia foi ativado e selecionou o endereço, porém não foi iniciada nenhuma atividade de rede.
- `NETWORK_LOADING` (2) – O agente de usuário está em processo de download.
- `NETWORK_NO_RESOURCE` (3) – O algoritmo de seleção do endereço do arquivo de mídia foi ativado, mas o arquivo ainda não foi encontrado.

O exemplo a seguir demonstra a sintaxe para esse atributo.

```
var video = document.getElementById('video')
switch (video.networkState) {
  case 0:
    // código;
    break;
```

```
case 1:  
    // código;  
    break;  
case 2:  
    // código;  
    break;  
case 3:  
    // código;  
    break;  
}
```

preload

Esse atributo fornece uma dica de quanto do arquivo de vídeo deve ser pré-carregado. A sintaxe de uso é como mostrado a seguir.

```
var video = document.getElementById('video')  
alert(video.preload);
```

buffered

Esse atributo retorna um objeto representando um período de tempo correspondente à quantidade de mídia já inserida no buffer, ou seja, armazenada na memória temporária.

readyState

Esse atributo monitora o estado em que encontra o vídeo, em determinado tempo de execução, em relação à renderização. Retorna um número código para o estado com o significado mostrado a seguir.

- HAVE NOTHING(0) – Não se dispõe de nenhuma informação sobre o arquivo de mídia.
- HAVE_METADATA(1) – Já se conhece a duração da mídia e, no caso de vídeos, suas dimensões, contudo, informações sobre determinado tempo de execução ainda não estão disponíveis.
- HAVE_CURRENT_DATA (2) – Informações sobre o tempo atual de execução já estão disponíveis, mas não se conhece nada sobre o tempo de execução futuro.
- HAVE_FUTURE_DATA (3) – Há informações sobre o tempo de execução posterior ao atual, mas sem se conhecer as informações completas.
- HAVE_ENOUGH_DATA (4) – Todas as informações são conhecidas.

paused

Esse atributo é booleano e retorna `true` caso a mídia tenha sido parada e `false` caso contrário. No estado inicial de carregamento da página, o valor desse atributo é `true`. A sintaxe de uso é como mostrado a seguir.

```
var video = document.getElementById('video');
if (video.paused) {
    // código para quando o vídeo for parado
}
```

ended

Esse atributo é booleano e retorna `true` caso a mídia tenha chegado ao seu final. A sintaxe de uso é como mostrado a seguir.

```
var video = document.getElementById('video');
if (video.ended) {
    // código para quando o vídeo chegar ao final
}
```

played

Esse atributo retorna um objeto contendo a quantidade de mídia executada.

playbackRate

Esse atributo retorna a taxa de velocidade da mídia sendo executada. A velocidade normal é expressa para valor desse atributo igual a 1. Pode ser usado para definir uma velocidade diferente da normal. Assim, podemos controlar a velocidade de apresentação da mídia definindo valores diferentes de 1 para esse atributo. Valores maiores de 1 aceleram a apresentação (Fast Forward), valores negativos atrasam a apresentação (Rewind) e valores entre 0 e 1 apresentam a mídia em câmera lenta (Slow Motion). Atualmente somente a *engine WebKit* (Chrome e Safari) suporta esse atributo.

defaultPlaybackRate

Esse atributo retorna a taxa de velocidade da mídia sendo executada em condições normais, quando o usuário não acionou adiantar ou atrasar a mídia.

seeking

Um dos controles de mídia é aquele que possibilita ao usuário adiantar ou atrasar a execução para determinada posição de execução. O atributo `seeking` é booleano e retorna `true` se a mídia está em processo de busca de uma nova posição de execução.

seekable

Retorna um objeto contendo informações sobre o processo de busca de uma nova posição de execução.

tracks

Esse atributo possui a propriedade `length` (`track.length`) que retorna o número de trilhas de texto de uma mídia. Também retorna um objeto tipo array contendo informações sobre determinada trilha (`track[n]`).

controls

Esse atributo é booleano e retorna `true` se o atributo `controls` foi definido para o elemento mídia. A sintaxe de uso é como mostrado a seguir.

```
var video = document.getElementById('video');
if (video.controls) {
    video.removeAttribute('controls'); // remove os controles de mídia para inserir
                                    // controles personalizados
}
```



Ao definirmos controles personalizados, devemos definir o atributo `controls` no elemento de mídia e retirá-lo com uso de script, como mostrado no código. Pois, se JavaScript estiver desabilitado, o usuário não será privado dos controles nativos.

autoplay

Esse atributo é booleano e retorna `true` se o atributo `autoplay` foi definido para o elemento mídia.

poster

Esse atributo retorna o endereço definido no atributo `poster` para o vídeo. A sintaxe de uso é como mostrado a seguir.

```
var video = document.getElementById('video');
video.poster = ' '; // remove a imagem inserida como capa do vídeo
```

loop

Esse atributo é booleano e retorna `true` se o atributo `loop` foi definido para o elemento mídia.

width

Esse atributo retorna a largura do elemento de mídia definida no atributo `width` para o elemento.

height

Esse atributo retorna a altura do elemento de mídia definida no atributo `height` para o elemento.

videoWidth

Esse atributo retorna a largura da mídia vídeo quer tenha sido definida ou não no atributo `width`.

videoHeight

Esse atributo retorna a altura da mídia vídeo quer tenha sido definida ou não no atributo `height`.

volume

Esse atributo retorna o nível de volume de áudio, se existir som na mídia, em uma escala que varia de 0 (silêncio) a 1 (máximo).

muted

Esse atributo é booleano e retorna `true` se o canal de áudio, se existir som na mídia, estiver em silêncio.

currentTime

Esse atributo retorna a posição de execução da mídia expressa em segundos.

startTime

Esse atributo retorna a posição em segundos na qual se inicia determinada legenda (ou track) inserida na mídia.

endTime

Esse atributo retorna a posição em segundos na qual termina determinada legenda ou track) inserida na mídia.

duration

Esse atributo retorna o tempo de duração da mídia em segundos.

3.3.1.1 Exemplo interativo

Desenvolvemos um exemplo interativo disponível no site do livro onde poderá ser constatado o funcionamento da maioria dos atributos da API de mídia para a HTML5. Ao visualizar e interagir com o exemplo, esteja ciente de que nem todos os navegadores implementaram as funcionalidades mostradas, além de haver diferenças de implementação de um navegador para outro.

O ideal é que você visualize e interaja em diferentes navegadores ou, pelo menos, no Firefox e Opera.



Os atributos da API não são definitivos. Uns poderão ser abandonados; outros, acrescidos, pois as especificações para a HTML5 estão em fase de rascunho. Os atributos aqui mostrados são atualmente previstos na especificação, mas não significa que podemos usá-los indiscriminadamente em fase de produção.

O exemplo desenvolvido é conforme mostrado a seguir.

► JavaScript

```
function init() {  
    var video = document.getElementById('video');  
    var btn = document.getElementsByTagName('button');  
    btn[0].onclick = function() {video.poster = ''};  
    btn[1].onclick = function() {video.poster = 'capa.jpg'};  
    btn[2].onclick = function() {alert('Largura do vídeo: '+video.width+'px')};  
    btn[3].onclick = function() {video.width = video.width*1.2};  
    btn[4].onclick = function() {video.width = '400'};  
    btn[5].onclick = function() {alert('Nível de volume do áudio (0-1): '+video.volume)};  
    btn[6].onclick = function() {  
        if (video.paused) {  
            alert('O vídeo está em Pausa');  
        } else if (video.ended) {  
            alert('Terminou a execução do vídeo');  
        }  
    };  
}
```

```

} else {
    alert('O vídeo está rodando');
}
}

btn[7].onclick = function() {
    var atr = 'Os atributos do elemento vídeo são: \n';
    if (video.currentSrc) atr += 'src\n';
    if (video.controls) atr += 'controls\n';
    if (video.autoplay) atr += 'autoplay\n';
    if (video.loop) atr += 'loop\n';
    if (video.poster) atr += 'poster\n';
    if (video.id) atr += 'id\n';
    if (video.width) atr += 'width\n';
    if (video.height) atr += 'height\n';
    alert(atr);
}

btn[8].onclick = function() {
    alert('A duração do vídeo é de: '+video.duration+' segundos');
}

btn[9].onclick = function() {
    alert('O tempo de execução no click foi de: '+video.currentTime+' segundos');
}

}

window.onload = init;

```

► HTML

```

<button type="button">Retirar capa</button>
<button type="button">Recolocar capa</button>
<button type="button">Largura do vídeo?</button><br>
<button type="button">Aumentar dimensões</button>
<button type="button">Dimensões default</button><br>
<button type="button">Volume?</button> Rode o vídeo altere o volume e click<br>
<button type="button">Estado do vídeo?</button> Click com vídeo rodando, em pause e final execução<br>
<button type="button">Atributos?</button>
<button type="button">Duração?</button><br>
<button type="button">Tempo execução?</button> Click com vídeo rodando

```



[c3-video-ex10.html]

3.3.2 Métodos

Vamos examinar os métodos constantes da API para as mídias de áudio e vídeo. Os exemplos mostrados nesse item foram criados para a mídia `video`, mas, na sua maioria, são aplicáveis à mídia `audio`.

`load()`

Esse método causa o carregamento da mídia.

`play()`

Esse método causa a reprodução do vídeo a partir da sua posição inicial.

`pause()`

Esse método causa a interrupção da reprodução do vídeo.

`canPlayType(tipo)`

Esse método verifica se o agente de usuário é capaz de reproduzir o vídeo cujo tipo é conforme especificado no parâmetro `tipo`. No parâmetro `tipo` deve-se especificar o MIME type e os codecs do vídeo. O método retorna uma string vazia ou a string “`maybe`” ou a string “`probably`”, conforme o navegador não ofereça suporte, ofereça suporte com uso de plugin, ou não há certeza de suporte respectivamente.

O exemplo a seguir, extraído das especificações para a HTML5, demonstra o uso desse método.

```
<section id="video">
  <p><a href="playing-cats.nfv">Download video</a></p>
</section>
<script>
  var videoSection = document.getElementById('video');
  var videoElement = document.createElement('video');
  var support = videoElement.canPlayType('video/x-new-fictional-format;codecs="kittens,bunnies"');
  if (support != "probably" && "New Fictional Video Plug-in" in navigator.plugins) {
    // provavelmente o navegador suporta o tipo com uso de um plugin
    // se o plugin tiver sido instalado use-o
    videoElement = document.createElement("embed");
  } else if (support == "") {
    // o navegador não suporta o tipo de vídeo
    // não insira o elemento video no DOM
    videoElement = null;
}
```

```
if (videoElement) { // retira o parágrafo download do DOM e insere o vídeo
    while (videoSection.hasChildNodes())
        videoSection.removeChild(videoSection.firstChild);
    videoElement.setAttribute("src", "playing-cats.nfv");
    videoSection.appendChild(videoElement);
}
</script>
```

Nesse exemplo a situação inicial é para o download do vídeo denominado *playing-cats.nfv*, um formato não previsto nas especificações da HTML5. Um script JavaScript vai tentar inserir a marcação HTML para a apresentação do vídeo com uso do elemento `video` da HTML5, uma vez que a apresentação do vídeo é possível com uso de um plugin.

O script começa verificando se o agente de usuário é capaz de reproduzir o vídeo do tipo `video/x-new-fictional-format; codecs="kittens,bunnies"`. Para essa verificação usa uma condicional `if` para o método `canPlayType(tipo)`. Dependendo da resposta do navegador ao método (string vazia ou a string “maybe” ou a string “probably”), o script retira o parágrafo de download inicial, cria a marcação HTML5 para vídeo e insere no lugar do parágrafo, servindo o vídeo ou então nada cria e deixa tudo como está na situação inicial.

addTrack(*tipo, legenda, idioma*)

No item 3.2.1 estudamos a criação de trilhas diretamente na marcação com uso do elemento `track`. Esse método destina-se igualmente a criar trilhas para o vídeo com uso de `script`. Se você não se lembra daquele elemento, faça uma nova leitura do item 3.2.1.

O uso desse método, assim como do elemento `track`, ainda não é suportado pelos navegadores atuais e o formato de sua implementação ainda é objeto de discussão.

3.3.3 Eventos

Vamos examinar os eventos constantes da API para as mídias de áudio e vídeo. Os exemplos mostrados nesse item foram criados para a mídia `video`, mas, na sua maioria, são aplicáveis à mídia `audio`.

loadstart

Evento ocorre quando o agente de usuário iniciou o processo de busca dos metadados da mídia no endereço especificado no atributo `src`.

progress

Evento ocorre quando o agente de usuário se encontra em processo de obtenção e download dos dados da mídia.

suspend

Evento ocorre quando o agente de usuário suspendeu intencionalmente a obtenção dos dados da mídia sem que o processo de download tenha chegado ao fim.

abort

Evento ocorre quando o agente de usuário parou a obtenção dos dados da mídia sem que o processo de download tenha chegado ao fim e por motivo que não foi causado por um erro.

error

Evento ocorre quando há um erro que interrompe a obtenção dos dados da mídia.

emptied

Evento ocorre quando o elemento de mídia, não vazio, foi esvaziado durante o processo de obtenção e download dos dados da mídia (seja pela ocorrência de um erro fatal, seja por ter sido evocado o método `load()` durante o processo)

stalled

Evento ocorre quando o agente de usuário está tentando a obtenção dos dados da mídia, mas os dados não estão vindo.

play

Evento ocorre quando o agente de usuário inicia a reprodução da mídia.

pause

Evento ocorre quando o agente de usuário coloca a reprodução da mídia em pausa.

loadedmetadata

Evento ocorre quando o agente de usuário acaba de determinar a duração, legendas e dimensões da mídia.

loadeddata

Evento ocorre quando o agente de usuário fica em condições de renderizar os dados da mídia, em determinada posição de reprodução, pela primeira vez.

waiting

Evento ocorre quando o agente de usuário para a execução da mídia porque o próximo quadro ainda não está disponível, mas estará em breve.

playing

Evento ocorre quando o agente de usuário inicia a reprodução do vídeo.

canplay

Evento ocorre quando o agente de usuário estima que se a reprodução começar agora não estará em condições de renderizar a mídia até seu final sem ter de parar para carregar o buffer.

canplaythrough

Evento ocorre quando o agente de usuário estima que se a reprodução começar agora estará em condições de renderizar a mídia até seu final sem ter de parar para carregar o buffer.

seeking

Evento ocorre quando o atributo `seeking` assume o valor `true` (está havendo busca por uma nova posição de reprodução).

seeked

Evento ocorre quando o atributo `seeking` assume o valor `false` (terminou a busca por uma nova posição de reprodução).

timeupdate

Evento ocorre quando há uma mudança na posição de reprodução da mídia decorrente de um fator previsto na reprodução ou quando ocorre uma descontinuidade na reprodução.

ended

Evento ocorre quando a reprodução da mídia chega ao seu final.

ratechange

Evento ocorre quando há uma mudança na taxa de reprodução da mídia decorrente de mudança no valor dos atributos `defaultPlaybackRate` ou `playbackRate`.

durationchange

Evento ocorre quando há uma mudança na duração da mídia decorrente de mudança no valor do atributo `duration`.

volumechange

Evento ocorre quando há uma mudança no nível de volume da mídia decorrente de mudança no valor dos atributos `volume` ou `muted`.

3.3.3.1 Sintaxe para eventos

A sintaxe geral para atrelar uma função a ser executada na ocorrência de um evento usa o método do DOM 2 `addEventListener(evento, função, booleano)`.

Esse método permite atrelar um evento a um nó do DOM. O parâmetro `evento` define o tipo do evento; o parâmetro `função` define a função a ser executada, ou seja, o manipulador do evento. O parâmetro `booleano` admite os valores `true` e `false`. Esse parâmetro define em que fase da propagação do evento deverá ser executada a função. Na maioria dos casos, esse parâmetro deve ser definido como `false`.

Convém ressaltar que o navegador Internet Explorer versões 8 e anteriores não dão suporte a esse método, adotando um método proprietário denominado `attachEvent()`. Felizmente o navegador Internet Explorer 9 promete suporte ao método e às funcionalidades da HTML5.

O exemplo a seguir, disponível no site do livro, é interativo e esclarece a sintaxe para eventos e exemplifica o funcionamento de alguns dos eventos mostrados.

► **JavaScript**

```
function init() {  
    var video = document.getElementById('video');  
    var btn = document.getElementsByTagName('button');  
    var valor = document.getElementById('valor');
```

```

var evento = function (evt) {
    valor.innerHTML = ('Disparado o evento: '+evt.type);
}
video.addEventListener('play', evento, false);
video.addEventListener('progress', evento, false);
video.addEventListener('pause', evento, false);
video.addEventListener('ended', evento, false);
video.addEventListener('volumechange', evento, false);
}
window.onload = init;

```

► **HTML**

```

<video poster="capa.jpg" controls id="video" width="400">
    <source src="video.ogv" type="video/ogg">
    <source src="video.mp4" type="video/mp4">
    <source src="video.w ebm" type="video/webm">
    <!-- Código (X)HTML para inserção de vídeo com Flash -->
    <p>Seu navegador não suporta o elemento video da HTML5.
        <br>Faça <a href="video.mp4">download do vídeo</a></p>
</video>
<p id="valor" class="dest"></p>

```



[c3-video-ex11.html]

3.3.3.2 Controles personalizados

Para ilustrar o uso dos atributos, métodos e eventos da API de mídia da HTML5, desenvolveremos e comentaremos passo a passo um script para personalizar os controles de um vídeo. Os diferentes arquivos constantes em cada um dos passos mostrados estão disponíveis para visualização no site do livro.

Nos passos iniciais, já com o controle play e pause devidamente personalizados e em funcionamento, decidimos manter o controle-padrão do navegador, apesar de desnecessário, com fins puramente didáticos. Em um estágio mais avançado, retiraremos o controle-padrão do navegador.

A marcação HTML para o vídeo é a mesma mostrada para os exemplos anteriores e transcrita a seguir.

► **HTML**

```

<video poster="capa.jpg" controls id="video" width="400">
    <source src="video.ogv" type="video/ogg">
    <source src="video.mp4" type="video/mp4">

```

```
<source src="video.w ebm" type="video/webm">
<!-- Código (X)HTML para inserção de video com Flash -->
<p>Seu navegador não suporta o elemento video da HTML5.
<br>Faça <a href="video.mp4">download do video</a></p>
</video>
```

Primeiro passo

Vamos inserir na marcação HTML um botão contendo o símbolo-padrão para acionar a reprodução de vídeo (um triângulo apontando para a direita), como mostrado a seguir.

► HTML

```
<video poster="capa.jpg" controls id="video" width="400">
  ...
</video>
<p><button type="button" id="btn">▶</button></p>
```

A entidade XML ▶ representa um triângulo idêntico ao gráfico típico representativo de um botão do tipo play. Para a representação do botão de pause adotaremos a entidade XML ▶ duas vezes (uma ao lado da outra), pois ela representa uma barra vertical e o gráfico típico representativo de um botão tipo pause é constituído por duas barras verticais.

Inserido o botão com o símbolo para play, vamos desenvolver um script para iniciar a reprodução do vídeo quando nele ocorrer um click do usuário. Observe a seguir.

► JavaScript

```
function init() {
  var video = document.getElementById('video');
  var btn = document.getElementById('btn');
  var play = function { video.play(); }
  btn.addEventListener('click', play, false);
}
window.onload = init;
```



[c3-video-ex12-P1.html]

Clicando o botão começa a reprodução do vídeo, mas não há ainda um botão para pausa.

Segundo passo

Poderíamos inserir outro botão na marcação HTML, contendo o símbolo-padrão para pausa, e a ele atrelar uma função para acionar pausa tal como fizemos com play.

Contudo, há uma solução mais elegante que consiste em usar um só botão para desempenhar as funções de play e pause. A lógica da programação para conseguir esse efeito quando o botão é clicado é a seguinte:

- Se o vídeo estiver em pausa, comece a reprodução e troque o símbolo do botão para pausa.
- Se o vídeo estiver em reprodução, pare e troque o símbolo do botão para reprodução.

Escrever essa lógica é como mostrado a seguir.

► JavaScript

```
function init() {  
    var btn = document.getElementById('btn');  
    var video = document.getElementById('video');  
    var playPause = function() {  
        if (video.paused || video.ended) {  
            video.play();  
            this.innerHTML = '&#x2588; &#x2588;';  
        } else {  
            video.pause();  
            this.innerHTML = '&#x25B6';  
        }  
    }  
    btn.addEventListener('click', playPause, false);  
}  
window.onload = init;
```



[c3-video-ex12-P2.html]

Clicando o botão começa a reprodução do vídeo e o símbolo do botão muda para o gráfico típico de pausa. Clicando novamente, a reprodução do vídeo cessa e o símbolo do botão muda para o gráfico típico de reprodução. Notar o uso do atributo `paused` e dos métodos `play()` e `pause()` da API de mídia da HTML5.

Você observou o que acontece com o botão quando o vídeo chega ao final?

Terceiro passo

Você deve ter observado que, quando a reprodução do vídeo chega ao final, ela cessa e o vídeo entra em estado de pausa, mas o botão não muda para o símbolo de reprodução como era de se esperar.

Vamos usar o evento `ended` da API de mídia da HTML5 para detectar o instante de chegada da reprodução do vídeo ao seu final e, quando esse evento ocorrer, faremos a mudança do símbolo do botão. O seguinte acréscimo deverá ser feito no script anterior.

► JavaScript

```
function init() {  
    var btn = document.getElementById('btn');  
    // igual ao anterior  
    btn.addEventListener('click', playPause, false);  
    video.addEventListener('ended', function() {  
        btn.innerHTML = '▶';  
    }, false);  
}  
window.onload = init;
```



[c3-video-ex12-P3.html]

Terminada a personalização dos botões play e pause? Ainda não. Você saberia dizer o que falta sob o ponto de vista da funcionalidade dos botões?

Quarto passo

A maioria dos navegadores oferece ao usuário um menu de contexto quando se clica com o botão direito do mouse sobre a área de apresentação do vídeo, como mostrado na figura 3.22, para o sistema operacional Windows e navegador Firefox.

O menu de contexto oferece ao usuário várias opções para controle do vídeo; entre elas, as opções para reproduzir e parar a apresentação do vídeo, isto é, acionar play e pause no menu de contexto.

Sugiro que você abra o arquivo `c3-video-ex12-P3.html` e faça as seguintes experiências:

- Inicie o vídeo clicando o botão play e depois abra o menu de contexto e pare a reprodução clicando a opção **Pausa**.
- Recarregue a página, abra o menu de contexto e inicie o vídeo clicando a opção **Exibir vídeo**.

Em ambos os casos, observe o botão personalizado e compare com o botão do controle-padrão do navegador. Qual a conclusão?

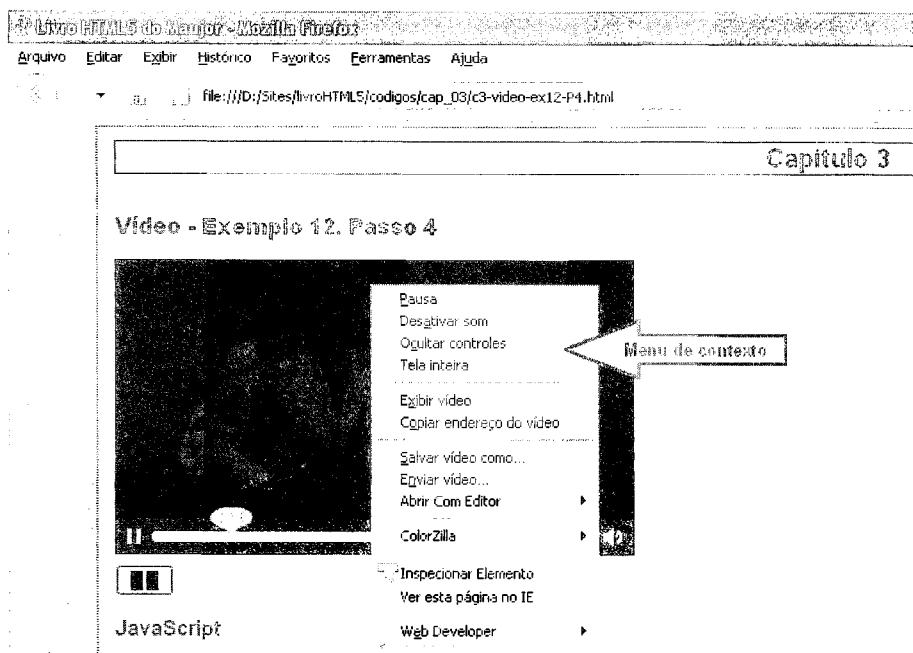


Figura 3.22 – Menu de contexto para vídeo.

Quando o controle do vídeo é feito pelo menu de contexto, não está havendo a troca do símbolo do botão, ou seja, acionar pausa pelo menu de contexto não coloca o botão no estado play e vice-versa. Isso fica claro quando se compara o botão personalizado com o botão do controle-padrão do navegador. Por que não está ocorrendo o comportamento correto para o botão personalizado?

Porque a troca do símbolo no botão se faz mediante click no próprio botão. Então, como resolver?

Resolvemos transferindo a responsabilidade de disparo das funções que acionam `play()` e `pause()` de um evento clique no botão para um evento ocorrendo no próprio vídeo, mas precisamente para os eventos `play` e `pause`. Dessa forma, a troca de símbolos se fará em função da mudança do estado de reprodução do vídeo, independentemente do local de onde tenha partido o comando, seja o botão, seja o menu de contexto.

O exemplo a seguir mostra os acréscimos feitos no exemplo anterior para atender à troca de símbolos quando o usuário usa o menu de contexto.

► JavaScript

```
function init() {  
    var btn = document.getElementById('btn');  
    // igual ao anterior  
    video.addEventListener('ended', function() {  
        btn.innerHTML = '\u25b6';  
    }, false);  
    video.addEventListener('pause', function() {  
        btn.innerHTML = '\u25b6';  
    }, false);  
    video.addEventListener('play', function() {  
        btn.innerHTML = '\u2588 \u2588';  
    }, false);  
}  
window.onload = init;
```



[c3-video-ex12-P4.html]

Quinto passo

Se optamos por personalizar os controles do vídeo com JavaScript, não necessitamos dos controles-padrão do navegador. Podemos nos livrar dos controles-padrão de duas formas: retirar o atributo `controls` do elemento vídeo removendo-o diretamente na marcação HTML ou removendo-o com uso de script.

A remoção com uso de script é a solução mais indicada, pois, na falta de suporte à JavaScript no navegador, o usuário não ficará privado dos controles.

Ainda, na falta de JavaScript, irá aparecer um botão personalizado sem qualquer funcionalidade, pois ele foi inserido diretamente na marcação HTML. Nesse caso, é também conveniente retirar o botão personalizado da marcação e inserir com uso de script. O exemplo a seguir demonstra esse quinto passo. As linhas de código em destaque mostram as modificações introduzidas no script anterior.

► JavaScript

```
function init() {  
    var btn = document.getElementById('btn');  
    var btn = document.createElement('button');  
    btn.innerHTML = ('\u25b6');  
    var secao = document.getElementsByTagName('section')[0];  
    secao.insertBefore(btn, video);  
    var video = document.getElementById('video');  
    video.removeAttribute('controls');
```

```

var playPause = function() {
    if (video.paused || video.ended) {
        video.play();
        this.innerHTML = '&#x2588; &#x2588;';
    } else {
        video.pause();
        this.innerHTML = '&#x25B6';
    }
}
btn.addEventListener('click', playPause, false);
video.addEventListener('ended', function() {
    btn.innerHTML = '&#x25B6';
}, false);
video.addEventListener('pause', function() {
    btn.innerHTML = '&#x25B6';
}, false);
video.addEventListener('play', function() {
    btn.innerHTML = '&#x2588; &#x2588;';
}, false);
}
window.onload = init;

```



[c3-video-ex12-P5.html]

Nesse quinto passo encerramos a personalização do botão play/pause.

Sexto passo

Vamos examinar nesse passo o script para personalização do controle de volume. Usaremos o atributo `volume` da API para elementos de mídia da HTML5. Lembramos que esse atributo é do tipo leitura e escrita e admite valores que vão de 0 a 1, sendo o valor 0 para silêncio e 1 para volume máximo do som.

Usaremos também o elemento `input` do tipo `range` que em HTML5 se destina a marcar um slider, ou seja, um controle corrediço. Definiremos uma faixa de 0 a 1 para esse elemento e, quando o usuário deslizar o controle, o seu valor será passado por script para o atributo `volume`. A sintaxe para marcação desse elemento é mostrada a seguir.

```
<input type="range" min="0" max="1" value="1" step="0.1" id="slider">
```

Navegadores que não suportam o tipo `range` para o elemento `input` renderizam um campo do tipo `text`. Atualmente os navegadores Chrome, Safari e Opera suportam esse elemento.

Esse elemento cria um slider horizontal, e sua renderização-padrão nos navegadores Chrome e Opera é mostrada na figura 3.23. É possível obter uma renderização vertical definindo-se regra de estilo, como mostrado a seguir.

```
#slider {width:15px; height:50px;}
```

Na figura 3.23 mostra-se também o resultado da estilização para tornar o controle vertical que atualmente é suportada no navegador Opera, mas falha no Chrome e Safari.

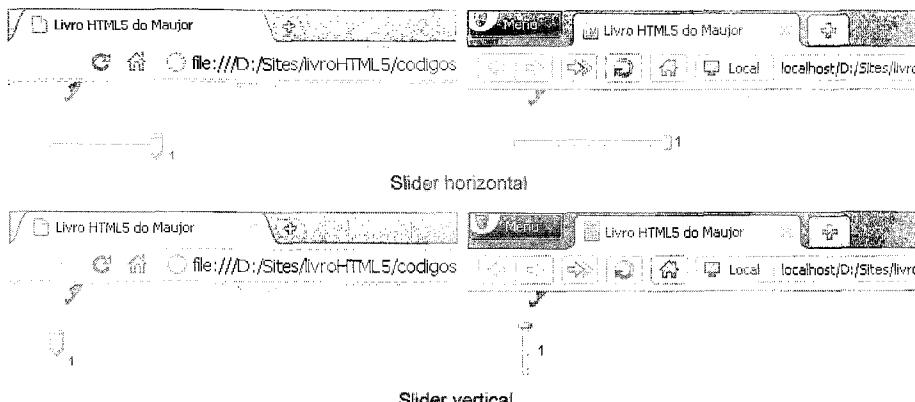


Figura 3.23 – Slider no Chrome e Opera.

O exemplo a seguir demonstra esse sexto passo. As linhas de código em destaque mostram os acréscimos introduzidos no script anterior.

► JavaScript

```
function init() {
    var video = document.getElementById('video');
    var slider = document.getElementById('slider');
    var vol = document.getElementById('vol');
    slider.addEventListener('change', function() {
        video.volume = this.value;
        vol.innerHTML = (this.value);
    }, false);
    // a partir daqui igual ao anterior
    secao.insertBefore(btn, video);
    secao.insertBefore(btn, slider);
    var video = document.getElementById('video');
    ...
}
```

window.onload = init;

► HTML

```
<video poster="capa.jpg" id="video" width="400">
  ...
</video>
<input type="range" min="0" max="1" value="1" step="0.1" id="slider" />
<span id="vol">1</span>
```



[c3-video-ex12-P6.html]

Sétimo passo

Nesse passo vamos inserir controles personalizados para adiantar (Fast Forward), atrasar (Rewind) e reproduzir o vídeo em câmera lenta.

Usaremos também elementos `button` e entidades XML para representar os respectivos símbolos nos botões, tal como fizemos com o botão play. O atributo que controla a velocidade de movimentação do vídeo é o atributo `playbackRate`. Para esse atributo o valor 1 representa a velocidade normal de reprodução, valores positivos maiores que 1 aumentam a velocidade, valores negativos causam retrocesso do vídeo e valores entre 0 e 1 causam a reprodução em câmera lenta (Slow Motion)

O exemplo a seguir demonstra esse sétimo passo. As linhas de código em destaque mostram os acréscimos introduzidos no script anterior.

► JavaScript

```
function init() {
  ...
  var btnFF = document.createElement('button');
  btnFF.innerHTML = '&#x25BA;&#x25BA;';
  secao.insertBefore(btnFF, slider);

  var btnRW = document.createElement('button');
  btnRW.innerHTML = '&#x25C4;&#x25C4;';
  secao.insertBefore(btnRW, slider);

  var btnSM = document.createElement('button');
  btnSM.innerHTML = '&#x25BA;&#x2590;';
  secao.insertBefore(btnSM, slider);

  ...
}
```

```
btnFF.addEventListener('click', function() {
    video.play();
    video.playbackRate = 3;
}, false);
btnRW.addEventListener('click', function() {
    video.play();
    video.playbackRate = -20;
}, false);

btnSM.addEventListener('click', function() {
    video.play();
    video.playbackRate = 0.3;
}, false);
}
window.onload = init;
```



[c3-video-ex12-P7.html]

Oitavo passo

O último controle do vídeo a personalizar é a barra de progresso que indica o andamento da reprodução.

Usaremos o elemento `progress` para criar a barra de progresso. Esse é um elemento novo da HTML5 e se destina a fornecer um indicador dinâmico da quantidade realizada de determinada tarefa. Em nosso caso, indicará a quantidade de vídeo reproduzida durante o curso da reprodução.

A medição da quantidade de tarefa a realizar será feita com uso do atributo `duration` da API para elementos de mídia. Esse atributo retorna o tempo de duração da mídia, em segundos, necessário para a reprodução. Para medir a quantidade de tarefa realizada em determinado momento, usaremos atributo `currentTime` da API para elementos de mídia. Esse atributo retorna o número de segundos decorridos desde o início da reprodução da mídia até o momento atual.

O exemplo a seguir demonstra esse oitavo passo. As linhas de código em destaque mostram os acréscimos introduzidos no script anterior.

► JavaScript

```
function init() {  
    ...  
    var progress = document.getElementsByTagName('progress')[0];  
    var pp = document.getElementById('pp');  
    function atualizar() {  
        progress.setAttribute('max', video.duration);  
        var cT = video.currentTime;  
        progress.value = cT;  
        var porcentagem = cT/video.duration * 100;  
        if (parseInt(porcentagem%2) == 0) {  
            pp.textContent = parseInt(porcentagem);  
            if (porcentagem == 100) {clearInterval(atualizar)}  
        }  
    }  
    setInterval(atualizar, 300);  
}  
window.onload = init;
```

► HTML

```
...  
<br>  
<progress max=""></progress>  
<span id="pp">0</span>%  
...
```



[c3-video-ex12-P8.html]

Atualmente só o navegador Chrome suporta o elemento `progress` e seus atributos. O Opera suporta parcialmente e o Firefox não suporta. Sobre o script convém ressaltar os seguintes pontos:

- A função `atualizar` destina-se a atualizar o andamento da reprodução da mídia colhendo o valor, em segundos, do instante atual de reprodução e, com cálculos simples, transforma esse valor em porcentagem do total de tempo necessário para a reprodução da mídia.
- O valor calculado para a porcentagem é inserido em um elemento `span` e, ao mesmo tempo, passado como valor do atributo `value` do elemento `progress`. Isso fornece ao usuário uma indicação da porcentagem da mídia executada não só no elemento `span` como na barra de progresso.

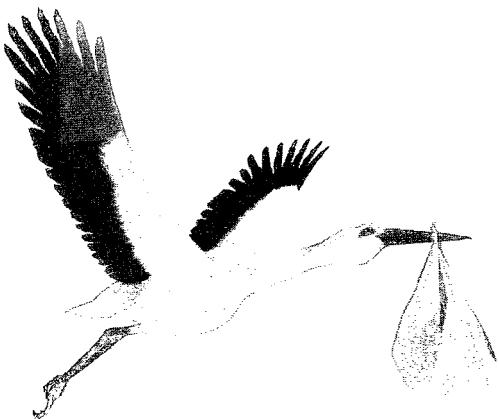
- A função atualizar é executada a cada 300ms com uso do método `setInterval` da JavaScript.
- Notar o uso do atributo `textContent` para atualizar o conteúdo do elemento `span`. Esse atributo é do tipo escrita/leitura e funciona de forma idêntica à propriedade `innerHTML` para escrita. É um atributo constante do DOM3 e suportado pelos navegadores atuais, exceto o Internet Explorer versões 8 e anteriores.

Nono passo

Esse fica para você exercitar suas habilidades com JavaScript.

Você deve ter notado que os botões de controle foram inseridos no DOM com uso de script; contudo, o slider de controle do som e a barra de progresso da reprodução do vídeo constam da marcação HTML da página. Nesse nono passo, proponho que você retire esses controles da marcação e insira-os no DOM com uso de script tal como fizemos com os botões.

Qual a vantagem dessa técnica? Como o funcionamento dos controles depende da JavaScript, se ela estiver desabilitada no navegador, os controles personalizados não serão renderizados e o usuário usará os controles nativos sem ver botões, barra de progresso e slider que não têm qualquer função.



CAPÍTULO 4

Canvas

Neste capítulo, será apresentado o elemento `canvas` e suas funcionalidades. Mostraremos a interface do elemento relacionando, definindo e explicando o funcionamento de cada um dos seus métodos e atributos. Apresentaremos exemplos práticos disponíveis para consulta online ou download para estudo local nos quais as funcionalidades de criação serão demonstradas a título de exemplificação prática do estudo teórico relatado.

4.1 Canvas

O elemento `canvas` destina-se a delimitar uma área para criação dinâmica de imagens, como gráficos estáticos, jogos e gráficos dinâmicos e imagens em geral criadas com linguagem de programação dinâmica. Todo o trabalho de criação e animação é feito com JavaScript.

É necessário que o desenvolvedor faça uma análise criteriosa para uso desse elemento. Não devemos usá-lo se houver um elemento mais apropriado. Por exemplo: para marcar o cabeçalho ou o topo de um site, ainda que ele tenha sido projetado com uso maciço de imagens, talvez seja mais apropriado o uso do elemento `h1` devidamente estilizado e não o uso de `canvas`. Como regra geral, não devemos simplesmente decidir usar `canvas` e descartar outros elementos sempre que temos uma imagem na página.

O acesso ao conteúdo gráfico contido em `canvas`, em navegadores que não suportam esse elemento, deve ser garantido com uso de conteúdo alternativo com a mesma função ou propósito do conteúdo gráfico. Tal conteúdo deve constar da marcação

da página e ser inserido tendo `canvas` como seu container. Tal como ocorre com o elemento `audio` estudado anteriormente, o conteúdo inserido em `canvas` é ignorado por navegadores que suportam esse elemento e renderizado pelos que não suportam.

As questões de acessibilidade ao conteúdo gráfico transmitido por `canvas` ainda estão sem solução nas especificações da HTML5. O navegador Internet Explorer 9 deu um passo à frente e implementou a funcionalidade que permite às tecnologias assistivas e à navegação por teclado acessar o conteúdo alternativo de `canvas` mesmo que o navegador suporte esse elemento. Até o momento é o único navegador com essa funcionalidade.

O grupo de trabalho da HTML5, desde julho de 2007, estuda alternativas na API de `canvas` que permitam prover nativamente a acessibilidade a esse elemento; contudo, até o presente (meados do ano 2011) nada está definido.

A marcação para definir a área de criação dinâmica de imagens é muito simples, como mostrada a seguir.

► CSS

```
canvas {  
    border: 1px solid #000;  
    background: #ffc;  
}
```

► HTML

```
<canvas>  
    <p> Conteúdo alternativo para navegadores que não suportam canvas.</p>  
</canvas>
```



[c4-canvas-ex1.html]

No exemplo mostrado, disponível no site do livro, definimos, com uso de regra de estilo, uma borda e uma cor de fundo para o elemento `canvas` com a finalidade de facilitar a visualização. Convém notar que, por padrão, as dimensões de `canvas` quando não forem definidas serão iguais a 300 x 150px.

Contudo, você não estará limitado a essas dimensões, pois, além dos atributos globais, o elemento `canvas` suporta os atributos `width` e `height` destinados a definir respectivamente a largura e a altura da área de criação das imagens dinâmicas. No exemplo a seguir, usamos esses atributos para definir uma área com dimensões personalizadas.

► CSS

```
canvas {
    border: 1px solid #000;
    background: #ffc;
}
```

► HTML

```
<canvas width="600" height="300">
<p> Conteúdo alternativo para navegadores que não suportam canvas.</p>
</canvas>
```



[c4-canvas-ex2.html]

A unidade de medida das dimensões definidas com uso dos atributos é o pixel, porém é admitido definir-se as dimensões de `canvas` com CSS; nesse caso, podemos usar qualquer medida CSS válida, tais como `em`, `porcentagem` etc. O exemplo a seguir ilustra o caso de definição das dimensões com CSS.

► CSS

```
canvas {
    width:100%;
    height:200px;
    border: 1px solid #000;
    background: #ffc;
}
```

► HTML

```
<canvas>
<p> Conteúdo alternativo para navegadores que não suportam canvas.</p>
</canvas>
```



[c4-canvas-ex3.html]

Uma vez definida a área de criação, o próximo passo é a criação da imagem propriamente dita. Todo o trabalho de criação é feito com JavaScript.

A especificação para a HTML5 prevê dois contextos de criação: o contexto bidimensional e o contexto tridimensional.

O contexto bidimensional destina-se à criação de imagens em duas dimensões, `x` e `y`, segundo um sistema de eixos cartesianos, cuja origem é no canto superior esquerdo da área de criação (`canvas`), com as coordenadas `x` crescendo da esquerda para a direita e `y` de cima para baixo, conforme esquema mostrado na figura 4.1.

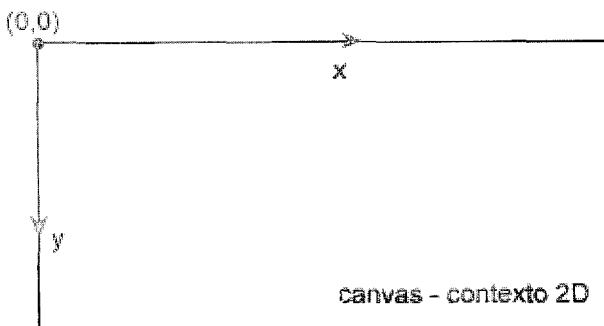


Figura 4.1 – Coordenadas do contexto 2D de canvas.

O contexto tridimensional destina-se à criação de imagens em três dimensões com acréscimo do eixo z no sentido da profundidade. Esse contexto ainda não consta das especificações; portanto, está fora do escopo deste livro. Estudaremos com detalhes o contexto bidimensional.

Para começar a desenhar, precisamos criar uma referência JavaScript para o elemento `canvas` no DOM e definir o contexto de desenho. Observe o exemplo comentado a seguir.

► HTML

```
<canvas id="quadro" width="400" height="200">
<p>Conteúdo alternativo para navegadores que não suportam canvas.</p>
</canvas>
```

► JavaScript

1. var canvas = document.getElementById('quadro');
2. var ctx = canvas.getContext('2d');

Código comentado:

Linha(s)	Descrição
Linha 1	Cria uma referência (var canvas) para o elemento <code>canvas</code> que servirá de área para a criação gráfica.
Linha 2	Usa o método <code>getContext()</code> previsto na API de <code>canvas</code> para criar a referência (var ctx) ao contexto de criação gráfica. Essa referência é a chave para acessar os métodos e atributos JavaScript da API.

A API de `canvas` prevê métodos e atributos próprios que visam a facilitar a criação gráfica. Além deles, é claro, qualquer objeto e seus métodos e propriedades próprios da linguagem JavaScript podem ser utilizados na criação.

Mostraremos a seguir alguns exemplos usando métodos e atributos da API de canvas. Para uma visão geral de todos os métodos e atributos, consulte <http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html>.

Para os exemplos a seguir, a marcação utilizada e a tomada JavaScript das referências para a criação são as mostradas no exemplo anterior e transcritas a seguir.



Inserimos o script diretamente na página dos exemplos, constantes do site do livro, com a finalidade de facilitar a visualização quando o leitor abrir o código-fonte da página. Em produção devemos linkar a página para o script. Além disso, ao mostrar os scripts no livro, omitimos a declaração `window.onload = function() { //script }` que consta das páginas dos exemplos.

► HTML

```
<canvas id="quadro" width="400" height="200">
<p>Conteúdo alternativo para navegadores que não suportam canvas.</p>
</canvas>
```

► JavaScript

1. var canvas = document.getElementById('quadro');
2. var ctx = canvas.getContext('2d');

•

4.1.1 API de canvas

4.1.1.1 strokeRect(*x, y, l, h*)

Esse método destina-se a desenhar o contorno de um retângulo. Os parâmetros *x* e *y* definem as coordenadas do canto superior esquerdo do retângulo e os parâmetros *l* e *h* definem, respectivamente, a largura e a altura, em pixel, do retângulo.

4.1.1.2 strokeStyle

Esse atributo destina-se a definir a cor das linhas do elemento gráfico criado. Os valores possíveis são as cores válidas em CSS, tais como HSL, HSLA, RGB, RGBA (hex e porcentagem) e keywords. Não sendo definida a cor com uso desse atributo, o padrão é a cor preta #000000.

4.1.1.3 lineWidth

Esse atributo destina-se a definir a espessura das linhas, em pixel, do elemento gráfico criado. Os valores possíveis são os números inteiros. Não sendo definida a espessura com uso desse atributo, o padrão é zero e a linha não será visualizada.

4.1.1.4 fillRect(x, y, l, h)

Esse método destina-se a desenhar um retângulo em cor sólida. Os parâmetros x e y definem as coordenadas do canto superior esquerdo do retângulo e os parâmetros l e h definem, respectivamente, a largura e a altura, em pixel, do retângulo.

4.1.1.5 fillStyle

Esse atributo destina-se a definir a cor de preenchimento do elemento gráfico criado. Os valores possíveis são as cores válidas em CSS, tais como HSL, HSLA, RGB, RGBA (hex e porcentagem) e keywords.

Exemplo:

No exemplo a seguir desenharemos três retângulos usando os métodos e atributos descritos.

► JavaScript

```
1.      <script>
2.          var canvas = document.getElementById('quadro');
3.          var ctx = canvas.getContext('2d');
4.          ctx.strokeStyle = '#f00';
5.          ctx.lineWidth = 2;
6.          ctx.strokeRect(30,30, 80,40);
7.          ctx.strokeStyle = '#00f';
8.          ctx.lineWidth = 4;
9.          ctx.strokeRect(130,100, 140,90);
10.         ctx.strokeStyle = '#000';
11.         ctx.lineWidth = 2;
12.         ctx.fillStyle = 'rgba(0,255,0,0.4)';
13.         ctx.fillRect(150,60, 200,70);
14.         ctx.strokeRect(148,58, 204,74);
15.     </script>
```



[c4-canvas-ex4.html]

Código comentado:

Linha(s)	Descrição
Linha 4	Define a cor vermelha para as linhas do elemento gráfico a criar.
Linha 5	Define a espessura de 2px para as linhas do elemento gráfico a criar.
Linha 6	Cria um retângulo nas coordenadas x=30px e y=30px com 80px de largura e 40px de altura.
Linhas 4 a 6	Notar que o uso do método <code>strokeRect()</code> para a criação do retângulo foi definido na linha 6, depois de termos definido o estilo para a cor e espessura da linha. Essa condição é mandatória, define-se o gráfico depois da definição dos seus estilos.
Linhas 7 a 8	Cria um retângulo em outra posição de <code>canvas</code> nas mesmas condições anteriores.
Linhas 10 a 14	Cria um retângulo nas condições anteriores e preenche-o com um retângulo de cor sólida.
Linha 12	Usa o atributo <code>fillStyle</code> para definir uma cor e transparência para preenchimento de um retângulo.
Linha 13	Usa o método <code>fillRect()</code> para criar um retângulo sólido.

Notar que o elemento gráfico definido nas linhas 10 a 14 é formado pela criação de dois retângulos. Um com uso do método `strokeRect()` que criou o contorno e outro com uso do método `fillRect()` que criou o retângulo sólido dentro do contorno. Observe que as coordenadas e dimensões dos dois retângulos diferem para compensar a espessura da borda.

4.1.1.6 `clearRect(x, y, l, h)`

Esse método destina-se a apagar uma área retangular de `canvas`. No exemplo a seguir, desenhamos um retângulo e a seguir apagamos um retângulo menor nele contido. Exemplo:

► JavaScript

```
<script>
    var canvas = document.getElementById('quadro');
    var ctx = canvas.getContext('2d');
    ctx.strokeStyle = '#000';
    ctx.lineWidth = 2;
    ctx.fillStyle = '#060';
    ctx.fillRect(20,30, 300,150);
    ctx.strokeRect(18,28, 304,154);
    ctx.clearRect(50,60,180,100);
</script>
```



[c4-canvas-ex5.html]

4.1.1.7 shadow

Os atributos para sombreamento (shadow) dos elementos gráficos de `canvas` são: `shadowColor` que define a cor da sombra com uso de um valor CSS válido para cor; `shadowOffsetX` e `shadowOffsetY` que definem com uso de um número finito a espessura e deslocamento da sombra (em pixel) e `shadowBlur` que define com uso de um número maior ou igual a zero o desfoque (efeito blur) da cor da sombra. Os valores que definem `shadowOffset` podem ser positivos ou negativos determinando o sentido da sombra.

O exemplo a seguir é interativo e encontra-se disponível no site do livro. Nele você escolhe a cor da sombra e seus atributos e ao pressionar um botão visualiza a aplicação da sombra. No script mostrado a seguir não consta a programação JavaScript para fazer funcionar a interatividade do exemplo. Se você tiver interessado naquela programação, consulte o código-fonte da página.

Exemplo:

► JavaScript

```
<script>
    var canvas = document.getElementById('quadro');
    var ctx = canvas.getContext('2d');
    ctx.fillStyle = '#0f0';
    ctx.shadowColor = cód_hexa_cor; // você escolhe a cor
    ctx.shadowOffsetX = valor; // você escolhe o valor
    ctx.shadowOffsetY = valor; // você escolhe o valor
    ctx.shadowBlur = valor; // você escolhe o valor
    ctx.fillRect(40,20, 290,150);
</script>
```



[c4-canvas-ex6.html]

Estudaremos a seguir os atributos para a construção de caminhos (*path*) e subcaminhos (*subpaths*) em `canvas`. Um caminho é constituído de zero ou mais subcaminhos. Cada subcaminho é constituído por um ou mais pontos conectados por uma linha reta ou curva. Os subcaminhos podem ser fechados ou abertos. Diz-se que um subcaminho é fechado quando o ponto inicial está conectado ao ponto final por uma linha reta.

4.1.1.8 **beginPath()**

Esse método destina-se a resetar o path corrente e nativo do contexto `canvas` criando condições para se iniciar a construção de um novo path. Todo path ou subpath a ser construído deverá começar com esse método.

4.1.1.9 **moveTo(x,y)**

Esse método destina-se a posicionar a pena de desenho nas coordenadas x e y.

4.1.1.10 **lineTo(x,y)**

Esse método destina-se a desenhar uma linha reta desde a posição na qual se encontra a pena até as coordenadas x e y.

4.1.1.11 **stroke()**

Esse método destina-se a aplicar os estilos definidos pelos atributos de criação no caminho construído.

O exemplo a seguir cria uma linha poligonal com três segmentos de reta nas cores vermelha, azul e verde.

► JavaScript

```
<script>
    var ctx = canvas.getContext('2d');
    ctx.beginPath();
    ctx.strokeStyle = '#f00';
    ctx.lineWidth = 2;
    ctx.moveTo(20,20);
    ctx.lineTo(80,120);
    ctx.stroke();

    ctx.beginPath();
    ctx.strokeStyle = '#009';
    ctx.moveTo(80,120);
    ctx.lineTo(150,30);
    ctx.stroke();

    ctx.beginPath();
    ctx.strokeStyle = '#090';
```

```
ctx.moveTo(150,30);
ctx.lineTo(350,180);
ctx.stroke();
<script>
```



[c4-canvas-ex7.html]

4.1.1.12 `arcTo(x1, y1, x2, y2, r)`

Esse método destina-se a desenhar um arco com origem nas coordenadas x1 e y1 e término nas coordenadas x2 e y2 e com raio r.

O exemplo a seguir cria uma linha poligonal com três segmentos de arco nas cores vermelha, azul e verde.

► JavaScript

```
<script>
var canvas = document.getElementById('quadro');
var ctx = canvas.getContext('2d');

ctx.beginPath();
ctx.moveTo(20,50);
ctx.strokeStyle = '#f00';
ctx.arcTo(100, 50, 100, 130, 80);
ctx.stroke();

ctx.beginPath();
ctx.moveTo(100,130);
ctx.strokeStyle = '#009';
ctx.arcTo(200, 130, 200, 30, 100);
ctx.stroke();

ctx.beginPath();
ctx.moveTo(200,30);
ctx.strokeStyle = '#090';
ctx.arcTo(200, 190, 380, 190, 180);
ctx.stroke();
</script>
```



[c4-canvas-ex8.html]

4.1.1.13 `arc(x, y, r, anguloInicial, anguloFinal, sentido)`

Esse método destina-se a desenhar um arco com centro nas coordenadas x e y com raio r e ângulo inicial e final definidos em radianos (com sinal positivo ou negativo). O sentido do ângulo pode ser anti-horário (sentido trigonométrico positivo) ou horário (sentido trigonométrico negativo) e deve ser definido pelos booleanos true e false respectivamente. Lembramos que ângulos medidos em radianos na sintaxe JavaScript são medidos com o uso do objeto Math da linguagem. Por exemplo: Math.PI/2=90°, Math.PI=180°, 3/2*Math.PI=270°, Math.PI*2=360° e assim por diante.

O exemplo a seguir cria alguns arcos com uso desse método.

► JavaScript

```
<script>
    var canvas = document.getElementById('quadro');
    var ctx = canvas.getContext('2d');

    ctx.lineWidth = 4;      // espessura da linha para todos os arcos

    ctx.beginPath();
    ctx.strokeStyle = '#f00';
    ctx.arc(80, 100, 40, 0, Math.PI*2, true);
    ctx.stroke();

    ctx.beginPath();
    ctx.strokeStyle = '#090';
    ctx.arc(200, 100, 40, 0, Math.PI, true);
    ctx.stroke();

    ctx.beginPath();
    ctx.strokeStyle = '#00f';
    ctx.arc(200, 100, 40, 0, Math.PI, false);
    ctx.stroke();

    ctx.beginPath();
    ctx.strokeStyle = '#f90';
    ctx.arc(320, 100, 40, 0, 3/4*Math.PI, true);
    ctx.stroke();
</script>
```



[c4-canvas-ex9.html]

4.1.1.14 fill()

Esse método destina-se a preencher com uma cor sólida um caminho. Para demonstrar esse método vamos fazer o preenchimento dos arcos do exemplo anterior com uma cor amarela. Destacamos as alterações feitas no script anterior. Exemplo:

► JavaScript

```
<script>

    var canvas = document.getElementById('quadro');
    var ctx = canvas.getContext('2d');

    ctx.lineWidth = 4; // espessura da linha para todos os arcos

    ctx.beginPath();
    ctx.strokeStyle = '#f00';
    ctx.fillStyle = '#0ff';
    ctx.arc(80, 100, 40, 0, Math.PI*2, true);
    ctx.fill();
    ctx.stroke();

    ctx.beginPath();
    ctx.strokeStyle = '#090';
    ctx.fillStyle = '#996';
    ctx.arc(200, 100, 40, 0, Math.PI, true);
    ctx.fill();
    ctx.stroke();

    ctx.beginPath();
    ctx.strokeStyle = '#00f';
    ctx.fillStyle = '#c6c';
    ctx.arc(200, 100, 40, 0, Math.PI, false);
    ctx.fill();
    ctx.stroke();

    ctx.beginPath();
    ctx.strokeStyle = '#f90';
    ctx.fillStyle = '#0c0';
    ctx.arc(320, 100, 40, 0, 3/4*Math.PI, true);
    ctx.fill();
    ctx.stroke();

</script>
```



[c4-canvas-ex10.html]

Quando definimos uma espessura para a linha de contorno e um preenchimento para o caminho criado, a ordem de declaração dos métodos `stroke()` (para a linha) e `fill()` (para o preenchimento) é importante. Declarar `stroke()` depois de `fill()`, como fizemos no exemplo anterior, posiciona a linha sobre o preenchimento e, ao contrário, posiciona o preenchimento sobre a metade da espessura da linha. Faça uma cópia do exemplo anterior e inverta a ordem de declaração para visualizar a diferença.

4.1.1.15 `rect(x, y, l, h)`

Esse método destina-se a desenhar um retângulo. Os parâmetros `x` e `y` definem as coordenadas do canto superior esquerdo do retângulo e os parâmetros `l` e `h` definem, respectivamente, a largura e a altura, em pixel do retângulo.

Exemplo:

► JavaScript

```
<script>
    var canvas = document.getElementById('quadro');
    var ctx = canvas.getContext('2d');

    ctx.lineWidth = 4;

    ctx.beginPath();
    ctx.strokeStyle = '#f00';
    ctx.fillStyle = '#0ff';
    ctx.rect(50, 60, 200, 120);
    ctx.fill();
    ctx.stroke();
</script>
```



[c4-canvas-ex11.html]

4.1.1.16 `closePath()`

Esse método destina-se a fazer o fechamento de um caminho. Por exemplo: criar uma linha poligonal ou um arco aberto e declarar `closePath()` faz com que os pontos inicial e final da criação sejam unidos por uma linha reta. Observe o exemplo a seguir no qual fizemos o fechamento de uma linha poligonal de três segmentos de reta e de um arco de circunferência.

Exemplo:

► **JavaScript**

```
<script>

var canvas = document.getElementById('quadro');
var ctx = canvas.getContext('2d');
ctx.lineWidth = 2;

ctx.beginPath();
ctx.moveTo(20,20);
ctx.strokeStyle = '#f00';
ctx.lineTo(80,190);
ctx.lineTo(180,160);
ctx.lineTo(220,20);
ctx.closePath();
ctx.stroke();

ctx.beginPath();
ctx.strokeStyle = '#090';
ctx.arc(300, 100, 80, 0, -3/2*Math.PI, true);
ctx.closePath();
ctx.stroke();

</script>
```



[c4-canvas-ex12.html]

4.1.1.17 globalAlpha

Esse atributo destina-se a definir uma transparência e ser aplicada em todos os elementos de `canvas`. O valor para esse atributo varia de 0 a 1, sendo 0 transparência total (invisibilidade) e 1 opacidade total (nada transparente).

4.1.1.18 lineJoin

Esse atributo destina-se a definir a forma como as linhas se unem. Os valores possíveis são: `miter`, `bevel` e `round`.

4.1.1.19 lineCap

Esse atributo destina-se a definir a forma de acabamento das extremidades das linhas. Os valores possíveis são: `butt`, `round` e `square`.

O exemplo a seguir encontra-se no site do livro e é interativo para você visualizar os efeitos desses três atributos

► JavaScript

```
<script>
    var canvas = document.getElementById('quadro');
    var ctx = canvas.getContext('2d');

    ctx.lineJoin = escolha;
    ctx.lineCap = escolha;
    ctx.globalAlpha = escolha;

    ctx.lineWidth = 50;
    ctx.beginPath();
    ctx.moveTo(100,40);
    ctx.strokeStyle = '#f00';
    ctx.lineTo(100,130);
    ctx.lineTo(350,130);
    ctx.stroke();
</script>
```



[c4-canvas-ex13.html]

As especificações para a HTML5 preveem dois tipos de gradiente, linear e radial. Para criar gradientes são previstos três métodos, como mostrado a seguir.

4.1.1.20 `createLinearGradient(x,y,x1,y1)`

Esse método destina-se a criar um gradiente linear. As coordenadas x,y e x1,y1 definem os pontos nos quais começa e termina respectivamente o gradiente.

4.1.1.21 `createRadialGradient(x,y,r,x1,y1,r1)`

Esse método destina-se a criar um gradiente radial. As coordenadas x,y e x1,y1 definem os centros dos círculos nos quais começam e terminam respectivamente o gradiente e os parâmetros r e r1 seus raios.

4.1.1.22 `addColorStop(offset, cor)`

Esse método destina-se a definir como as cores do gradiente serão distribuídas. O valor do parâmetro offset varia de 0 a 1. Zero define a cor de início e 1 a cor do final

do gradiente. Em um gradiente de duas cores, esse método será aplicado duas vezes, uma com offset 0 e cor inicial e outra com offset 1 e cor final. Podemos criar um gradiente de várias cores definindo várias vezes esse método com offset crescente, por exemplo: 0.1, 0.2, 0.3, 0.4, 0.8, 1 (seis cores).

No exemplo a seguir, que se encontra no site do livro, você poderá visualizar os efeitos da aplicação do método para gradientes lineares.

► JavaScript

```
<script>
    var canvas = document.getElementById('quadro');
    var ctx = canvas.getContext('2d');

    // cria um gradiente de vermelho para azul
    gradient = ctx.createLinearGradient(10,10,250,100);
    gradient.addColorStop(0, '#f00');
    gradient.addColorStop(1, '#00f');

    // cria um gradiente de verde para amarelo e azul claro
    gradient1 = ctx.createLinearGradient(330,20,330,160);
    gradient1.addColorStop(0, '#090');
    gradient1.addColorStop(0.5, '#ff9');
    gradient1.addColorStop(1, '#9cf');

    // cria um gradiente de laranja para amarelo claro
    gradient2 = ctx.createLinearGradient(60,120,180,120);
    gradient2.addColorStop(0, '#f90');
    gradient2.addColorStop(1, '#ffc');

    // aplica os gradientes em dois retângulos e um círculo
    ctx.fillStyle = gradient;
    ctx.fillRect(10,10,250,100);

    ctx.fillStyle = gradient1;
    ctx.fillRect(280,20,100,160);

    ctx.fillStyle = gradient2;
    ctx.arc(120,120,60,0,2*Math.PI, true);
    ctx.fill();
</script>
```



[c4-canvas-ex14.html]

No exemplo a seguir, que se encontra no site do livro, você poderá visualizar os efeitos da aplicação do método para gradientes radiais.

Exemplo:

► JavaScript

```
<script>

    var canvas = document.getElementById('quadro');
    var ctx = canvas.getContext('2d');

    gradient = ctx.createRadialGradient(130,60,10,130,60,160);
    gradient.addColorStop(0, '#f00');
    gradient.addColorStop(1, '#00f');

    gradient1 = ctx.createRadialGradient(330,50,5,330,50,130);
    gradient1.addColorStop(0, '#090');
    gradient1.addColorStop(0.5, '#fd8');
    gradient1.addColorStop(1, '#9cf');

    gradient2 = ctx.createRadialGradient(100,100,5,100,100,60);
    gradient2.addColorStop(0, '#f90');
    gradient2.addColorStop(1, '#f5f5f5');

    ctx.fillStyle = gradient;
    ctx.fillRect(10,10,250,100);

    ctx.fillStyle = gradient1;
    ctx.fillRect(280,20,100,160);

    ctx.fillStyle = gradient2;
    ctx.arc(120,120,60,0,2*Math.PI, true);
    ctx.fill();
</script>
```



[c4-canvas-ex15.html]

4.1.1.23 createPattern(*imagem, repetição*)

Esse método destina-se a inserir uma imagem como preenchimento de um contorno. O parâmetro imagem define a imagem a ser usada como preenchimento e o parâmetro repetição define como a forma gráfica se repetirá. A imagem poderá ser um elemento img, um elemento canvas ou um elemento video e os valores para o parâmetro são: repeat, repeat-x, repeat-y e no-repeat.

No exemplo a seguir, que se encontra no site do livro, você poderá visualizar os efeitos da aplicação desse método. No retângulo da esquerda usamos o valor `no-repeat` para o parâmetro repetição e no da direita `repeat`. Os valores `repeat-x` e `repeat-y` causam o mesmo efeito dos seus equivalentes nas CSS.

► JavaScript

```
<script>
    var canvas = document.getElementById('quadro');
    var ctx = canvas.getContext('2d');
    var imagem = new Image();
    imagem.src = 'flor.png';

    imagem.onload = function() { // importante executar o desenho depois de a imagem ter sido carregada
        var pattern = ctx.createPattern(imagem, 'no-repeat');
        var pattern1 = ctx.createPattern(imagem, 'repeat');
        ctx.lineWidth = 4;
        ctx.strokeStyle = '#f00';

        ctx.fillStyle = pattern;
        ctx.fillRect(2,2,190,196);
        ctx.strokeRect(2,2,190,196);

        ctx.fillStyle = pattern1;
        ctx.fillRect(200,2,198,196);
        ctx.strokeRect(200,2,198,196);
    }
</script>
```



[c4-canvas-ex16.html]

4.1.1.24 globalCompositeOperation

Esse atributo destina-se a definir como uma imagem será desenhada em relação ao estado atual do desenho em `canvas`, ou, ainda, como será a composição das duas imagens.

Para melhor entendimento do funcionamento dessa propriedade, vamos criar um exemplo mostrando duas imagens: um retângulo e um círculo. Considere que o retângulo já existe em `canvas`, assim ele representa o que denominamos de estado atual de desenho em `canvas`. A especificação denomina essa imagem de `destination`. A imagem do círculo é denominada pela especificação de `source`.

A composição das imagens `destination` e `source` pode ser feita de onze maneiras diferentes. Por padrão, a imagem `source` é colocada sobre a imagem `destination`, ou seja, o empilhamento das imagens em `canvas` se dá com a imagem declarada por último no código sobre as imagens anteriores. Em resumo, esse atributo interfere não só na ordem de empilhamento, mas também na integração entre as duas imagens.

Os valores possíveis para esse atributo são: `source-atop`, `source-in`, `source-over` (default), `source-out`, `destination-atop`, `destination-in`, `destination-over`, `destination-out`, `lighter`, `copy` e `xor`.

A especificação prevê ainda o valor `vendorName-operationName` para a aplicação de composições proprietárias.

No exemplo a seguir, que se encontra no site do livro, você poderá visualizar os efeitos da aplicação desse atributo em uma página interativa na qual você escolhe um valor e visualiza o seu efeito de composição. Não constam do script os comandos para a interatividade.

► JavaScript

```
<script>
    var canvas = document.getElementById('quadro');
    var ctx = canvas.getContext('2d');
    ctx.fillStyle = '#900';
    ctx.fillRect(60,60,220,130);

    ctx.globalCompositeOperation = escolha;

    ctx.fillStyle = '#090';
    ctx.beginPath();
    ctx.arc(300,100,80,0,2*Math.PI,true);
    ctx.fill();
</script>
```



[c4-canvas-ex17.html]

Transformações visam a movimentar `canvas`. Os métodos para transformações permitem deslocar `canvas` horizontal e verticalmente, rotacionar `canvas`, aplicar escala (reduzir ou aumentar) a unidade de medida de `canvas` e modificar a matriz de coordenadas de `canvas`. Antes de estudarmos esses métodos, vamos apresentar os métodos `save()` e `restore()` que normalmente são usados em conjunto com os métodos de transformação e estilização.

Quando se aplicam métodos de transformação em `canvas`, alteramos as referências. Em criações mais complexas, tanto as transformações quanto as definições de estilos podem exigir que tenhamos de reverter as configurações iniciais para criarmos novos desenhos. Isso significa necessidade de se escrever mais código.

Os métodos `save()` e `restore()` visam a eliminar a necessidade de códigos para a reversão, pois armazenam e restabelecem estados anteriores de `canvas`. A melhor maneira de entender o funcionamento desses métodos é com a análise de um exemplo prático.

Antes, porém, vamos examinar o esquema gráfico mostrado na figura 4.2 que esclarece o mecanismo de aplicação de estilos e transformações com uso desses métodos.

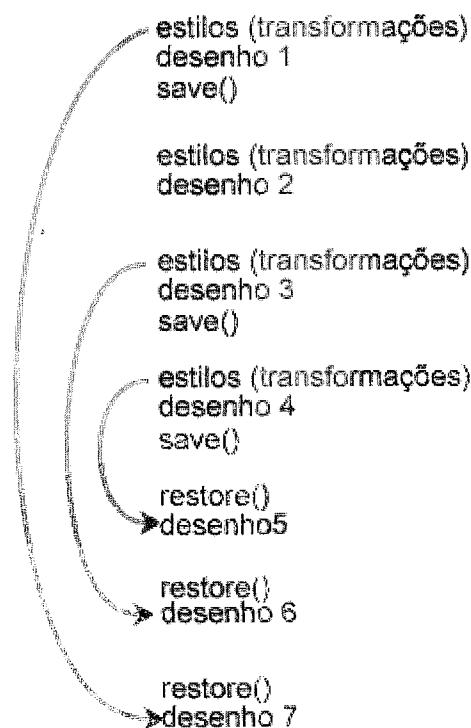


Figura 4.2 – Mecanismo `save()` `restore()`.

Podemos usar quantos métodos `save()` quisermos em nosso script para armazenar estilos, mas, é lógico, devemos posicioná-los no código após declarações de estilos (ou transformações) a serem armazenados.

Os estilos (ou transformações) vão sendo armazenados como se estivessem sendo “empilhados” com os mais recentes no código na parte superior da pilha. Formada a pilha de estilos, podemos chamá-los e aplicá-los a novos desenhos com uso do método `restore()`, que deverá ser evocado antes do script de desenho.

Cada `restore()` no código vai desempilhando os estilos a aplicar na ordem inversa à que eles foram empilhados.

No diagrama da figura 4.2 são armazenados em uma pilha os estilos dos desenhos 1, 3 e 4. A aplicação de estilo no desenho 5 faz-se pelo método `restore()` que busca o último estilo armazenado. Assim, ao desenho 5 é aplicado o estilo do desenho 4. O próximo `restore()` é para o desenho 6, que busca o estilo do desenho 3 (seguinte na pilha), e assim por diante.

No exemplo existente no site do livro armazenamos e restabelecemos declarações de estilo. O exemplo contém dois `canvas` com scripts de desenho idênticos, exceto pelo fato de que no primeiro `canvas` não usamos os métodos `save()` e `restore()` e no segundo `canvas` usamos.

Em ambos desenharmos cinco retângulos preenchidos com cor sólida. O primeiro com a cor vermelha, o segundo verde, o terceiro azul e os dois últimos sem estilização de cor.

No primeiro `canvas` os dois últimos retângulos assumem a última declaração de estilo para a cor que é a do terceiro retângulo, ou seja, cor azul. Assim, as cores dos retângulos são vermelha, verde, azul, azul e azul.

No segundo `canvas` usamos duas vezes o método `save()` para armazenar os estilos do primeiro (vermelha) e segundo (verde) retângulos.

► JavaScript

```
<script>
    var canvas = document.getElementById('quadro');
    var ctx = canvas.getContext('2d');      // canvas
    ctx.fillStyle = 'red';
    ctx.fillRect(40,20,40,140);
    ctx.fillStyle = 'green';
    ctx.fillRect(110,20,40,140);
    ctx.fillStyle = 'blue';
    ctx.fillRect(180,20,40,140);
    ctx.fillRect(250,20,40,140);
    ctx.fillRect(320,20,40,140);
```

```
var canvas1 = document.getElementById('quadro1');
var ctx1 = canvas1.getContext('2d');      // canvas 1

ctx1.fillStyle = 'red';
ctx1.fillRect(40,20,40,140);
ctx1.save();

ctx1.fillStyle = 'green';
ctx1.fillRect(110,20,40,140);
ctx1.save();

ctx1.fillStyle = 'blue';
ctx1.fillRect(180,20,40,140);

ctx1.restore();
ctx1.fillRect(250,20,40,140);

ctx1.restore();
ctx1.fillRect(320,20,40,140);
</script>
```



[c4-canvas-ex18.html]

4.1.1.25 scale(x, y)

Esse método destina-se a escalar as dimensões x e/ou y de todos os gráficos inseridos em `canvas`. A área de desenho (ou seja, a largura e a altura de `canvas`) não é afetada ou escalável. Aplicar esse método aumenta ou diminui não somente as dimensões, mas também as coordenadas proporcionalmente aos valores dos parâmetros x e y passados ao método. O valor do parâmetro é o fator de multiplicação.

Há aumento quando os parâmetros são números positivos inteiros maiores do que zero. Assim, definir `x=2` resulta em duplicação das dimensões horizontais, `x=5` quintuplicação e assim por diante.

Para diminuir, os parâmetros devem ser definidos com números maiores que zero e menores que um. Assim, `x=0.3` resulta em redução de 70% nas dimensões horizontais, `x=0.5` em redução pela metade e assim por diante.

O exemplo a seguir encontra-se disponível no site do livro. É interativo e você poderá testar e verificar o funcionamento desse método escolhendo alguns valores para os fatores de multiplicação. Nele um quadro de 90 x 90px é escalável nas suas

dimensões horizontal e vertical. Notar como o fator de multiplicação se aplica às coordenadas de origem do quadrado. Para ressaltar os deslocamentos, criando uma referência para as dimensões iniciais, foi desenhado um quadrado cinza fixo com aquelas dimensões.

► JavaScript

```
<script>
    var canvas = document.getElementById('quadro');
    var ctx = canvas.getContext('2d');
    ctx.fillStyle = '#ddd';
    ctx.fillRect(10,5,90,90); // quadrado referência

    ctx.scale(escolha,escolha);
    ctx.fillStyle = 'rgba(0,100,0,0.5)';
    ctx.fillRect(10,5,90,90);
</script>
```



[c4-canvas-ex19.html]

4.1.1.26 translate(x,y)

Esse método destina-se a deslocar de um valor x e/ou y em pixel as coordenadas de origem de `canvas`, o que equivale ao deslocamento de todos os gráficos nele inseridos. A área de desenho (ou seja, a largura e a altura de `canvas`) não é afetada e não sofrerá movimentação. Aplicar esse método faz com que haja um movimento de translação.

O valor do parâmetro x define uma translação na direção horizontal. Valores positivos causam deslocamento para a direita e negativos para a esquerda.

O valor do parâmetro y define uma translação na direção vertical. Valores positivos causam translação para baixo e negativos para cima.

O exemplo a seguir encontra-se disponível no site do livro. É interativo e você poderá testar e verificar o funcionamento desse método escolhendo alguns valores para a translação. Nele um quadrado de 20 x 20px é deslocado nas suas dimensões horizontal e/ou vertical. Para ressaltar os deslocamentos, criamos uma referência da posição inicial desenhando um quadrado cinza fixo naquela posição.

► JavaScript

```
<script>
    var canvas = document.getElementById('quadro');
    var ctx = canvas.getContext('2d');
```

```
ctx.fillStyle = '#ddd';
ctx.fillRect(190,90,20,20); // quadrado referência

ctx.translate(escolha,escolha);
ctx.fillStyle = 'rgba(0,100,0,0.5)';
ctx.fillRect(190,90,20,20);

</script>
```



[c4-canvas-ex20.html]

4.1.1.27 **rotate(ângulo)**

Esse método destina-se a transformar o desenho, alterando sua posição como se tivéssemos rotacionado `canvas`. A rotação é definida pelo parâmetro ângulo, em radianos, e segue o sentido trigonométrico, ou seja, ângulo positivo causa rotação no sentido anti-horário e negativo no sentido horário. O centro de rotação é o ponto de origem de `canvas`. A figura 4.3 esclarece o efeito de uma rotação positiva igual a `Math.PI/6` (30 graus).

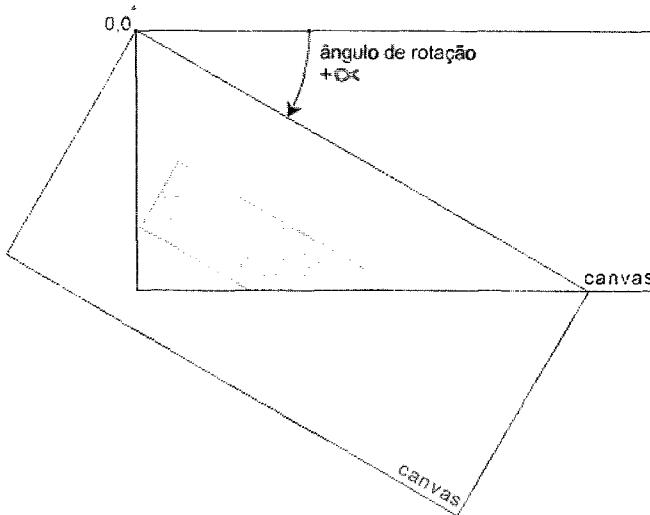


Figura 4.3 – Mecanismo para `rotate()`.

O exemplo a seguir encontra-se disponível no site do livro. É interativo e você poderá testar e verificar o funcionamento desse método escolhendo alguns valores para a rotação. Nele um retângulo de 260 x 60px está posicionado inicialmente no centro de `canvas` e, mediante a escolha de um ângulo de rotação, você poderá visualizar o efeito do método. Para ressaltar a rotação, criamos uma referência à posição inicial desenhando um retângulo cinza fixo naquela posição.

► JavaScript

```
<script>
    var canvas = document.getElementById('quadro');
    var ctx = canvas.getContext('2d');
    ctx.fillStyle = '#ddd';
    ctx.fillRect(70,70,260,60); // retângulo referência
    var angulo = escolha;
    ctx.rotate(angulo);
    ctx.fillStyle = 'rgba(0,100,0,0.5)';
    ctx.fillRect(70,70,260,60);
</script>
```



[c4-canvas-ex21.html]

O método `drawImage()` destina-se a inserir imagens em `canvas`. A imagem a ser inserida pode ser o conteúdo de um elemento `img`, `canvas` ou `video`. Para inserir a imagem, é necessário que ela tenha sido completamente carregada. Esse método pode ser evocado de três maneiras diferentes, como mostrado a seguir.

4.1.1.28 `drawImage(imagem, dx, dy)`

Insere a imagem definida no parâmetro `imagem` com seu canto superior esquerdo nas coordenadas definidas nos parâmetros `dx` e `dy`. O exemplo a seguir esclarece o uso desse método.

► JavaScript

```
<script>
    var canvas = document.getElementById('quadro');
    var ctx = canvas.getContext('2d');
    var elImagen = document.createElement('img');
    elImagen.src = 'flor.png';
    ctx.drawImage(elImagen, 20, 50); // insere a imagem elImagen nas coordenadas 20, 50
</script>
```



[c4-canvas-ex22.html]

4.1.1.29 `drawImage(imagem, dx, dy, dl, dh)`

Insere a imagem definida no parâmetro `imagem` com seu canto superior esquerdo nas coordenadas definidas nos parâmetros `dx` e `dy` redimensionando a imagem para as dimensões em pixel `dl` de largura por `dh` de altura. O exemplo a seguir esclarece o uso desse método.

► JavaScript

```
<script>
    var canvas = document.getElementById('quadro');
    var ctx = canvas.getContext('2d');
    var elImagen = document.createElement('img');
    elImagen.src = 'flor.png';
    ctx.drawImage(elImagen, 20, 50, 100, 140); // insere a imagem elImagen com 100x140px
                                                // nas coordenadas 20, 50
</script>
```



[c4-canvas-ex23.html]

4.1.1.30 drawImage(imagem, sx, sy, sl, sh, dx, dy, dl, dh)

Recorta a imagem definida no parâmetro imagem a partir das coordenadas sx, sy com dimensões sl, sh e insere com seu canto superior esquerdo nas coordenadas definidas nos parâmetros dx e dy redimensionando a imagem para as dimensões em pixel dl de largura por dh de altura. O diagrama mostrado na figura 4.4 esclarece o recorte da imagem. Para o recorte mostrado no diagrama, foram definidos os seguintes parâmetros: `drawImage(elImagen,10,5,25,30,100,100,50,60);`

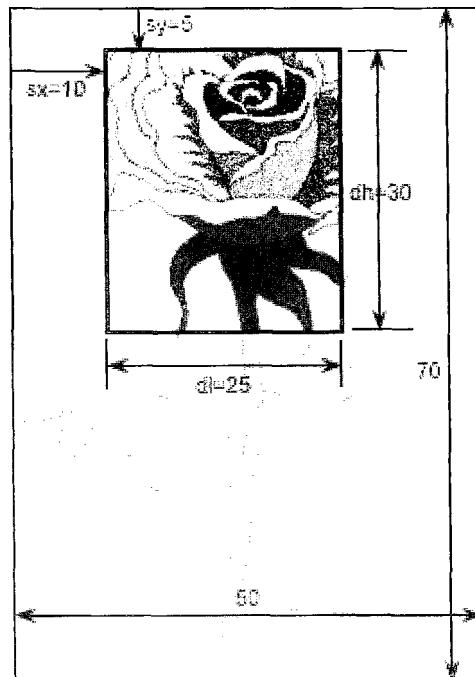


Figura 4.4 – Recorte de imagem.

As dimensões originais da imagem são: 50 x 70px. O recorte foi feito a partir do ponto localizado em 10, 5 na imagem e definiu-se uma largura de 25 pixel e uma altura de 30px. A imagem assim recortada tem as dimensões de 25 x 30px e será inserida nas coordenadas 100, 100 de canvas. As dimensões da imagem recortada para inserção foram definidas em 50 x 60px (o dobro de 25 x 30px originais). Dessa forma, a imagem será redimensionada para o dobro do seu tamanho.

O exemplo a seguir esclarece o uso desse método para a imagem mostrada na figura 4.4.

► JavaScript

```
<script>
    var canvas = document.getElementById('quadro');
    var ctx = canvas.getContext('2d');
    var elImagen = document.createElement('img');
    elImagen.src = 'flor.png';
    // recorta a imagem elImagen e insere com o dobro do tamanho nas coordenadas 100,100
    ctx.drawImage(elImagen, 10, 5, 25, 30, 100, 100, 50, 60);
</script>
```



[c4-canvas-ex24.html]

4.1.1.31 font

Esse atributo define as características da fonte a ser usada nos textos a serem inseridos em canvas. A sintaxe para a definição das características é a mesma usada nas CSS e mostrada a seguir a título de recordação.

```
font: [font-style || font-variant || font-weight]? font-size[/line-height]? font-family |
caption | icon | menu | message-box | small-caption | status-bar | inherit
```

Exemplo: ctx.font = 'italic small-caps bold 40px Georgia, sans-serif'

É natural que, em um primeiro momento, chegemos à conclusão de que é possível estilizar os textos de canvas com CSS declarando, por exemplo, a seguinte regra CSS.

```
canvas {
    text-align: justify;
    text-transform: lowercase;
    color: red;
}
```

Isso não vai funcionar. As características de apresentação dos textos e fontes inseridas em `canvas` são determinadas pelos métodos e atributos da API e não por CSS. A única estilização em `canvas` que interfere com a fonte inserida é o tamanho de fonte declarada para `canvas`, pois é nesse tamanho que será tomada a base para o cálculo da fonte declarada com tamanho relativo. Por exemplo:

```
canvas {  
    font-size: 10px; // base para fontes relativas  
}  
  
ctx.font = ctx.font = '250% Arial, serif'; // a fonte em canvas será com tamanho igual a  
// 250% de 10px = 25px
```

4.1.1.32 `fillText(texto, x, y, [cMax])`

Esse método destina-se a definir o preenchimento do texto a inserir. Admite três parâmetros obrigatórios e um parâmetro facultativo. O parâmetro `texto` é uma string e define o texto a ser inserido, os parâmetros `x` e `y` definem as coordenadas de inserção do texto. O parâmetro facultativo `cMax` define o comprimento máximo do texto e, quando definido para um valor numérico menor do comprimento-padrão do texto, faz com que ele sofra um efeito de compressão na horizontal, diminuindo o espaçamento e a largura das letras.

4.1.1.33 `strokeText(texto, x, y, [cMax])`

Esse método destina-se a definir o contorno do texto a inserir. Admite três parâmetros obrigatórios e um parâmetro facultativo. O parâmetro `texto` é uma string e define o texto a ser inserido, os parâmetros `x` e `y` definem as coordenadas de inserção do texto. O parâmetro facultativo `cMax` define o comprimento máximo do texto e, quando definido para um valor numérico menor do comprimento-padrão do texto, faz com que ele sofra um efeito de compressão na horizontal, diminuindo o espaçamento e a largura das letras.

O exemplo a seguir esclarece o uso desses métodos. Foram criados e posicionados dois textos: o primeiro com uma cor sólida verde e um contorno na cor azul e o segundo somente o contorno na cor vermelha. Notar que para o segundo texto diminuímos a largura com uso do parâmetro facultativo `cMax`.

► JavaScript

```
<script>  
var canvas = document.getElementById('quadro');  
var ctx = canvas.getContext('2d');
```

```

ctx.font = 'italic small-caps bold 40px Georgia, sans-serif';
ctx.fillStyle = '#0f0';
ctx.strokeStyle = '#009';
ctx.fillText('Livro html5', 20, 50);
ctx.strokeText('Livro html5', 20, 50);

ctx.font = '100px Arial, serif';
ctx.lineWidth = 2;
ctx.strokeStyle = '#c30';
ctx.strokeText('Livro html5 do Maujor', 60, 160, 200);
</script>

```



[c4-canvas-ex25.html]

4.1.1.34 textAlign

Esse atributo destina-se a definir o alinhamento do texto a inserir em relação ao ponto de inserção. Admite os valores `start`, `end`, `left`, `right` e `center`.

O exemplo a seguir é interativo e encontra-se no site do livro. Nele você poderá visualizar o efeito desse atributo escolhendo um dos valores possíveis para ele.

► JavaScript

```

<script>
var canvas = document.getElementById('quadro');
var ctx = canvas.getContext('2d');
// Texto referência em cinza
ctx.font = '50px serif';
ctx.fillStyle = '#ccc';
ctx.fillText('Maujor', 200, 100);
// Ponto referência em azul
ctx.fillStyle = '#006';
ctx.arc(200, 100, 3, 0, 2*Math.PI, true);
ctx.fill();
// Texto a alinhar em vermelho
ctx.font = '50px serif';
ctx.fillStyle = 'rgba(200, 0, 0, 0.6)';
ctx.textAlign = escolha;
ctx.fillText('Maujor', 200, 100);
</script>

```



[c4-canvas-ex26.html]

4.1.1.35 `textBaseline`

Esse atributo destina-se a definir o alinhamento do texto a inserir em relação às linhas-base do texto. Admite os valores `top`, `hanging`, `ideographic`, `middle`, `alphabetic` e `bottom`. O valor-padrão é `alphabetic`.

O exemplo a seguir é interativo e encontra-se no site do livro. Nele você poderá visualizar o efeito desse atributo escolhendo um dos valores possíveis para ele.

► JavaScript

```
<script>
    var canvas = document.getElementById('quadro');
    var ctx = canvas.getContext('2d');
    // Texto referência em cinza
    ctx.font = '50px serif';
    ctx.fillStyle = '#ccc';
    ctx.fillText('Maujor', 200, 100);
    // linha de referência em azul
    ctx.moveTo(0,101);
    ctx.lineTo(400,101);
    ctx.strokeStyle = '#00f';
    ctx.stroke();
    // Texto a alinhar em vermelho
    ctx.font = '50px serif';
    ctx.fillStyle = 'rgba(200, 0, 0, 0.6)';
    ctx.textBaseline = escolha;
    ctx.fillText('Maujor', 200, 100);
</script>
```



[c4-canvas-ex27.html]

A figura 4.5 retirada das especificações para a HTML5 mostra as linhas de base para alinhamento de texto.

Encerramos este capítulo transcrevendo um exemplo de animação em `canvas` constante das especificações para HTML5. Fizemos pequenas adaptações no script original para compatibilizar com nosso layout dos exemplos. Visualize o arquivo disponível no site do livro e consulte o código-fonte para maiores detalhes.



[c4-canvas-ex28.html]

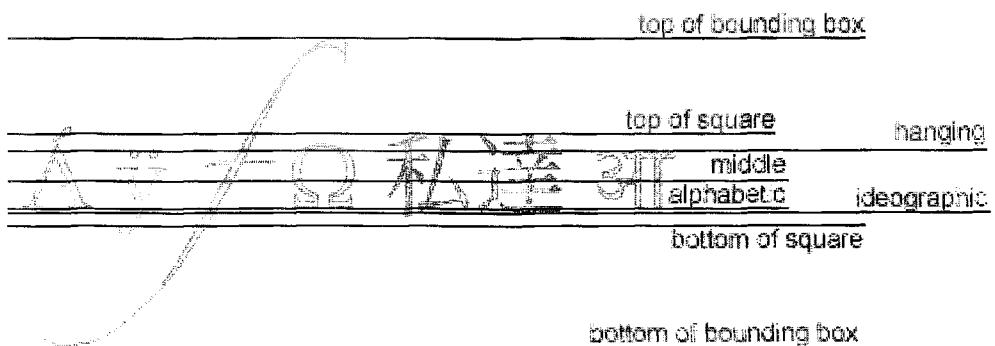
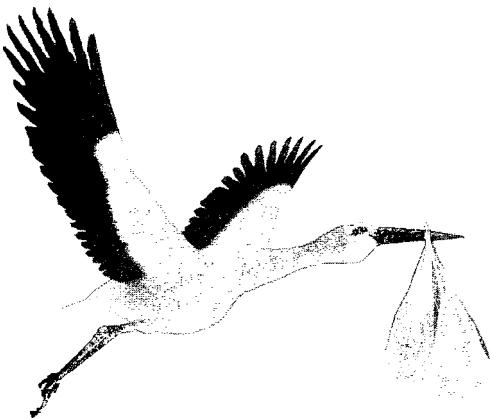


Figura 4.5 – Linhas de base para textos.

O elemento `canvas` permite criações e animações cujo limite é a capacidade do desenvolvedor em programar JavaScript. Existem centenas de criações disponíveis para consulta na Internet. Faça uma busca pela palavra-chave “`canvas examples`” e você encontrará exemplos surpreendentes e fantásticos.



CAPÍTULO 5

Atributos HTML

Neste capítulo estudaremos os atributos para os elementos HTML relacionando e comentando cada um deles, em particular, os novos atributos introduzidos pela HTML5. Definiremos cada atributo, sua finalidade e, sempre que necessário, apresentaremos um exemplo prático de uso. A relação de atributos para os elementos da HTML foi ampliada e revisada na HTML5. Novos atributos foram criados, vários atributos existentes foram redefinidos e muitos elementos tiveram a lista de atributos suportados ampliada. A HTML 4.01 prevê 118 atributos e a HTML5 expandiu a lista.

5.1 Atributos globais

Atributos globais são aqueles comuns a todos os elementos da HTML, ou seja, podem ser definidos para qualquer elemento da linguagem. Os atributos comuns da HTML5 foram mostrados no capítulo 2 e são também relacionados a seguir.

5.1.1 accesskey

Esse atributo destina-se a definir uma tecla de atalho para ativar ou dar o foco ao elemento quando o usuário navega com auxílio do teclado.

O valor desse atributo é sensível ao tamanho de caixa (case sensitive) e constituído por uma lista de caracteres Unicode de teclado (tecla do teclado) separados por espaço. Nas versões anteriores da HTML esse atributo é definido para uso com apenas sete elementos e seu valor deve ser um caractere. Na HTML5 tornou-se um atributo global e admite uma lista de valores. Os exemplos a seguir esclarecem seu uso.

Atualmente a implementação de como acionar (combinação de teclas) um `accesskey` varia de acordo com o navegador. A especificação para a HTML5 preconiza que a combinação de teclas para acionar o atributo `accesskey` deve ser **Ctrl + Alt + tecla** e que o caractere do valor do atributo é case sensitive. O exemplo mostrado a seguir funciona no Firefox e Chrome com a combinação **Shift + Alt + tecla**.

Todavia, convém ressaltar que nenhum navegador implementa esse atributo para todos os elementos da HTML e o Exemplo 1 só funciona porque o elemento `a` (âncora) é um dos sete elementos que suportam o atributo segundo as versões anteriores da HTML. O exemplo 2 não funciona em nenhum navegador.

Exemplo 1:

Nesse exemplo o valor do atributo `accesskey` é um caractere. Nele o acesso via teclado aos links de uma lista de navegação é feito com uso de uma letra maiúscula. Por exemplo: para acessar o site do Maujor, pressione a tecla representativa da letra “M” estando as teclas **Shift + Alt** previamente pressionadas.

► HTML

```
<nav>
  <p>
    <a href="http://maujor.com" accesskey="M">Site do Maujor</a> |
    <a href="http://w3.org" accesskey="W">Site W3C</a> |
    <a href="http://google.com.br" accesskey="G">Google</a>
  </p>
</nav>
```



[c5-accesskey-ex1.html]

Exemplo 2:

Já dissemos que a especificação para a HTML5 prevê que o valor do atributo `accesskey` pode ser uma lista de caracteres separados por espaço. Esse exemplo ilustra o uso de uma lista de dois caracteres separados por espaço.

Para dispositivos de usuário que utilizam teclados completos, a tecla de atalho é definida pelo primeiro caractere, sendo o segundo caractere destinado a dispositivos de usuário com teclados reduzidos, tais como aqueles de certos dispositivos móveis que só possuem teclado numérico.

No exemplo a seguir teclados com a tecla da letra “S” usam essa como `accesskey` e teclados numéricos usam a tecla para o número 2.

► HTML

```
<form action="/busca" method="get">
  <label>Buscar: <input type="search" name="q" accesskey="s 2"></label>
  <input type="submit">
</form>
```



[c5-accesskey-ex2.html]

5.1.2 class

Esse atributo destina-se a definir uma classe para o elemento com a finalidade de servir de referência para estilização e scripts.

O valor desse atributo é constituído por uma lista de strings separadas por espaço, não havendo qualquer restrição de caracteres para a formação das strings. Nas versões anteriores da HTML, esse atributo é definido para todos os elementos, exceto os elementos `base`, `basefont`, `head`, `html`, `meta`, `param`, `script`, `style` e `title`. Na HTML5 tornou-se um atributo global podendo ser usado para todos os elementos. Por exemplo:

```
<script type="text/javascript" class="script-um">      // inválido em HTML4 e XHTML1
<script type="text/javascript" class="script-um">      // válido em HTML5
```

5.1.3 contenteditable

Esse atributo foi inventado pela Microsoft e implementado em seus navegadores desde a versão 5 e que agora foi reconhecido e adotado pela HTML5. Admite os valores `true`, `false` e `inherit` que definem se o conteúdo do elemento é editável ou não ou se sua condição de edição é herdada. A edição se faz no próprio agente de usuário que renderiza o conteúdo.

O exemplo a seguir exemplifica o uso desse atributo e seus respectivos valores. Nele você poderá testar a edição dos conteúdos.

```
<article contenteditable="true">
  <h2>Texto de um cabeçalho nível 2 editável</h2>
  <p>Parágrafo editável</p>
  <p contenteditable="false">Parágrafo <b>não</b> editável</p>
</article>
```



[c5-contenteditable.html]

5.1.4 contextmenu

Esse atributo destina-se a associar o elemento ao qual foi aplicado com um menu de contexto criado com uso do elemento `menu`.

A associação se faz atribuindo um `id` ao elemento `menu` com valor igual ao valor definido para esse atributo.

Esse é um atributo não existente nas versões anteriores da HTML, tendo sido criado pela especificação da HTML5.

O exemplo a seguir, extraído das especificações para a HTML5, esclarece o uso desse atributo associando um elemento `input` ao menu de contexto para preenchimento de campos de um formulário.

```
<form name="npc">  
    <label>Nome do personagem: <input name=char type=text contextmenu=personagem required></label>  
    <menu type=context id=personagem>  
        <command label="Sortear um nome"  
            onclick="document.forms.npc.elements.char.value = getRandomName()">  
        <command label="Características do personagem"  
            onclick="prefillFields(document.forms.npc.elements.char.value)">  
    </menu>  
</form>
```

O exemplo mostra um formulário em uma interface de um jogo para escolha de um personagem controlado pela máquina (non-player character). O menu de contexto oferece a possibilidade de escolha aleatória e de exibição das características do personagem escolhido.

5.1.5 dir

Esse atributo destina-se a definir o sentido de escrita dos textos. Os valores possíveis para esse atributo são: `ltr`, `rtl` e `auto`.

O valor `ltr` é o valor-padrão para a escrita ocidental na qual o texto flui no sentido da esquerda para a direita (`left to right`).

O valor `rtl` é o valor-padrão para a escrita oriental na qual o texto flui no sentido da direita para a esquerda (`right to left`).

O valor `auto` define o sentido de escrita com base nos padrões do dispositivo que renderiza os conteúdos.

Nas versões anteriores da HTML esse atributo é definido para todos os elementos exceto os elementos `applet`, `base`, `basefont`, `br`, `frame`, `frameset`, `iframe`, `param` e `script`. Na HTML5 tornou-se um atributo global podendo ser usado para todos os elementos.

O exemplo a seguir é interativo e demonstra o efeito desse atributo quando definido para o elemento `body` de uma página.

► JavaScript

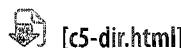
```
window.onload = function() {
    var botao = document.getElementsByTagName('button');
    var pg = document.getElementsByTagName('body');

    botao[0].onclick = function() {
        pg[0].setAttribute('dir', 'ltr');
    }
    botao[1].onclick = function() {
        pg[0].setAttribute('dir', 'rtl');
    }
    botao[2].onclick = function() {
        pg[0].setAttribute('dir', 'auto');
    }
}
```

► HTML

```
<body>
...
<h3>Demonstração</h3>
<p>Essa página integra o conjunto de páginas exemplo do livro HTML5 do Maujor.</p>
<p>Foi criada para demonstrar o atributo global dir</p>
<p>Clique os botões a seguir para definir os atributos
    ltr, rtl e auto para o elemento body dessa página</p>
<button type="button">ltr</button>
<button type="button">rtl</button>
<button type="button">auto</button></section>
...

```



5.1.6 `draggable`

Esse atributo destina-se a definir se o elemento será ou não arrastável na página. Os valores possíveis para esse atributo são: `true`, `false` e `auto`. O atributo apenas define o

elemento arrastável. Para tornar o arraste efetivo há necessidade de uso de métodos e atributos JavaScript previstos na API Drag&Drop.

O valor `true` faz com que seja possível arrastar pela página o elemento para o qual o atributo foi definido.

O valor `false` não permite o arraste do elemento.

O valor `auto` coloca o elemento no estado de arraste-padrão definido pelo agente de usuário.

Esse é um atributo não existente nas versões anteriores da HTML, tendo sido criado pela especificação da HTML5.

5.1.7 dropzone

Esse atributo destina-se a definir o elemento que receberá (área de recebimento) um elemento arrastável na página. Os valores possíveis para esse atributo são: `copy`, `move` e `link`.

O valor `copy` faz com que seja criada uma cópia do elemento arrastável na área de recebimento.

O valor `move` move o elemento arrastável para a área de recebimento.

O valor `link` faz com que seja criado um link para o elemento arrastável na área de recebimento.

Esse é um atributo não existente nas versões anteriores da HTML, tendo sido criado pela especificação da HTML5.

5.1.8 hidden

Esse atributo destina-se a indicar que o elemento não é relevante na página e os agentes de usuário não devem renderizá-lo. Trata-se de um atributo booleano, ou seja, é ou não definido para um elemento.



Não confundir atributo booleano com valor booleano para o atributo. Neste atribuem-se os valores `true` ou `false` e naqueles a presença ou ausência do atributo sem declaração de valor é suficiente.

Esse é um atributo não existente nas versões anteriores da HTML, tendo sido criado pela especificação da HTML5.

O exemplo a seguir esclarece a sintaxe e uso desse atributo.

► HTML

```
<h1>Star Trek Game</h1>
<section id="login">
    <h2>Login</h2>
    <form>
        ...
        <!-- formulário para login -->
    </form>
<script>
function login() {
    // chave mostra/esconde
    document.getElementById('login').hidden = true;
    document.getElementById('game').hidden = false;
}
</script>
</section>
<section id="game" hidden>
    <!-- Interface do jogo -->
</section>
```

Nesse exemplo, para uma interface de jogo, inicialmente é apresentado ao usuário um formulário de login. Preenchidos os dados no formulário, o script valida e estando tudo OK esconde o formulário de login e apresenta a interface do jogo ao usuário.

5.1.9 id

Esse atributo destina-se a criar um identificador único para o elemento na página. O valor desse atributo deve conter pelo menos um caractere e não poderá conter caracteres vazios.

Um valor para esse atributo deve ser único na página, ou seja, a dois ou mais elementos de uma página não é permitido atribuir um mesmo valor de id.

Esse é um atributo que já existia nas versões anteriores da HTML e a diferença é na regra de atribuição do seu valor. Nas versões anteriores da HTML, valores de id devem começar com uma letra maiúscula ou minúscula seguida de qualquer número de letras, números, sublinhado, hífen, dois pontos e pontos. Na HTML5 qualquer caractere é permitido menos espaço em branco.

5.1.10 itemid, itemprop, itemref, itemscope, itemtype

Esses atributos são descritos nas especificações para Microdados e serão estudados com detalhes no capítulo 11 deste livro.

Esses atributos não existem nas versões anteriores da HTML, tendo sido criados pela especificação da HTML5.

5.1.11 lang

Esse atributo destina-se a identificar o idioma primário do texto inserido no elemento para o qual foi definido. O valor desse atributo deve ser uma abreviatura-padrão identificadora de idiomas.

Nas versões anteriores da HTML esse atributo é definido para todos os elementos exceto os elementos `applet`, `base`, `basefont`, `br`, `frame`, `frameset`, `iframe`, `param` e `script`. Na HTML5 tornou-se um atributo global podendo ser usado para todos os elementos.

O exemplo a seguir esclarece a sintaxe e uso desse atributo.

► HTML

```
<p>A palavra <span lang="en">star</span> em inglês significa estrela.</p>
```

5.1.12 spellcheck

Esse atributo destina-se a marcar conteúdos inseridos pelo usuário que devam ter sua grafia e sintaxe verificada por um corretor ortográfico.

Vários editores de texto, por exemplo, o Word do ambiente Windows, possuem uma funcionalidade conhecida como corretor ortográfico que destaca uma palavra ou texto digitado sempre que encontra um possível erro de sintaxe ou grafia, com a finalidade de chamar a atenção do autor para o erro. Esse atributo cria a funcionalidade de correção ortográfica para o conteúdo do elemento ao qual é inserido. O software utilizado para fazer a correção (por exemplo, um dicionário ou gramática) e a maneira como se integra ao agente de usuário não são descritos nas especificações, ficando por conta do fabricante.

Esse atributo admite os valores `true`, `false` e `default` que definem se o conteúdo do elemento deve ser submetido ao corretor ortográfico ou não ou se sua condição de correção é aquela definida pelo navegador.

Esse é um atributo não existente nas versões anteriores da HTML, tendo sido criado pela especificação da HTML5.

O exemplo a seguir esclarece a sintaxe e uso desse atributo.

► HTML

```
<div contenteditable="true">
    <p spellcheck="true">Conteúdo editável a ser verificado pelo corretor ortográfico</p>
</div>
```

5.1.13 style

Esse atributo destina-se a definir regras CSS para o elemento ao qual foi aplicado.

Nas versões anteriores da HTML esse atributo é definido para todos os elementos exceto os elementos `base`, `basefont`, `head`, `html`, `meta`, `param`, `script`, `style` e `title`. Na HTML5 tornou-se um atributo global podendo ser usado para todos os elementos.

Quando se usa esse atributo, segundo a especificação para HTML5, alterar a apresentação de maneira que a retirada do atributo altere o significado geral do conteúdo é uma prática não conforme. Por exemplo: é desaconselhável usar o atributo `style` para esconder e revelar conteúdos. Para isso use o atributo `hidden`.

O exemplo a seguir esclarece a sintaxe e uso desse atributo.

► HTML

```
<p style="color:red; background:black;">Conteúdo estilizado com o atributo style</p>
```

5.1.14 tabindex

Esse atributo destina-se a definir se um elemento pode receber o foco. O valor desse atributo é um número inteiro positivo ou negativo e cria uma ordem de tabulação para o elemento ao qual foi aplicado. Ordem de tabulação é a sequência na qual o foco é dado quando o usuário navega com auxílio do teclado acionando continuamente a tecla Tab.

Nas versões anteriores da HTML esse atributo é definido somente para os elementos `a`, `area`, `button`, `input`, `object`, `select`, e `textarea`. Na HTML5 tornou-se um atributo global podendo ser usado para todos os elementos. Por padrão os seguintes elementos podem receber o foco independentemente de o atributo ter sido para eles definido: `a`, `link`, `button`, `input`, `select`, `textarea` e `command`.

Se o valor do atributo for um número negativo, o elemento poderá receber o foco, mas ficará fora da ordem de tabulação.

Se o valor do atributo for zero, o elemento poderá receber o foco e sua posição na ordem de tabulação é determinada por padrão pela plataforma.

Se o valor do atributo for um número positivo, o elemento poderá receber o foco e a ordem de tabulação se dará dos números mais baixos para os mais altos. Convém notar que a numeração não precisa ser uma sequência corrida do tipo 1,2,3,4..., pode ser 23,67,82,91... ainda que pareça estranho não se adotar uma sequência corrida.

5.1.15 title

Esse atributo destina-se a fornecer uma informação adicional sobre o conteúdo do elemento para o qual é definido. O valor desse atributo é um texto contendo a informação.

Nas versões anteriores da HTML esse atributo é definido para todos os elementos exceto para os elementos `base`, `basefont`, `head`, `html`, `meta`, `param`, `script`, e `title`. Na HTML5 tornou-se um atributo global podendo ser usado para todos os elementos.

Se esse atributo for omitido para um elemento, fica implícito que para ele é relevante o atributo `title` de seu ancestral mais próximo, caso tenha sido definido para aquele ancestral.

Os exemplos a seguir esclarecem a sintaxe e uso desse atributo.

► HTML

```
<a href="http://maujor.com title="Site sobre CSS" >Site do Maujor</a>

<p title="Regiões geográficas do Brasil">Texto descritivo da regiões geográficas do Brasil</p>
```

5.2 Novos atributos

A HTML5 criou vários atributos não existentes na HTML4. Alguns desses atributos são globais e foram mostrados no item anterior. Outros são atributos específicos de um ou mais elementos e serão mostrados a seguir.

5.2.1 `async`

Esse atributo destina-se a indicar como um script deve ser executado. Quando presente, faz com que o script seja executado de modo assíncrono tão logo ele tenha sido carregado e esteja disponível.

O único elemento que admite o uso desse atributo é o elemento `script`.

Trata-se de um atributo booleano, ou seja, é ou não definido para um elemento. Esse atributo não deve ser definido para o elemento `script` que não contenha o atributo `src`, ou seja, ele só tem efeito para scripts externos chamados na página pelo atributo `src`.



Não confundir atributo booleano com valor booleano para o atributo. Neste atribuem-se os valores `true` ou `false` e naqueles a presença ou ausência do atributo sem declaração de valor é suficiente.

5.2.2 `autocomplete`

Alguns agentes de usuário oferecem uma funcionalidade que consiste em preencher ou sugerir opções de preenchimento de um campo com base no que nele foi digitado anteriormente pelo usuário.

Esse atributo destina-se a controlar esse comportamento habilitando ou não o ato de autopreenchimento.

Os elementos que admitem o uso desse atributo são: `form` e `input`.

Esse atributo admite os valores `on` e `off` que definem se o campo deve ser autopreenchido ou não.

5.2.3 `autofocus`

Esse atributo destina-se a definir em qual elemento deve ser dado o foco tão logo a página seja carregada.

Os elementos que admitem o uso desse atributo são: `button`, `input`, `keygen`, `select` e `textarea`.

Trata-se de um atributo booleano, ou seja, é ou não definido para um elemento e apenas um elemento da página poderá conter esse atributo.



Não confundir atributo booleano com valor booleano para o atributo. Neste atribuem-se os valores `true` ou `false` e naqueles a presença ou ausência do atributo sem declaração de valor é suficiente.

O exemplo a seguir esclarece a sintaxe e uso desse atributo. Nele o foco é dado ao campo para a coleta do nome. Atualmente somente os navegadores Opera e Chrome suportam esse atributo.

► HTML

```
<form method="get" action="">  
  <p><label>Nome:<br><input type="text" autofocus></label></p>  
  <p><label>Email:<br><input type="text"></label></p>  
  <p><label>Cidade:<br><input type="text"></label></p>  
  ...  
</form>
```



[c5-autofocus.html]

5.2.4 autoplay e controls

Esses são atributos para os elementos `video` e `audio` e foram estudados com detalhes em 3.3.1.

5.2.5 challenge

Esse atributo destina-se a definir um texto a ser anexado ao pacote a ser enviado ao servidor contendo a chave gerada pelo elemento `keygen`. Atualmente a forma de implementação do elemento `keygen` não está definida.

O único elemento que admite o uso desse atributo é o elemento `keygen`.

5.2.6 default

Esse atributo destina-se a habilitar um texto `default` para o elemento `track` sempre que não for definido um texto personalizado para o elemento.

O único elemento que admite o uso desse atributo é o elemento `track`.

5.2.7 dirname

Esse atributo destina-se a definir um dado adicional a ser anexado aos dados enviados por um formulário. O dado adicional informa a direção de escrita (ltr ou rtl) do texto que está sendo enviado.

Os elementos que admitem o uso desse atributo são: `input` e `textarea`.

Esse atributo será estudado com detalhes em 6.2.5.

5.2.8 form

Esse atributo destina-se a associar um controle de formulário a um elemento formulário. Nas versões anteriores da HTML um controle de formulário está associado (ou pertence) ao formulário no qual está inserido. Esse atributo permite que se associem a um formulário controles inseridos fora dele e até mesmo controles dentro de outro formulário.

O valor desse atributo é o mesmo valor do atributo `id` do formulário ao qual está associado.

Os elementos que admitem o uso desse atributo são: `button`, `fieldset`, `input`, `keygen`, `label`, `meter`, `object`, `output`, `progress`, `select` e `textarea`.

Esse atributo será estudado com detalhes em 6.2.6.

5.2.9 formaction

Esse atributo tem o mesmo efeito do atributo `action` para o elemento `form`, ou seja, destina-se a definir um URL para o qual o formulário será enviado para processamento.

Os elementos que admitem o uso desse atributo são: `button` e `input`. Assim, a definição do URL para processamento é definida pelo valor desse atributo inserido em um desses dois elementos.

Esse atributo será estudado com detalhes em 6.2.7.

5.2.10 formenctype

Esse atributo tem o mesmo efeito do atributo `enctype` para o elemento `form`, ou seja, destina-se a definir o tipo de conteúdo que está sendo enviado pelo formulário.

Os elementos que admitem o uso desse atributo são: `button` e `input`. Assim, a definição da codificação é definida pelo valor desse atributo inserido em um desses dois elementos.

Esse atributo será estudado com detalhes em 6.2.8.

5.2.11 `formmethod`

Esse atributo tem o mesmo efeito do atributo `method` para o elemento `form`, ou seja. destina-se a definir o método usado para enviar os dados do formulário.

Os elementos que admitem o uso desse atributo são: `button` e `input`. Assim, a definição do método é definida pelo valor desse atributo inserido em um desses dois elementos.

Esse atributo será estudado com detalhes em 6.2.9.

5.2.12 `formnovalidate`

Esse atributo tem o mesmo efeito do atributo `novalidate` para o elemento `form`, ou seja. destina-se a desabilitar a validação dos dados do formulário. (O atributo `novalidate` é novo na HTML5. Ver 5.2.23).

Os elementos que admitem o uso desse atributo são: `button` e `input`. Assim, a definição da necessidade ou não de se validar os dados é definida pela inserção desse atributo em um desses dois elementos.

Esse atributo será estudado com detalhes em 6.2.10.

5.2.13 `formtarget`

Esse atributo define o destino do documento que será aberto após o envio do formulário.

Os elementos que admitem o uso desse atributo são: `button` e `input`. Assim, o destino do documento é definido pela inserção desse atributo em um desses dois elementos.

Os valores possíveis para esse atributo são: `_blank`, `_top`, `_self` e `_parent` ou qualquer nome no contexto da janela atual.

Esse atributo será estudado com detalhes em 6.2.11.

5.2.14 icon

Esse atributo é para uso exclusivo com o elemento `command` e destina-se a definir um URL apontando para a imagem de um ícone representativo do comando.

O exemplo a seguir, retirado e adaptado das especificações para a HTML5, esclarece a sintaxe e uso desse atributo.

► HTML

```
<menu type="toolbar">
    <command type="radio" radiogroup="alinhamento" checked="checked"
        label="Alinhar à esquerda" icon="icons/all.png" onclick="alinhar('esquerda')">
    <command type="radio" radiogroup=" alinhamento "
        label="Centralizar" icon="icons/alc.png" onclick="alinhar('centro')">
    <command type="radio" radiogroup=" alinhamento "
        label=" Alinar à direita" icon="icons/alR.png" onclick="alinhar('direita')">
        <hr>
    <command type="command" disabled
        label="Publicar" icon="icons/pub.png" onclick="publish()">
</menu>
```

5.2.15 keytype

Esse atributo é para uso exclusivo com o elemento `keygen` e destina-se a definir o tipo de algoritmo para codificar a chave gerada pelo elemento.

5.2.16 kind

Esse atributo é para uso exclusivo com o elemento `track` e destina-se a definir o tipo de trilha a ser acrescentado na mídia pelo elemento. Os valores possíveis para esse atributo são: `subtitles`, `captions`, `descriptions`, `chapters`, `metadata` para trilhas de subtítulos, legendas, descrições, capítulos e metadados, respectivamente.

O exemplo a seguir, retirado e adaptado das especificações para a HTML5, esclarece a sintaxe e uso desse atributo.

► HTML

```
<video src="brave.webm">
    <track kind=subtitles src=brave.en.vtt srclang=en label="Inglês">
    <track kind=captions src=brave.en.vtt srclang=en label="Inglês para deficientes auditivos">
    <track kind=subtitles src=brave.fr.vtt srclang=fr label="Francês">
    <track kind=subtitles src=brave.de.vtt srclang=de label="Alemão">
</video>
```

5.2.17 list

Esse atributo é para uso exclusivo com o elemento `input` e destina-se a definir uma lista de opções para preenchimento de um campo de texto. O seu valor deve ser igual ao valor do `id` de um elemento `datalist`.

O exemplo a seguir esclarece a sintaxe e uso desse atributo.

► HTML

```
<label>Informe sua cidade preferida:<br />
    <input list="cidades">
</label><br />
<datalist id="cidades">
    ou <label>escolha uma<br />
        <select>
            <option value="Bruxelas">Bruxelas</option>
            <option value="Lisboa">Lisboa</option>
            <option value="Londres">Londres</option>
            <option value="Moscou">Moscou</option>
            <option value="Oslo">Tokyo</option>
            <option value="Paris">Paris</option>
            <option value="Roma">Roma</option>
            <option value="Tokyo">Tokyo</option>
        </select>
    </label>
</datalist>
```



[c5-list.html]

Dos navegadores atuais somente o Opera e o Firefox 4 suportam esse atributo. Nesses navegadores, quando o usuário entra no campo de texto, a lista de cidades é apresentada para escolha e, caso nenhuma delas seja a preferida, é possível digitar outra opção e o campo `select` não é renderizado. Em navegadores que não suportam o atributo, o campo `select` é renderizado normalmente.

5.2.18 loop

Esse atributo é para uso exclusivo com os elementos `audio` e `video` e foi estudado com detalhes no capítulo 3.

5.2.19 low

Esse atributo é para uso exclusivo com o elemento `meter` e destina-se a definir um valor dentro da faixa à qual a medida pertence e abaixo do que se considera um valor baixo. Ver 2.4.17.

5.2.20 manifest

Esse atributo é para uso exclusivo com o elemento `html` que será estudado com detalhes no capítulo 10.

5.2.21 max

Esse atributo é para uso exclusivo com os elementos `meter`, `progress` e `input` e destina-se a definir um valor máximo. O uso desse atributo para os elementos `meter` e `progress` foi estudado em 2.4.17 e 2.4.20 respectivamente.

Para o elemento `meter` ele destina-se a definir o valor máximo da escala à qual a medida marcada pertence e, se for omitido, será considerado igual a 1; para o elemento `progress` ele destina-se a definir o quanto é necessário para realizar uma tarefa.

Usar esse atributo com o elemento `input` define um valor numérico máximo permitido como entrada no campo. Usado em conjunto com o atributo `min` estabelece uma faixa de entrada de dados no campo.

O exemplo a seguir esclarece a sintaxe e uso desse atributo.

► HTML

```
<input name="quantidade" required type="number" min="10" max="20" value="15">
```

 [c5-max-min.html]

Nesse exemplo, quando a página é carregada, o campo está preenchido com o valor 15; ao usuário é permitido entrar valores compreendidos entre 10 e 20 inclusive. Esse exemplo está disponível no site do livro para visualização no navegador Opera ou posteriores.

5.2.22 min

Esse atributo é para uso exclusivo com os elementos `meter` e `input` e destina-se a definir um valor mínimo. O uso desse atributo para o elemento `meter` define o valor mínimo da escala à qual a medida marcada pertence, se for omitido, será considerado igual a 0. Ver 2.4.17.

Usar esse atributo com o elemento `input` define um valor numérico mínimo permitido como entrada no campo. Usado em conjunto com o atributo `max` estabelece uma faixa de entrada de dados no campo. Ver 5.2.21.

5.2.23 novalidate

Esse atributo é para uso exclusivo com o elemento `form` e tem o mesmo efeito do elemento `formnovalidate` que foi estudado em 5.2.12.

5.2.24 open

Esse atributo é para uso exclusivo com o elemento `details` e destina-se a definir se o conteúdo do elemento deve ou não ser apresentado ao usuário. Trata-se de um atributo booleano, ou seja, é ou não definido para um elemento. A ausência desse atributo no elemento indica que seu conteúdo não deve ser apresentado ao usuário.



Não confundir atributo booleano com valor booleano para o atributo. Neste atribuem-se os valores `true` ou `false` e naqueles a presença ou ausência do atributo sem declaração de valor é suficiente.

5.2.25 optimum

Esse atributo é para uso exclusivo com o elemento `meter` e destina-se a definir um valor considerado ótimo dentro da escala à qual a medida marcada pertence. Ver 2.4.17.

5.2.26 pattern

Esse atributo é para uso exclusivo com o elemento `input` e destina-se a definir uma expressão regular (um padrão) com a qual o valor entrado em um campo de formulário deve ser comparado.

O exemplo a seguir esclarece a sintaxe e uso desse atributo.

► HTML

```
<input type="text" pattern="[0-9]{5}-[0-9]{3}" name="cep"  
      title="Um CEP é constituído de 5 números um traço e mais 3 números." />
```

Nesse exemplo, caso o usuário entre com um valor que não seja um número de CEP válido, o valor é recusado e é apresentada uma mensagem informando ao usuário que o valor digitado não está de acordo com o formato exigido pelo controle.

5.2.27 ping

Esse atributo é para uso exclusivo com os elementos `a` e `area` e destina-se a definir URLs relacionadas interessados em serem notificadas caso o usuário siga o link. O valor desse atributo é um URL ou uma lista de URLs separados por espaço.

5.2.28 placeholder

Esse atributo é para uso exclusivo com os elementos `input` e `textare`a e destina-se a definir uma dica (palavra ou frase curta) de preenchimento do campo.

O exemplo a seguir esclarece a sintaxe e uso desse atributo.

► HTML

```
<legend>Conta de e-mail</legend>  
<p><label>Nome:<br /><input type="text" name="name" placeholder="Maurício Samy Silva">  
    </label></p>  
<p><label>E-mail:<br /><input type="email" name="email" placeholder="maujor@maujor.com">  
    </label></p>  
<p><label>Senha:<br /><input type="password" name="password"></label></p>  
</fieldset>
```



[c5-placeholder.html]

Nesse exemplo, quando a página é carregada, os campos nome e e-mail estão preenchidos com os valores definidos pelo atributo `placeholder`. Quando o usuário dá o foco ao campo, os valores desaparecem para dar lugar ao valor a ser entrado. Esse exemplo está disponível no site do livro para visualização no navegador Opera 11 ou posteriores.

5.2.29 poster

Esse atributo é para uso exclusivo com o elemento `video` e foi estudado com detalhes em 3.3.1.

5.2.30 preload

Esse atributo é para uso exclusivo com os elementos `video` e `audio` e foi estudado com detalhes em 3.3.1.

5.2.31 pubdate

Esse atributo é para uso exclusivo com o elemento `time` e destina-se a definir a data de publicação. Trata-se de um atributo booleano, ou seja, é ou não definido para um elemento.



Não confundir atributo booleano com valor booleano para o atributo. Neste atribuem-se os valores `true` ou `false` e naqueles a presença ou ausência do atributo sem declaração de valor é suficiente.

O exemplo a seguir esclarece a sintaxe e uso desse atributo.

► HTML

```
<article>
  <h1>Pequenas tarefas</h1>
  <footer>Publicado em <time pubdate>05/02/2011</time>.</footer>
  <p>Trocar a lâmpada da sala da sala.</p>
</article>
```

Esse atributo define a data de publicação do primeiro elemento `article` ancestral do elemento `time` no qual o atributo foi inserido. Não havendo um elemento `article` ancestral, a data refere-se à publicação do documento como um todo.

5.2.32 radiogroup

Esse atributo é para uso exclusivo com o elemento `command` e destina-se a agrupar comandos do tipo `radio`. Só deve ser usado se o comando é do tipo `radio`.

O exemplo a seguir esclarece a sintaxe e uso desse atributo.

► HTML

```
<menu type="toolbar">
    <command type="radio" radiogroup="alinhamento" checked="checked"
        label="Esquerda" icon="icons/all.png" onclick="setAlign('esq')">
    <command type="radio" radiogroup="alinhamento"
        label="Centro" icon="icons/alc.png" onclick="setAlign('centro')">
    <command type="radio" radiogroup="alinhamento"
        label="Direita" icon="icons/alR.png" onclick="setAlign('dir')">
</menu>
```

5.2.33 required

Esse atributo é para uso exclusivo com os elementos `input`, `select` e `textareá` e destina-se a marcar um controle de formulário como sendo de preenchimento obrigatório. Veremos, com detalhes, o uso desse atributo em formulários no capítulo 6.

Trata-se de um atributo booleano, ou seja, é ou não definido para um elemento.



Não confundir atributo booleano com valor booleano para o atributo. Neste atribuem-se os valores `true` ou `false` e naqueles a presença ou ausência do atributo sem declaração de valor é suficiente.

O exemplo a seguir esclarece a sintaxe e uso desse atributo.

► HTML

```
...
<p>
    <label for="email">E-mail:</label>
    <input id="email" type=email required name=email>
</p>
```

5.2.34 reversed

Esse atributo é para uso exclusivo com o elemento `ol` e destina-se a definir uma contagem descendente para o marcador da lista ordenada, isto é, a numeração começa do número mais alto para o mais baixo (...4,3,2,1) em lugar da numeração-padrão crescente (1,2,3,4 ...).

Trata-se de um atributo booleano, ou seja, é ou não definido para um elemento.



Não confundir atributo booleano com valor booleano para o atributo. Neste atribuem-se os valores `true` ou `false` e naqueles a presença ou ausência do atributo sem declaração de valor é suficiente.

O exemplo a seguir esclarece a sintaxe e uso desse atributo.

► HTML

```
<ol reversed>
  <li>Maçã</li>
  <li>Pera</li>
  <li>Uva</li>
  <li>Laranja</li>
</ol>
```

A marcação mostrada deverá causar a renderização conforme segue.

4. Maçã
3. Pera
2. Uva
1. Laranja

5.2.35 `sandbox`

Esse atributo é para uso exclusivo com o elemento `iframe` e destina-se a definir permissões para inserção de conteúdos no elemento.

Quando definido, sem declaração explícita de um valor, permite inserções somente dos conteúdos provenientes de uma mesma origem. Formulários, scripts e plugins são desabilitados e links para recursos fora do contexto do navegador também desabilitados.

Para habilitar as funcionalidades bloqueadas, os valores possíveis para esse atributo são: `allow-same-origin`, `allow-top-navigation`, `allow-forms`, and `allow-scripts` que habilitam respectivamente diferentes origens, links, formulários e scripts.

O exemplo a seguir esclarece a sintaxe e uso desse atributo.

► HTML

```
<iframe sandbox=allow-same-origin allow-forms allow-scripts
src="http://maujor.com/xpto.html"></iframe>
```

5.2.36 `scoped`

Esse atributo é para uso exclusivo com o elemento `style` e destina-se a definir o escopo para aplicação de estilos. Quando definido, faz com que as regras de estilo contidas no elemento `style` sejam aplicadas somente no elemento-pai do elemento `style` e nos seus elementos-filho.

Trata-se de um atributo booleano, ou seja, é ou não definido para um elemento.



Não confundir atributo booleano com valor booleano para o atributo. Neste atribuem-se os valores `true` ou `false` e naqueles a presença ou ausência do atributo sem declaração de valor é suficiente.

O exemplo a seguir esclarece a sintaxe e uso desse atributo.

► HTML

```
...
<p>Parágrafo na cor definida na folha de estilos do documento</p>
<a href="http://maujor.com">Site do Maujor</a>
<div>
  <style scoped>
    p {color:red;}
  </style>
  <p>Parágrafo na cor vermelha</p>
</div>
... mais marcação ...
```

Nesse exemplo todo texto inserido no `div` será na cor vermelha e os inseridos no restante do documento serão na cor definida pelas CSS para o documento. Notar que em sintaxe HTML5 é válido inserir o elemento `style` dentro do elemento `body`.

5.2.37 `seamless`

Esse atributo é para uso exclusivo com o elemento `iframe` e destina-se a fazer com que o `iframe` seja inserido no mesmo contexto do documento que o contém.

Isto significa que links existentes no `iframe` abrem no mesmo contexto de abertura dos links do documento (a não ser que o atributo `target="_self"` esteja presente), regras de estilo para o documento sejam aplicadas ao conteúdo do `iframe` e que todos os comportamentos do documento se apliquem ao conteúdo do `iframe`.

Trata-se de um atributo booleano, ou seja, é ou não definido para um elemento.



Não confundir atributo booleano com valor booleano para o atributo. Neste atribuem-se os valores `true` ou `false` e naqueles a presença ou ausência do atributo sem declaração de valor é suficiente.

O exemplo a seguir esclarece a sintaxe e uso desse atributo.

► HTML

```
<iframe seamless src="http://maujor.com/xpto.html"></iframe>
```

5.2.38 sizes

Esse atributo é para uso exclusivo com o elemento `link` e destina-se a definir as dimensões do ícone eventualmente definido para o `link`.

Os valores possíveis para esse atributo são: a palavra-chave `any` ou o par largura x altura do ícone em pixels.

O exemplo a seguir, retirado da especificação para HTML do WHATWG, esclarece a sintaxe e uso desse atributo.

► HTML

```
<link rel=icon href=favicon.png sizes="16x16" type="image/png">
<link rel=icon href=windows.ico sizes="32x32 48x48" type="image/vnd.microsoft.icon">
<link rel=icon href=mac.icns sizes="128x128 512x512 8192x8192 32768x32768">
<link rel=icon href=iphone.png sizes="57x57" type="image/png">
<link rel=icon href=gnome.svg sizes="any" type="image/svg+xml">
```

5.2.39 srcdoc

Esse atributo é para uso exclusivo com o elemento `iframe` e destina-se a definir a marcação HTML a ser inserida no `iframe`.

O valor para esse atributo é uma marcação HTML.

O exemplo a seguir, retirado da especificação para HTML do WHATWG, esclarece a sintaxe e uso desse atributo.

► HTML

```
<article>
  <h1>Título do post</h1>
  <p>Conteúdo do post</p>
  <footer>
```

```
<p>Escrito por: <a href="/autores/maujor">Maujor</a>. <time pubdate>2009-08-21T23:32Z</time></p>
</footer>
<article>
<footer>Em <time pubdate>2009-08-21T23:35Z</time>, <a href="/usuarios/m">Marcos</a> comentou:
</footer>
<iframe seamless sandbox srcdoc="<p>Gostei do post, mas ...</p>"></iframe>
</article>
<article>
<footer>Em <time pubdate>2009-08-21T23:44Z</time>, <a href="/usuarios/p">Paulo</a> comentou:
</footer>
<iframe seamless sandbox srcdoc="<p>Veja exemplo
<a href="/gallery?mode=cover&amp;page=1">na minha galeria</a>.">
</iframe>
</article>
</article>
```

Esse exemplo resume uma marcação para um post publicado em blog com dois comentários. Na marcação os comentários foram inseridos no documento com uso do elemento `iframe` no qual o atributo `src` foi usado em conjunto com os atributos `seamless` e `sandbox` (estudados anteriormente), fornecendo um layer extra de proteção contra injeção de scripts maliciosos pelo formulário de comentário.

5.2.40 `srclang`

Trata-se de um atributo para o elemento `track`, estudado com detalhes em 3.2.1.

5.2.41 `step`

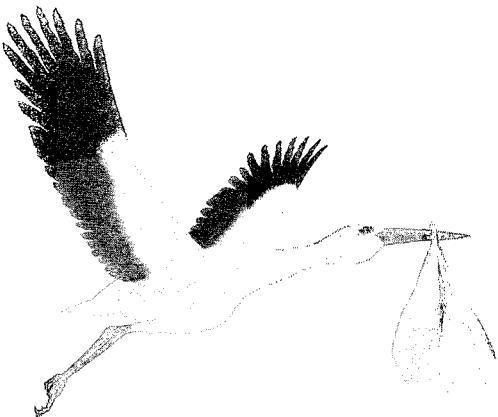
Esse atributo é para uso exclusivo com o elemento `input` e será estudado com detalhes no capítulo 6.

5.2.42 `wrap`

Esse atributo é para uso exclusivo com o elemento `textarea` e destina-se a definir como será o fluxo de texto quando submetido para processamento. Será enviado em uma linha ou conterá quebras de linhas.

Os valores possíveis para esse atributo são: `soft` e `hard` sendo `soft` o valor-padrão, usado quando o atributo não é especificado.

Quando for usado o valor `hard` devemos especificar o atributo `cols` para a `textarea`. Nesse caso, quebras de linhas inseridas pelo usuário serão consideradas no envio do texto ao servidor.



CAPÍTULO 6

Formulários

Neste capítulo, mostraremos as funcionalidades da HTML5 para formulários. A grande novidade nessa área é a implementação de atributos capazes de disparar mecanismos de validação nativos do navegador. Estudaremos com detalhes tais atributos e forneceremos exemplos práticos que ilustram as funcionalidades de validação. Além da validação, novos atributos tais como `placeholder` e `autofocus` e a possibilidade de se inserir campos de formulário em qualquer lugar da página fora do elemento `form` alteram e agilizam a maneira como se desenvolvem formulários na HTML5.

6.1 Introdução

Um formulário HTML é uma seção do documento contendo texto normal, marcação, elementos especiais chamados de controles (`inputs`, `checkboxes`, `radio-buttons`, `textarea` etc.) e seus respectivos rótulos (`label`).

Geralmente os usuários “preenchem” um formulário modificando seus controles (digitando textos, selecionando itens de menu etc.), antes de enviar o formulário a um agente para processamento.

Formulário é talvez a funcionalidade de integração com o usuário mais usada na web e também uma das mais antigas. Sua função é recolher dados entrados pelo usuário e enviá-los ao servidor para processamento.

O processamento dos dados no servidor faz-se por scripts desenvolvidos em linguagens de programação no lado servidor e consiste desde o disparo de um simples e-mail contendo os dados captados no formulário até sofisticados mecanismos de consultas e integração com banco de dados em transações envolvendo dados sigilosos e encriptados como pagamentos on-line e movimentações financeiras.

Um usuário ao preencher um formulário poderá digitar em seus controles não apenas textos simples, mas também marcação HTML e até mesmo scripts inteiros. Assim, um campo de formulário é uma porta de entrada de scripts maliciosos capazes de alterar a programação de processamento do formulário no servidor e desencadear ações danosas ao proprietário do site e seus usuários.

Por essa razão, ao se desenvolver um formulário e seu script de processamento, devemos considerar com alta prioridade a criação de mecanismos que impeçam a entrada de scripts maliciosos. A maneira consagrada para atingir esse objetivo é o que conhecemos como validação do formulário.

Validar um formulário é a ação de verificação dos dados digitados pelo usuário nos controles do formulário. Por exemplo: se em um controle destinado à coleta do CPF esperamos que o usuário digite onze dígitos numéricos no formato xxxxxxxx-xx devemos criar um script capaz de ler o que foi digitado no campo, verificar se existem somente onze números de 0 a 9 e um traço (-) e conferir se os números satisfazem ao algoritmo de formação de um número de CPF. Assim, garantimos primeiro que o usuário entrou um CPF real e segundo que não houve digitação de códigos ou outros dados inválidos.

Até a chegada da HTML5 a validação dos dados de um formulário era feita com uso de JavaScript e como medida de segurança replicada com script rodando no lado do servidor para o caso de JavaScript ter sido desabilitada no navegador.

Uma quantidade impressionante de sites mais antigos e outro tanto não tão antigos validam seus formulários com uso somente de JavaScript. Isso é uma temeridade, pois desabilitar JavaScript e anular os efeitos da validação é uma tarefa trivial, realizada com dois cliques somente, em qualquer navegador.

A HTML5 introduziu um conceito de validação de formulários totalmente inovador, simples e eficiente, reduzindo a necessidade de desenvolvimento de scripts duplicados para a tarefa. A criação de novos atributos para os controles de formulários, destinados a definir o tipo de dado a ser entrado, disparam mecanismos nativos de validação e mensagens de erro. Por exemplo: o atributo `email` quando presente em um elemento de controle do formulário não permite que ele seja enviado ao servidor para processamento se no controle constar uma string que não se pareça com um endereço de e-mail.

6.2 Atributos

As funcionalidades da HTML5 para formulários foram introduzidas na linguagem com uso de atributos a serem usados nos controles do formulário. Os novos atributos

não são destinados somente à validação, mas proporcionam facilidades outras aos formulários que, antes da chegada da HTML, eram conseguidas somente com uso de scripts. Nesta seção estudaremos com detalhes esses atributos.

6.2.1 placeholder

Esse atributo é para uso exclusivo com os elementos `input` e `textarea` e destina-se a definir uma dica (palavra ou frase curta) de preenchimento do campo. Esse atributo foi estudado com detalhes e exemplo online em 5.2.28.

O exemplo a seguir esclarece a sintaxe e uso desse atributo.

► HTML

```
<p><label>E-mail:<br /><input type="email" name="email" placeholder="maujor@maujor.com"></label></p>
```

6.2.2 autofocus

Esse atributo destina-se a definir em qual elemento deve ser dado o foco tão logo a página seja carregada.

Os elementos que admitem o uso desse atributo são: `button`, `input`, `keygen`, `select` e `textarea`. Esse atributo foi estudado com detalhes e exemplo online em 5.2.3.

O exemplo a seguir esclarece a sintaxe e uso desse atributo.

► HTML

```
<p><label>Nome:<br /><input type="text" autofocus></label></p>
```

6.2.3 required

Esse atributo é para uso exclusivo com os elementos `input`, `select` e `textarea` e destina-se a marcar um controle de formulário como sendo de preenchimento obrigatório.

Trata-se de um atributo booleano, ou seja, é ou não definido para um elemento.



Não confundir atributo booleano com valor booleano para o atributo. Neste atribuem-se os valores `true` ou `false` e naqueles a presença ou ausência do atributo sem declaração de valor é suficiente.

O exemplo a seguir esclarece a sintaxe e uso desse atributo.

► HTML

```
<form method="get" action="">
    <p><label for="nome">Nome: </label><input id="nome" type="text" required name="nome">
        <input type="submit" value="OK"></p>
</form>
```



[c6-required.html]

Nesse exemplo, disponível no site do livro e funcionando nos navegadores Opera e Firefox 4, uma mensagem de alerta será apresentada ao usuário se houver uma tentativa de envio do formulário com o controle destinado à coleta do nome em branco. Convém esclarecer que o uso desse atributo verifica simplesmente se o controle foi ou não deixado vazio. Qualquer caractere digitado é aceito.

Na figura 6.1 mostramos a mensagem de alerta nos navegadores citados. Notar que o formato e a estilização da mensagem ficam a cargo do fabricante do navegador.

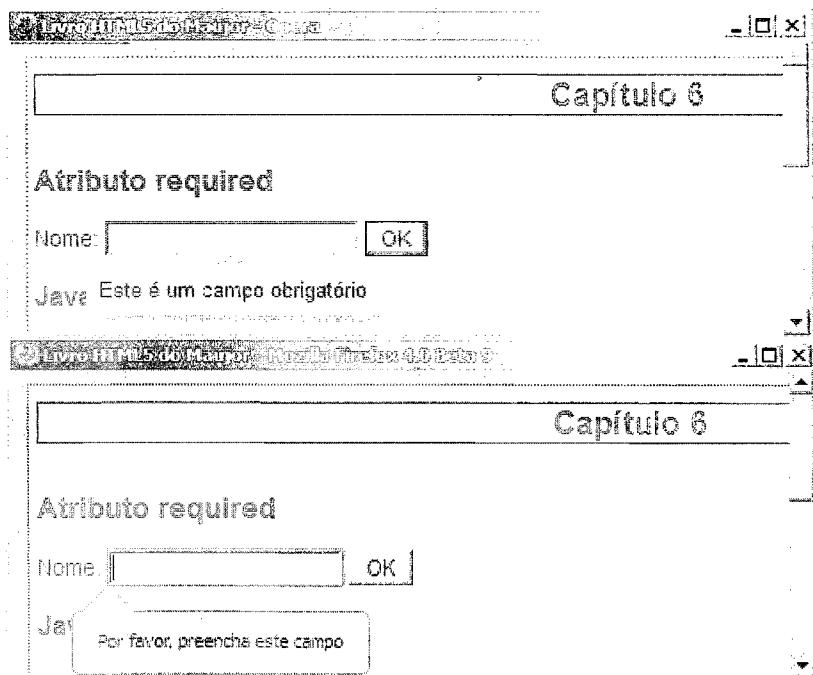


Figura 6.1 – Atributo required.

6.2.4 autocomplete

Alguns navegadores atuais implementam de forma variada uma funcionalidade que consiste em sugerir uma lista de opções para preenchimento de um controle

de formulário, quando a ele se dá o foco, com base no que nele foi digitado anteriormente pelo usuário. Esse atributo destina-se a padronizar o comportamento de autoprovisionamento.

Os elementos que admitem o uso desse atributo são: `form` e `input` e os valores possíveis são: `on` e `off` que definem, respectivamente, se o navegador deve ou não apresentar uma lista de sugestões para autoprovisionamento.

Quando não for especificado esse atributo para determinado controle do formulário, fica subentendido que ele assume o comportamento de autoprovisionamento especificado para o elemento `form` ao qual pertence o controle.

Não sendo especificado o atributo para o elemento `form`, sua condição-padrão de autoprovisionamento é `on`.

O exemplo a seguir esclarece a sintaxe e uso desse atributo.

► HTML

```
<form method="get" action="">
<p><label for="nome">Nome: </label><input id="nome" type="text" required autocomplete=off name=nome>
<input type=submit value="OK"></p>
</form>
```



[c6-autocomplete.html]

Esse exemplo é idêntico ao anterior exceto pelo fato de termos incluído o atributo `autocomplete` no controle para entrada do nome do usuário. O navegador Firefox já oferece essa funcionalidade nativamente há algum tempo. Assim você pode constatar o funcionamento do atributo `autocomplete` naquele navegador comparando o comportamento do controle para a entrada do nome nos dois exemplos anteriores. No exemplo do item anterior, é apresentada uma lista de sugestões de nomes para completar o controle; no exemplo desse item, como declaramos `autocomplete=off` a lista não será apresentada.

6.2.5 `dirname`

Esse atributo destina-se a definir um dado adicional a ser anexado aos dados enviados por um formulário. O dado adicional informa a direção de escrita (ltr ou rtl) do texto que está sendo enviado.

Os elementos que admitem o uso desse atributo são: `input` e `textarea`.

O exemplo a seguir esclarece a sintaxe e uso desse atributo.

► HTML

```
<form method="get" action="">
  <p><label>Nome: <input type="text" name="nome"></label></p>
  <p><label>Email: <input type="text" name="email"></label></p>
  <p><label>Mensagem: <textarea name="msg" dirname="msg.dir"></textarea></label></p>
  <p><input type="submit" name="enviar" value="enviar"></p>
</form>
```

Supondo que o usuário digitou o nome “Maujor”, o e-mail “maujor@maujor.com” e a mensagem “Bom dia”, a string de dados enviada para processamento quando o formulário for enviado é:

```
name=Maujor&email=maujor%40maujor.com&msg=Bom+dia&msg.dir=ltr&enviar=enviar
```

6.2.6 form

Esse atributo destina-se a associar um controle de formulário a um elemento formulário. Nas versões anteriores da HTML, um controle de formulário está associado (ou pertence) ao formulário no qual está inserido. Esse atributo permite que se associem a um formulário controles inseridos fora dele e até mesmo controles dentro de outro formulário.

O valor desse atributo é o mesmo valor do atributo `id` do formulário ao qual está associado.

Os elementos que admitem o uso desse atributo são: `button`, `fieldset`, `input`, `keygen`, `label`, `meter`, `object`, `output`, `progress`, `select` e `textarea`.

O exemplo a seguir esclarece a sintaxe e uso desse atributo.

► HTML

```
<form method="get" action="" id="xpto">
  <p><label>Email: <input type="text" name="email"></label></p>
  <p><label>Mensagem: <textarea name="msg"></textarea></label></p>
  <p><input type="submit" name="enviar" value="enviar"></p>
</form>
<p><label>Nome: <input type="text" name="nome" form="xpto"></label></p>
```

Nesse exemplo o campo para preenchimento do nome está fora do formulário, mas seu conteúdo será enviado normalmente ao servidor para processamento quando o formulário for enviado. O valor desse exemplo é questionável; contudo, a finalidade é apenas mostrar o mecanismo de funcionamento do atributo `form`.

6.2.7 formaction

Esse atributo tem o mesmo efeito do atributo `action` para o elemento `form`, ou seja, destina-se a definir um URL para o qual o formulário será enviado para processamento.

Os elementos que admitem o uso desse atributo são: `button` e `input`. Assim, o URL para processamento é definido pelo valor desse atributo inserido em um desses dois elementos.

O exemplo a seguir esclarece a sintaxe e uso desse atributo.

► HTML

```
<form method="get">
    <p><label>Nome: <input type="text" name="nome"></label></p>
    <p><label>Email: <input type="text" name="email"></label></p>
    <p><label>Mensagem: <textarea name="msg"></textarea></label></p>
    <p><input type="submit" name="enviar" value="enviar" formaction="processa-form.php"></p>
</form>
```

6.2.8 formenctype

Esse atributo tem o mesmo efeito do atributo `enctype` para o elemento `form`, ou seja, destina-se a definir o tipo de conteúdo que está sendo enviado pelo formulário.

Os elementos que admitem o uso desse atributo são: `button` e `input`. Assim, a codificação é definida pelo valor desse atributo inserido em um desses dois elementos.

O exemplo a seguir esclarece a sintaxe e uso desse atributo.

► HTML

```
<form method="post" action="processa-form.php">
    <p><label>Arquivo: <input type="file" name="arq"></label></p>
    ...
    <p><input type="submit" name="enviar" value="enviar" formenctype="multipart/form-data"></p>
</form>
```

6.2.9 formmethod

Esse atributo tem o mesmo efeito do atributo `method` para o elemento `form`, ou seja, destina-se a definir o método usado para enviar os dados do formulário.

Os elementos que admitem o uso desse atributo são: `button` e `input`. Assim, o método é definido pelo valor desse atributo inserido em um desses dois elementos.

O exemplo a seguir esclarece a sintaxe e uso desse atributo.

► HTML

```
<form action="processa-form.php">
    <p><label>Nome: <input type="text" name="nome"></label></p>
    <p><label>Email: <input type="text" name="email"></label></p>
    <p><label>Mensagem: <textarea name="msg"></textarea></label></p>
    <p><input type="submit" name="enviar" value="enviar" formmethod="get"></p>
</form>
```

6.2.10 `formnovalidate`

Esse atributo tem o mesmo efeito do atributo `novalidate` para o elemento `form`, ou seja, destina-se a desabilitar a validação dos dados do formulário. (O atributo `novalidate` é novo na HTML5. Ver 6.2.15).

Os elementos que admitem o uso desse atributo são: `button` e `input`. Assim, a necessidade ou não de se validar os dados é definida pela inserção desse atributo em um desses dois elementos.

Trata-se de um atributo booleano, ou seja, é ou não definido para um elemento.



Não confundir atributo booleano com valor booleano para o atributo. Neste atribuem-se os valores `true` ou `false` e naqueles a presença ou ausência do atributo sem declaração de valor é suficiente.

O exemplo a seguir esclarece a sintaxe e uso desse atributo.

► HTML

```
<form method="post" action="processa-form.php">
    <p><label>Nome: <input type="text" name="nome"></label></p>
    <p><label>Email: <input type="text" name="email"></label></p>
    <p><label>Mensagem: <textarea name="msg"></textarea></label></p>
    <p><input type="submit" name="enviar" value="enviar"></p>
    <p><input type="submit" name="gravar" value="Salvar" formnovalidate></p>
</form>
```

6.2.11 `formtarget`

Esse atributo define o destino do documento que será aberto após o envio do formulário.

Os elementos que admitem o uso desse atributo são: `button` e `input`. Assim, o destino do documento é definido pela inserção desse atributo em um desses dois elementos.

Os valores possíveis para esse atributo são: `_blank`, `_top`, `_self` e `_parent` ou qualquer nome no contexto da janela atual.

6.2.12 list

Esse atributo é para uso exclusivo com o elemento `input` e destina-se a definir uma lista de opções para preenchimento de um campo de texto. O seu valor deve ser igual ao valor do `id` de um elemento `datalist`. Esse atributo foi estudado com detalhes em 5.2.17.

6.2.13 max

Esse atributo é para uso exclusivo com os elementos `meter`, `progress` e `input` e destina-se a definir um valor máximo. O uso desse atributo para os elementos `meter` e `progress` foi estudado em 2.4.17 e 2.4.20 respectivamente.

Para o elemento `meter` ele destina-se a definir o valor máximo da escala à qual a medida marcada pertence; se for omitido, será considerado igual a 1; para o elemento `progress` ele destina-se a definir o quanto é necessário para realizar uma tarefa. Usar esse atributo com o elemento `input` define um valor numérico máximo permitido como entrada no campo. Usado em conjunto com o atributo `min` estabelece uma faixa de entrada de dados no campo.

O exemplo a seguir esclarece a sintaxe e uso desse atributo.

► HTML

```
<input name="quantidade" required type="number" min="10" max="20" value="15">
```



[c6-max-min.html]

Nesse exemplo, quando a página é carregada, o campo está preenchido com o valor 15; ao usuário é permitido entrar valores compreendidos entre 10 e 20 inclusive. Esse exemplo está disponível no site do livro para visualização no navegador Opera ou posteriores.

6.2.14 min

Esse atributo é para uso exclusivo com os elementos `meter` e `input` e destina-se a definir um valor mínimo. O uso desse atributo para o elemento `meter` define o val

mínimo da escala à qual a medida marcada pertence; se for omitido, será considerado igual a 0. Ver 2.4.17.

Usar esse atributo com o elemento `input` define um valor numérico mínimo permitido como entrada no campo. Usado em conjunto com o atributo `max` estabelece uma faixa de entrada de dados no campo. Ver 6.2.13.

6.2.15 novalidate

Esse atributo é para uso exclusivo com o elemento `form` e tem o mesmo efeito do elemento `formnovalidate` que foi estudado em 6.2.10.

6.3 Atributo type para elemento input

A HTML5 criou uma série de novos valores para o atributo `type` do elemento `input`. Basicamente, os novos valores definem o tipo de dado esperado pelo controle. Esse atributo e seu valor destinam-se a informar ao agente de usuário o tipo de dado esperado e fazer com que seja disparado o mecanismo de validação nativo do navegador.

Os valores do atributo `type` previstos nas Recomendações para a HTML4 são 10, a saber: `text`, `password`, `checkbox`, `radio`, `submit`, `reset`, `file`, `hidden`, `image`, `button`.

A HTML5 manteve os 10 valores citados e acrescentou mais treze, conforme estudados a seguir.

6.3.1 search

Destina-se a definir um controle `input` do tipo busca. O efeito desse atributo é alterar a estilização e comportamento do controle diferenciando-o dos demais controles `input`. O modo como o controle é estilizado e seu comportamento fica por conta do fabricante do navegador.

Por exemplo: o navegador Chrome 90, que oferece suporte para esse atributo, apresenta uma letra “x” à direita da área de digitação do controle que se destina a fazer desaparecer o texto colocado como placeholder com uso do atributo `value` quando o usuário dá o foco ao controle. Não sendo definido o placeholder, a letra “x” aparece tão logo o usuário comece a digitar no controle.

O exemplo a seguir, disponível no site do livro, demonstra o uso desse valor para o atributo `type` de controles `input` usando dois campos: um com e outro sem placeholder.

► HTML

```
<h3>Com placeholder</h3>
<input name="q" type="search" value="palavra-chave">
<h3>Sem placeholder</h3>
<input name="q" type="search">
```



[c6-search.html]

6.3.2 tel

Destina-se a definir um controle `input` para a coleta de um número de telefone. Uma vez que números de telefone variam em formato, dependendo da região ou país, esse valor de atributo não força a entrada de uma sintaxe particular, ele apenas identifica a destinação do controle.

A validação de um controle para número de telefone deve ser feita com uso do atributo `pattern` ou do novo método `setCustomValidity()` disponível como mecanismo de JavaScript nativo do navegador.

O exemplo a seguir, disponível no site do livro, demonstra o uso desse valor para o atributo `type` de controles `input` e a respectiva validação com uso do atributo `pattern`.

► HTML

```
<form>
<p><label>Telefone: <input required name="tel" pattern="\\([0-9]{2}\\)\\s[0-9]{4}-[0-9]{4}"></label>
   (xx) xxxx-xxxx</p>
<p><input type=submit value=Enviar></p>
</form>
```



[c6-pattern.html]

Esse exemplo está disponível no site do livro e deverá ser visualizado no navegador Opera que suporta o atributo. O atributo `pattern` recebe como valor uma expressão regular construída com sintaxe JavaScript e o que for digitado no controle deverá satisfazer à expressão. No exemplo a expressão casa com números começando com (xx) seguido de um espaço em branco e duas vezes quatro números separados por um traço. Deve-se informar ao usuário o formato esperado para a entrada do número de telefone no controle, que, no caso do exemplo, foi (xx) xxxx-xxxx.

Outra opção para validar o dado inserido em um controle pelo usuário é com uso do novo método JavaScript `setCustomValidity()` conforme mostrado a seguir.

► JavaScript

```
<script>
    var padrao = /^[0-9]{2}\)[\s][0-9]{4}-[0-9]{4}$/;
    function validar(input) {
        if (!padrao.test(input.value)) {
            // Mensagem de erro
            input.setCustomValidity(
                'O número (' + input.value + ') não está no formato requerido. Por favor corrija.');
        } else {
            // Limpa mensagem de erro
            input.setCustomValidity('');
        }
        document.getElementById('msg-erro').innerHTML = input.validationMessage;
    }
</script>
```

► HTML

```
<form>
    <p><label>
        Telefone: <input required name="tel" onblur="validar(this)">
    </label> (xx) xxxx-xxxx</p>
    <div id="msg-erro" style="color:red;"></div>
    <input type="submit" value="OK">
</form>
```



[c6-setCustomValidity.html]

Esse exemplo está disponível no site e deverá ser visualizado nos navegadores Opera ou Chrome.

6.3.3 URL

Destina-se a definir um controle `input` para receber um URL absoluto.

Entre outras aplicações, podemos usar essa funcionalidade para fornecer uma indicação visual ao usuário sobre a validade do URL digitado no controle.

O exemplo a seguir, disponível no site do livro, demonstra esse uso. Nele usamos as pseudoclasses para o elemento `input` previstas nas CSS3 `:valid` e `:invalid` que identificam se um controle recebeu ou não um dado válido para seu tipo.

► CSS

```
<style>
.cssval:valid {
```

```

background:#0f0;
}
.cssval:invalid {
background:#f00;
color:#fff
}

```

► **HTML**

```

<form>
<p><label>URL: <input type=url name=url class="cssval"></label></p>
</form>

```



[c6-url.html]

As regras CSS aplicam uma cor de fundo ao elemento `input` identificado pela classe `cssval`. A cor será verde (#0f0) se o dado digitado no controle for válido e vermelha (#f00) caso contrário.

O comportamento de mudança de cor fica por conta de cada navegador e, na época que este livro foi escrito, tal comportamento variava. Todos os navegadores apresentam o controle no seu estado inicial na cor verde. Ao se digitar o primeiro caractere, seja ele qual for, a cor muda para vermelho e só volta para verde se o usuário completou a sequência digitada igual a `http:` nos navegadores Opera e Firefox e a sequência `http://` no navegador Chrome.

Esse comportamento pode confundir o usuário e seria melhor que o campo permanecesse verde se a string digitada inicialmente fosse `http://`. Vamos aguardar para ver como será a implementação definitiva do comportamento.

6.3.4 email

Destina-se a definir um controle `input` para receber um endereço de e-mail.

O exemplo a seguir, disponível no site do livro, demonstra o uso desse valor para o atributo `type` de controles `input` e a respectiva validação por mecanismos nativos do navegador.



```

<form>
<p><label>E-mail: <input type=email required name=email></label></p>
<p><input type=submit value=Enviar></p>
</form>

```



[c6-email.html]

O mecanismo de validação do navegador identifica a presença de uma string com formato de um endereço de e-mail para validar. Isso significa que um endereço de e-mail inexistente, desde que de acordo com aquele formato, valida normalmente. Por exemplo: a@a.a passa na validação.

6.3.5 `datetime`

Destina-se a definir um controle `input` para receber uma string representando uma data e hora UTC. A representação do grupo data/hora fica a cargo do agente de usuário em um formato conveniente considerando o local e o usuário. Para isso o agente de usuário normalmente apresentará um widget do tipo date picker quando for dada entrada no controle da data e um campo com slider do tipo seta para entrada da hora.

O exemplo a seguir, disponível no site do livro, demonstra o uso desse valor para o atributo `type` de controles `input`. Nele você poderá observar a renderização do widget date picker em navegadores que suportam esse valor de atributo, tal como o Opera, por exemplo.

► HTML

```
<form>
  <p><label>Data/hora: <input type=datetime name="datahora"></label></p>
  <p><input type=submit value=Enviar></p>
</form>
```



[c6-datetime.html]

6.3.6 `date`

Destina-se a definir um controle `input` para receber uma string representando uma data. A representação da data fica a cargo do agente de usuário em um formato conveniente considerando o local e o usuário. Para isso o agente de usuário normalmente apresentará um widget do tipo date picker quando for dada entrada no controle.

O exemplo a seguir, disponível no site do livro, demonstra o uso desse valor para o atributo `type` de controles `input`. Nele você poderá observar a renderização do widget date picker em navegadores que suportam esse valor de atributo, tal como o Opera, por exemplo.

► HTML

```
<form>
    <p><label>Data/hora: <input type=date name="data"></label></p>
    <p><input type=submit value=Enviar></p>
</form>
```



[c6-date.html]

6.3.7 time

Destina-se a definir um controle `input` para receber uma string representando uma hora. Normalmente o agente de usuário apresentará um slider do tipo seta para entrada da hora.

O exemplo a seguir, disponível no site do livro, demonstra o uso desse valor para o atributo `type` de controles `input`. Nele você poderá observar a renderização do slider do tipo seta em navegadores que suportam esse valor de atributo, tal como o Opera, por exemplo.

► HTML

```
<form>
    <p><label>Hora: <input type=time name="hora"></label></p>
    <p><input type=submit value=Enviar></p>
</form>
```



[c6-time.html]

6.3.8 month

Destina-se a definir um controle `input` para receber uma string representando um mês do ano. A representação do mês fica a cargo do agente de usuário em um formato conveniente considerando o local e o usuário; para tanto, o agente de usuário normalmente apresentará um widget do tipo date picker quando for dada entrada no controle.

O exemplo a seguir, disponível no site do livro, demonstra o uso desse valor para o atributo `type` de controles `input`. Nele você poderá observar a renderização do widget date picker em navegadores que suportam esse valor de atributo, tal como o Opera, por exemplo.

► HTML

```
<form>
  <p><label>Mês: <input type=month name="mes"></label></p>
  <p><input type=submit value=Enviar></p>
</form>
```



[c6-month.html]

6.3.9 week

Destina-se a definir um controle `input` para receber uma string representando uma semana do ano. A representação da semana fica a cargo do agente de usuário em um formato conveniente considerando o local e o usuário; para tanto, o agente de usuário normalmente apresentará um widget do tipo date picker quando for dada entrada no controle.

O exemplo a seguir, disponível no site do livro, demonstra o uso desse valor para o atributo `type` de controles `input`. Nele você poderá observar a renderização do widget date picker em navegadores que suportam esse valor de atributo, tal como o Opera, por exemplo.

► HTML

```
<form>
  <p><label>Semana: <input type=week name="semana"></label></p>
  <p><input type=submit value=Enviar></p>
</form>
```



[c6-week.html]

6.3.10 datetime-local

Destina-se a definir um controle `input` para receber uma string representando uma data e hora local. A representação do grupo data/hora fica a cargo do agente de usuário em um formato conveniente considerando o local e o usuário. Para isso o agente de usuário normalmente apresentará um widget do tipo date picker quando for dada entrada no controle da data e um campo com slider do tipo seta para entrada da hora.

O exemplo a seguir, disponível no site do livro, demonstra o uso desse valor para o atributo `type` de controles `input`. Nele você poderá observar a renderização do widget date picker em navegadores que suportam esse valor de atributo, tal como o Opera, por exemplo.

► HTML

```
<form>
    <p><label>Data/hora Local: <input type=datetime-local name="datahoralocal"></label></p>
    <p><input type=submit value=Enviar></p>
</form>
```



[c6-datetime-local.html]

6.3.11 number

Destina-se a definir um controle `input` para receber um número. Normalmente o agente de usuário apresentará um slider do tipo seta para entrada do número.

O exemplo a seguir, disponível no site do livro, demonstra o uso desse valor para o atributo `type` de controles `input`. Nele você poderá observar a renderização do slider do tipo seta em navegadores que suportam esse valor de atributo, tal como o Opera, por exemplo.

► HTML

```
<form>
    <p><label>Número: <input type=number name="numero"></label></p>
    <p><input type=submit value=Enviar></p>
</form>
```



[c6-number.html]

6.3.12 range

Destina-se a definir um controle `input` para receber um número contido dentro de um intervalo. Normalmente o agente de usuário apresentará um slider do tipo controle deslizante para a entrada do número.

O exemplo a seguir, disponível no site do livro, demonstra o uso desse valor para o atributo `type` de controles `input`. Nele você poderá observar a renderização do slider do tipo controle deslizante em navegadores que suportam esse valor de atributo, tal como o Opera, por exemplo.

► HTML

```
<form>
    <p><label>Número: <input type=range min="5" max="20" value="12" step="1" name=numero></label></p>
    <p><input type=submit value=Enviar></p>
</form>
```



[c6-range.html]

No exemplo mostrado foi renderizado um slider na horizontal, pois esse é o modo de renderização-padrão. Contudo, podemos forçar a renderização na vertical com uso de regras de estilo. Se definirmos um valor para a largura do input menor que o valor da sua altura, alteramos o modo de renderização, como mostrado no exemplo a seguir.

► CSS

```
<style>
  input {
    width: 40px;
    height: 120px;
  }
</style>
```

■ HTML

```
<form>
  <p><label>Número: <input type=range min="1" max="100" value="20" step="2" name=numero></label></p>
  <p><input type=submit value=Enviar></p>
</form>
```



[c6-range-vertical.html]

6.3.13 color

Destina-se a definir um controle `input` para receber o código sRGB de uma cor. Normalmente o agente de usuário apresentará um widget do tipo color-pick para a entrada da cor.

O exemplo a seguir, disponível no site do livro, demonstra o uso desse valor para o atributo `type` de controles `input`. Nele você poderá observar a renderização do color-pick em navegadores que suportam esse valor de atributo, tal como o Opera, por exemplo.

► HTML

```
<form>
  <p><label>Cor: <input type=color></label></p>
  <p><input type=submit value=Enviar></p>
</form>
```



[c6-color.html]



CAPÍTULO 7

Geolocalização

Neste capítulo, mostraremos as funcionalidades da API Geolocation. É comum afirmar que geolocalização é uma funcionalidade da HTML5, mas tal afirmativa não está correta. A API para Geolocation foi implementada pelo W3C que publicou o seu primeiro Rascunho de Trabalho em 22 de dezembro de 2008; desde então, vem desenvolvendo sua especificação que atualmente se encontra em fase de Candidata à Recomendação e é uma especificação sem qualquer relação e independente da HTML5.

7.1 Introdução

Geolocalização é uma funcionalidade que, apesar de ter sido lançada no ano de 2008, ganhou destaque e começou a ser difundida na mesma época em que a HTML5 ganhou seu espaço e começou a ser estudada e aplicada pela comunidade de desenvolvimento web. Talvez essa coincidência tenha sido a responsável por ter-se criado a falsa ideia de que geolocalização faz parte da HTML5.

A ideia marcou tanto que sempre que surge um tema relacionado à HTML5 como um todo, o assunto Geolocation vem à discussão. Assim, antecipando a um anseio do leitor, suponho que ele espere encontrar esse assunto neste livro e por essa razão o presente capítulo é dedicado à API Geolocation.

A especificação que trata da geolocalização define uma API destinada a oferecer acesso via script às informações relacionadas à localização geográfica do dispositivo de usuário que hospeda a funcionalidade.

A captação dos parâmetros de localização geográfica se faz com uso de GPS, sinais de redes tal como endereços de IP, redes sem fio, identificação por sinais de rádio (RFID), Bluetooth, identificadores GSM/CDMA ou ainda com uso de dados fornecidos diretamente pelo usuário.

Segundo as especificações para a API Geolocation, nenhuma garantia é fornecida de que os dados retornados pelas funcionalidades da API forneçam a localização exata do dispositivo do usuário.

A API foi projetada para prover três estados de localização do dispositivo de usuário, a saber: a posição estática atual, a localização dinâmica obtida por atualização repetida e contínua da posição atual e a possibilidade de recuperar uma posição anteriormente armazenada no dispositivo do usuário (posição em cache).

7.2 Segurança e privacidade

Obter a localização do dispositivo do usuário na maioria dos casos implica obter a localização do próprio usuário, comprometendo assim sua privacidade.

Uma implementação da API Geolocation em conformidade com as especificações do W3C deve obrigatoriamente fornecer ao usuário um mecanismo capaz de protegê-lo contra a invasão de sua privacidade. Tal mecanismo deverá assegurar ao usuário a opção de escolha por compartilhar ou não os dados relativos à sua localização geográfica. Assim, agentes de usuário não devem enviar, para qualquer que seja o Web Site, informações de posição sem o expresso consentimento do usuário.

O mecanismo de proteção da privacidade do usuário deve prever uma interface que informe claramente ao usuário a tentativa de obtenção da sua posição geográfica, fornecendo ainda o URI do documento web que está tentando obter a permissão e as opções, ao usuário, de aceitar ou rejeitar o compartilhamento da sua posição geográfica.

Na figura 7.1 a seguir podemos observar a interface de proteção de privacidade do usuário no navegador Firefox 4.

A especificação do W3C para a API Geolocation prevê ainda que a implementação deverá prover um mecanismo para revogar, a qualquer momento, uma permissão de compartilhamento de posição que tenha sido dada anteriormente. No navegador Firefox 4, para revogar uma permissão, acione o menu **Ferramentas > Propriedades da página > Permissões** e marque a opção **Perguntar** no item **Compartilhar localização**.

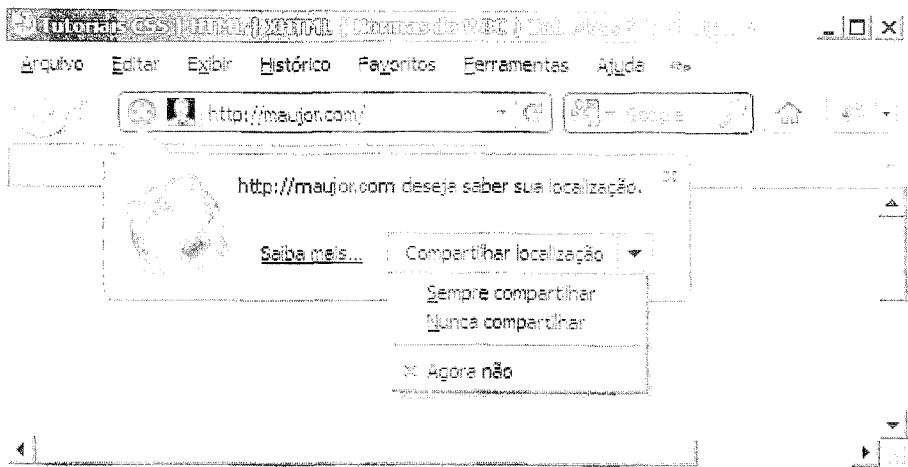


Figura 7.1 – Interface de proteção de privacidade.

7.3 Interface Geolocation

A API Geolocation está implementada em um agente de usuário que contenha o objeto JavaScript `navigator` se existir o objeto `geolocation` filho de `navigator`. Assim, para testar a existência da funcionalidade Geolocation em um dispositivo, devemos testar a existência do objeto `geolocation` como mostrado no código a seguir.

► JavaScript

```
if (navigator.geolocation) {
    alert("Este navegador suporta a funcionalidade Geolocation");
    /* Faça alguma coisa com Geolocation */
} else {
    alert("Este navegador não suporta a funcionalidade Geolocation");
}
```

[c7-teste-geolocation.html]

O arquivo de teste para o objeto `geolocation` encontra-se disponível no site do livro.

Se você estiver usando a biblioteca `Modernizr` poderá, opcionalmente, testar a existência da funcionalidade Geolocation com uso do objeto `Modernizr` como mostrado a seguir.

► JavaScript

```
<script>
if (Modernizr.geolocation) {
    alert("Este navegador suporta a funcionalidade Geolocation");
    /* Faça alguma coisa com Geolocation */
} else {
    alert("Este navegador não suporta a funcionalidade Geolocation");
}
</script>
```

7.4 Métodos do objeto geolocation

O objeto `geolocation` contém três métodos para a determinação e manipulação da posição geográfica do agente de usuário, conforme listados a seguir.

- `getCurrentPosition()`
- `watchPosition()`
- `clearWatch()`

Cada um desses métodos admite parâmetros obrigatórios e opcionais, conforme veremos detalhadamente a seguir.

7.4.1 `getCurrentPosition(callback sucesso, callback erro, {opções posição})`

Esse método admite um, dois ou três parâmetros. Os dois primeiros parâmetros são funções callback a serem chamadas nos casos de sucesso ou falha na obtenção da posição do dispositivo de usuário e o terceiro parâmetro é um objeto contendo pares nome/valor de propriedades da posição.

callback sucesso

A função callback para sucesso recebe como parâmetro um objeto contendo, entre outras estudadas adiante, duas propriedades das coordenadas geográficas do dispositivo de usuário que são a latitude e a longitude. A sintaxe para obtenção dessas propriedades é mostrada a seguir.

```
function sucesso (position) {
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;
```

No código mostrado o parâmetro objeto da função callback para sucesso foi denominado `position`.

O exemplo a seguir, disponível no site do livro, mostra o script para obtenção da posição do dispositivo de usuário e, em caso de sucesso, retorna uma mensagem informando as coordenadas geográficas do dispositivo.

► JavaScript

```
<script>
    function obterPosicao() {
        if (navigator.geolocation) {
            navigator.geolocation.getCurrentPosition(sucesso);
        } else {
            alert("Este navegador não suporta a funcionalidade Geolocation");
        }
    }
    function sucesso(position) {
        var latitude = position.coords.latitude;
        var longitude = position.coords.longitude;

        var mensagem = "As suas coordenadas são: <br>";
        mensagem += "Latitude: " + latitude + "<br>";
        mensagem += "Longitude: " + longitude;
        document.getElementById('msg').innerHTML = mensagem;
    }
</script>
```

► HTML

```
<p><input type=button value="Clique para obter posição" onclick="obterPosicao()"></p>
<p id="msg" class="dest"></p>
```



[c7-sucesso.html]

Além da latitude e da longitude, o objeto `position` fornece ainda as seguintes informações sobre as coordenadas de posicionamento do agente de usuário:

- `position.coords.altitude` – Fornece a altitude do agente de usuário em metros. Se não for possível determinar a altitude, o valor retornado deverá ser `null`.
- `position.coords.accuracy` – Fornece a precisão, em metros, das coordenadas geográficas latitude e longitude calculadas pela API.
- `position.coords.altitudeAccuracy` – Fornece a precisão, em metros, da coordenada geográfica altitude calculada pela API. Se não for possível determinar a precisão da altitude, o valor retornado deverá ser `null`.

- `position.coords.heading` – Fornece a direção de deslocamento do dispositivo de usuário. O valor dessa propriedade é um ângulo compreendido entre 0 e 360 graus indicando, no sentido horário, a direção de deslocamento em relação ao norte verdadeiro. Se não for possível determinar a direção de deslocamento, o valor retornado deverá ser `null`. Se o dispositivo de usuário estiver em situação estacionária (não em deslocamento), o valor retornado deverá ser `Nan`.
- `position.coords.speed` – Fornece a velocidade de deslocamento do agente de usuário em metros por segundo. Se não for possível determinar a velocidade, o valor retornado deverá ser `null`.
- `position.timestamp` – Essa é uma propriedade do objeto `position` que retorna um valor numérico em milissegundos representando um DOMTimeStamp do momento em que a posição do agente de usuário foi determinada.

O exemplo a seguir, disponível no site do livro, mostra o script para obtenção de uma mensagem informando os valores retornados para as propriedades descritas.

Dessas propriedades os navegadores deverão implementar a latitude, longitude e precisão das coordenadas. As demais propriedades ficam a cargo de cada implementação. Assim, no exemplo mostrado a seguir, os valores retornados para as demais propriedades não citadas anteriormente diferem de navegador para navegador.

► JavaScript

```
<script>
    function propriedadesPosition() {
        if (navigator.geolocation) {
            navigator.geolocation.getCurrentPosition(sucesso);
        } else {
            alert("Este navegador não suporta a funcionalidade Geolocation");
        }
    }

    function sucesso(position) {
        var latitude = position.coords.latitude;
        var longitude = position.coords.longitude;
        var altitude = position.coords.altitude;
        var precisao = position.coords.accuracy;
        var precisaoAltitude = position.coords.altitudeAccuracy;
        var direcao = position.coords.heading;
        var velocidade = position.coords.speed;
        var tempo = position.timestamp;
        var mensagem = "As propriedades de Position são: <br>";
        mensagem += "Latitude: " + latitude + "<br>";
    }
}
```

```

    mensagem += "Longitude: " + longitude + "<br>";
    mensagem += "Precisão das coordenadas: " + precisao + " metros<br>";
    mensagem += "<br>";
    mensagem += "Altitude: " + altitude + "<br>";
    mensagem += "Precisão da altitude: " + precisaoAltitude + " metros<br>";
    mensagem += "<br>";
    mensagem += "Direção: " + direcao + " graus<br>";
    mensagem += "Velocidade: " + velocidade + "<br>";
    mensagem += "Tempo: " + tempo + " milissegundos";
    document.getElementById('msg').innerHTML = mensagem;
}
</script>

```

► HTML

```

<p><input type=button value="Clique para obter propriedades" onclick="propriedadesPosition()"></p>
<p id="msg" class="dest"></p>

```

 [c7-propriedades-position.html]

callback erro

A função callback para erro admite um parâmetro destinado a coletar duas propriedades que são um código de erro e uma mensagem de erro. A sintaxe para obtenção dessas propriedades é mostrada a seguir.

```

function erro (err) {
    var codigo = err.code;
    var mensagem = err.message;
}

```

No código mostrado o parâmetro da função callback para erro foi denominado `err`.

Os códigos de erros e seus respectivos valores numéricos são listados a seguir.

- `PERMISSION_DENIED` (1) – Ocorre quando o usuário nega acesso à sua posição.
- `POSITION_UNAVAILABLE` (2) – Ocorre quando a rede está fora do ar ou não for possível estabelecer conexão com os satélites.
- `TIMEOUT` (3) – Ocorre quando a rede está ativa, mas o tempo de obtenção da posição torna-se muito longo.

A propriedade `message` retorna uma informação, em jargão técnico, sobre o erro ocorrido e é usada para fins de debug. Não deve ser apresentada ao usuário, devendo-se construir uma mensagem amigável para esse fim, como mostrado a seguir.

► JavaScript

```
function erro(err) {  
    switch (err) {  
        case 1 :  
            alert ("A permissão para obter a sua posição foi negada");  
            break;  
        case 2 :  
            alert ("Não foi possível estabelecer uma conexão para obter a sua posição.");  
            break;  
        case 3 :  
            alert ("Tempo esgotado");  
            break;  
        default:  
            alert("Não foi possível obter sua posição");  
    }  
}
```

O exemplo a seguir, disponível no site do livro, mostra o script para obtenção da posição do dispositivo de usuário e, em caso de erro, retorna uma mensagem informando o código do erro e uma mensagem ao usuário.

► JavaScript

```
function inspecionarErro() {  
    if (navigator.geolocation) {  
        navigator.geolocation.getCurrentPosition(sucesso,erro);  
    } else {  
        alert("Este navegador não suporta a funcionalidade Geolocation");  
    }  
}  
  
function erro(err) {  
    switch (err.code) {  
        case 1 :  
            var mensagemErro = "A permissão para obter a sua posição foi negada.";  
            break;  
        case 2 :  
            var mensagemErro = "Não foi possível estabelecer uma conexão para obter a sua posição.";  
            break;  
        case 3 :  
            var mensagemErro = "Tempo esgotado.";  
            break;  
        default:  
            var mensagemErro = "Não foi possível obter sua posição.";  
    }  
}
```

```
var codigoErro = err.code;
var mensagem = "Ocoreu um erro na determinação da posição: <br>";
mensagem += "Código do erro: " + codigoErro + "<br>";
mensagem += "Mensagem: " + mensagemErro;
document.getElementById('msg').innerHTML = mensagem;
}
function sucesso(position) {
    document.getElementById('msg').innerHTML = "Provoque um erro, negando ...";
}
```

► HTML

```
<p><input type=button value="Clique para inspecionar erro" onclick="inspecionarErro()"></p>
<p id="msg" class="dest"></p>
```



[c7-erro.html]

Ao consultar o arquivo, você deverá forçar um erro para visualizar o script em ação. No navegador Firefox, acione o menu **Ferramentas > Propriedades da página > Permissões**, desmarque a opção **Perguntar** e marque a opção **Bloquear** no item **Compartilhar localizações**. Com essa ação, você causará o erro **PERMISSION_DENIED** sempre que o script tentar obter sua posição. Clique o botão “Clique para inspecionar erro” no arquivo exemplo e uma mensagem de erro será apresentada. Não esqueça de desfazer o bloqueio no menu **Ferramentas** depois de concluir a visualização do exemplo.

{opções posição}

O terceiro parâmetro é um objeto JavaScript contendo informações a serem usadas na obtenção da posição do agente de usuário. Esse parâmetro é opcional e as informações passadas são as seguintes:

- **enableHighAccuracy** – informa ao dispositivo de usuário que ele deverá tentar obter a posição com a maior precisão possível. Em dispositivos móveis, a localização será feita com uso de GPS. Os valores possíveis para essa informação são `true` e `false` e o valor-padrão é `false`.
- **timeout** – define um tempo máximo, em milissegundos, para obtenção da posição. O valor-padrão é infinito.
- **maximumAge** – se o dispositivo de usuário guardou em cache uma posição previamente determinada, essa informação fornece um tempo, em milissegundos, abaixo do qual a posição em cache no dispositivo poderá ser aproveitada. O valor-padrão é zero, ou seja, não usar posição em cache.

Observe a seguir a sintaxe de uso desse parâmetro.

```
navigator.geolocation.getCurrentPosition(sucesso,erro, {  
    enableHighAccuracy: true,  
    timeout: 3000,  
    maximumAge: 300000  
});
```

Segundo esse exemplo, a obtenção da posição deverá ser feita com a maior precisão possível (depende do dispositivo de usuário), em, no máximo, 3s e se houver em cache uma posição que tenha sido obtida nos últimos 5 minutos, ela deverá ser usada.

7.4.2 **watchPosition(callback sucesso, callback erro, {opções posição})**

Esse método admite um, dois ou três parâmetros. Os dois primeiros parâmetros são funções callback a serem chamadas nos casos de sucesso ou falha na obtenção da posição do dispositivo de usuário e o terceiro parâmetro é um objeto contendo pares nome/valor de propriedades da posição.

A única diferença desse método para o método `getCurrentPosition()` é que ele obtém a posição do dispositivo de usuário continuamente. Conclui-se de imediato que é indicado para situações nas quais há um deslocamento ou movimentação do dispositivo.

Esse método é usado em conjunto com o método `clearWatch(id)` e seu funcionamento é semelhante ao funcionamento dos métodos `setInterval()` e `clearInterval()`.

O exemplo a seguir esclarece a sintaxe de uso desses métodos.

```
var positionId = navigator.geolocation.watchPosition(sucesso,erro, {enableHighAccuracy: true});  
function sucesso() {  
    // corpo da função sucesso  
}  
function erro() {  
    // corpo da função erro  
}  
function pararPosicionamento() {  
    // ...  
    botao.onclick = function() {  
        clearWatch(positionId);  
    }  
}
```

7.5 Google Maps API

7.5.1 Introdução

Conforme acabamos de estudar, os três métodos e as propriedades da API Geolocation retornam informações sobre a localização do agente de usuário. Somente isso e nada mais. Ao contrário do que muitos acreditam, a API não constrói mapas de localização e nem fornece qualquer funcionalidade de tratamento das informações retornadas.

Assim, se você deseja mostrar em um mapa a localização obtida com as informações fornecidas pela API Geolocation, terá de se valer de um serviço de mapas de terceiros e passar para a API do serviço escolhido as informações obtidas e esta se encarregará de renderizar devidamente o mapa.

O serviço do Google denominado Google Maps oferece várias APIs para uso público e gratuito, com inúmeros métodos e propriedades devidamente documentados em seu site. Para ilustrar alguns empregos da API Geolocation, mostraremos exemplos de integração e requisição de mapas do Google explicando de forma sucinta algumas das funcionalidades disponíveis em duas das suas APIs, bem como a sintaxe para requisitar mapas.

7.5.2 API do Google Static Maps

Essa API nos permite requisitar mapas estáticos. Esses mapas, gerados com as informações passadas para a API do Google, são imagens estáticas; por isso, não fornecem qualquer tipo de interação do mapa com o usuário. Elas simplesmente mostram a localização do dispositivo de usuário usando coordenadas de latitude e longitude retornadas pela API Geolocation.

A API do Google Static Maps retorna um URL parametrizado contendo o endereço da imagem do mapa gerado. A imagem é gerada nas extensões .gif, .jpg e .png e o desenvolvedor tem a opção de escolher uma dessas extensões ao inserir a imagem na página. De posse desse endereço, com uso de script, é fácil inserir o URL como valor do atributo `src` de um elemento `img` e obter a renderização do mapa personalizado de acordo com a latitude e longitude e os parâmetros do URL. Observe a seguir a sintaxe para uma marcação HTML capaz de inserir um mapa estático do Google.

```

```

Digite o código mostrado, para ilustrar a sintaxe, na barra de endereço do seu navegador e você verá um mapa contendo um trecho da praia de Copacabana, com seu centro nas coordenadas cartográficas, latitude: 22.975091 e longitude: -43.188762.

7.5.2.1 Parâmetros do URL

O URL que contém o mapa estático gerado pelo Google segue uma sintaxe-padrão, conforme mostrado anteriormente, e admite uma série de parâmetros que definem como o mapa deve ser apresentado. Veremos a seguir a descrição e o significado desses parâmetros.

Todos os URL das imagens de mapas estáticos têm a seguinte sintaxe:

`http://maps.google.com/maps/api/staticmap?parâmetros`

Os parâmetros são pares nome/valor separados por & e são descritos a seguir.

center

Parâmetro obrigatório caso não haja marcadores (estudados adiante). Seu valor é o par latitude, longitude do dispositivo do usuário e destina-se a definir a região terrestre onde se encontra o dispositivo de usuário.

zoom

Parâmetro obrigatório caso não haja marcadores (estudados adiante). Seu valor é um número inteiro e destina-se a definir o nível de ampliação do mapa. O valor 0 (zero) define a apresentação do mapa de todo o mundo e o valor 21 ou maior possibilita a visualização de construções individuais.

size

Parâmetro obrigatório. Seu valor é uma string no formato largura x altura destinada a definir as dimensões do mapa em pixels.

format

Parâmetro facultativo. Destina-se a definir o formato do arquivo de imagem. Os valores possíveis para esse parâmetro são: png ou png8, png32, gif, jpg e jpg-baseline. O valor-padrão é png. O valor `png-baseline` define uma imagem jpg não progressiva (indicada para impressão).

maptype

Parâmetro facultativo. Destina-se a definir o tipo de mapa a ser renderizado. Os valores possíveis para esse parâmetro são:

- **roadmap** – esse é o valor-padrão. Apresenta um mapa de rodovias no formato apresentado quando se solicita um mapa no site do Google Maps.
- **satellite** – apresenta um mapa-imagem captado por satélite.
- **terrain** – apresenta um mapa mostrando o relevo da terra com terreno e vegetação.
- **hybrid** – apresenta um mapa que é uma combinação de imagem captada por satélite e mapa de rodovias.

mobile

Parâmetro facultativo. Destina-se a definir se o mapa será ou não servido a dispositivos móveis. Os valores possíveis para esse parâmetro são **true** e **false** e o valor-padrão é **false**. A documentação do Google Maps recomenda que, para dispositivos móveis dotados de navegadores com recursos robustos, como iPhone e Android, não convém definir **mobile=true**.

language

Parâmetro facultativo. Destina-se a definir o idioma a ser usado para exibição dos marcadores do mapa.

markers

Parâmetro facultativo. Destina-se a definir marcadores a serem usados em locais específicos do mapa. Marcadores são aqueles balõezinhos coloridos que existem nos mapas do Google Maps.

A sintaxe geral para definição de valores para esse parâmetro é mostrada a seguir:

markers=estilos|local – **estilos** define os estilos para o marcador e **local** define a localização do marcador no mapa. Uma barra vertical deve separar **estilos** de **local**. Se for usado um marcador com estilização idêntica em diversos locais do mapa, a sintaxe é: **markers=estilos|local1|local2|local3|...**

A sintaxe para definir **estilos** é mostrada a seguir.

`size:valor|color:valor|label:valor`

- `size` – define o tamanho do marcador e os valores possíveis são: `tiny`, `mid` e `small` que definem os tamanhos mínimo, médio e pequeno. O valor-padrão é `normal`.
- `color` – define a cor do marcador e os valores da cor devem ser especificados no formato 24 bits (por exemplo, `0xff0000`) ou com uso de uma das seguintes palavras-chave: `black`, `brown`, `green`, `purple`, `yellow`, `blue`, `gray`, `orange`, `red`, `white`.
- `label` – define um rótulo a ser inserido no marcador. O valor para esse parâmetro é um caractere alfanumérico {A-Z, 0-9} em caixa-alta. Não é possível inserir rótulo em marcadores de tamanhos `tiny` e `small`. Rótulos são mostrados somente em marcadores do tamanho-padrão e `mid`. O valor-padrão é um pequeno círculo na cor preta.

Você não está restrito ao uso dos marcadores-padrão do Google Maps (balões-zinhos coloridos) e pode usar marcadores personalizados. Para isso construa seu marcador nos formatos `png`, `jpg` ou `gif` (o recomendado é `png`), com tamanho máximo de 64 x 64 pixel, hospede a imagem em um servidor e use a sintaxe a seguir.

`markers=icon: http://caminho para marcador|local`

Você pode usar até cinco marcadores personalizados diferentes em cada solicitação. Notar que a inserção de marcadores personalizados é ilimitada desde que respeitado o uso dos cinco.

Marcadores personalizados são renderizados, por padrão, com sombra. Para retirar a sombra use o parâmetro `shadow=false`.

path

Parâmetro facultativo. Destina-se a definir um caminho no mapa.

Esse parâmetro não se aplica para o nosso caso e não entraremos em maiores detalhes sobre ele.

visible

Parâmetro facultativo. Destina-se a definir áreas do mapa que permanecem invisíveis.

Esse parâmetro não se aplica para o nosso caso e não entraremos em maiores detalhes sobre ele.

sensor

Parâmetro obrigatório. Destina-se a informar se o aplicativo que está solicitando o mapa está usando um sensor (por exemplo: GPS) para obter a localização do usuário. Isso é importante para dispositivos móveis. Os valores possíveis são true e false.

O exemplo a seguir, disponível no site do livro, demonstra a requisição de um mapa estático centrado na posição em que se encontra o dispositivo de usuário.

► CSS

```
<style>
#mapa {
    width:420px;
    height:420px;
    display:none;
}
</style>
```

► JavaScript

```
<script>
function mostrarMapa() {
    if (navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(sucesso,erro);
    } else {
        alert("Este navegador não suporta a funcionalidade Geolocation");
    }
}
function erro(err) {
    switch (err.code) {
        case 1 :
            var mensagemErro = "A permissão para obter a sua posição foi negada.";
            break;
        case 2 :
            var mensagemErro = "Não foi possível estabelecer uma conexão para obter a sua posição.";
            break;
        case 3 :
            var mensagemErro = "Tempo esgotado.";
            break;
        default:
            var mensagemErro = "Não foi possível obter sua posição.";
    }
    var codigoErro = err.code;
    var mensagem = "Ocorreu um erro na determinação da posição: <br>";
    mensagem += "Código do erro: " + codigoErro + "<br>";
}
```

```
mensagem += "Mensagem: " + mensagemErro;
document.getElementById('msg').innerHTML = mensagem;
}

function sucesso(position) {
    var mapa = document.getElementById("mapa");
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;
    var url = "http://maps.google.com/maps/api/staticmap?center=";
    url += latitude +","+ longitude;
    url += "&zoom=15&size=400x400";
    url += "&markers=color:red|"+latitude +"," +longitude;
    url += "&sensor=true"
    mapa.src = url;
    mapa.style.display = "block";
}
</script>
```

► HTML

```
<section>
    <p><input type=button value="Clique para mostrar sua posição no mapa" onclick="mostrarMapa()"></p>
    <p id="msg" class="dest"></p>
</section>
<section>
    <img id="mapa" />
</section>
```



[c7-mapa-estatico-google.html]

7.5.3 API V3 do Google Maps

Essa API nos permite requisitar mapas dinâmicos. Esses mapas, gerados com as informações passadas para a API do Google, são, ao contrário dos mapas da API estática, mapas dinâmicos carregados assincronamente em tempo real e permitem integração com o usuário com todas as funcionalidades do Google Maps, inclusive street-view.

Explicamos a seguir as principais diretrizes para inserção de um mapa dinâmico, em uma página web, usando Geolocation.

Para receber um mapa dinâmico da API do Google Maps, uma página web precisa carregar a respectiva API. O carregamento é feito com uso do elemento `script` na seção `head` do documento apontando para um URL no site do Google, como mostrado a seguir.

```
<script type="text/javascript" src="http://maps.google.com/maps/api/js?sensor=true ou false">
</script>
```

No URL deverá obrigatoriamente ser passado o parâmetro `sensor` com o valor `true` ou `false`. Esse parâmetro informa se a busca pela posição do usuário se faz com ou sem o uso de um sensor (por exemplo: GPS em dispositivos móveis).

Na marcação HTML da página deverá constar um container para o mapa. O container em geral é um elemento `div` com um `id` definido. O `id` servirá de referência para que o script consiga identificar o container para o mapa. As dimensões do mapa serão iguais à largura e à altura do container, portanto é indispensável que as dimensões do container sejam definidas explicitamente com uso de CSS.

Notar que o container do mapa poderá ser criado diretamente na marcação da página ou com uso de JavaScript. Cabe ao desenvolvedor decidir sobre a adoção de um ou outro método de criação do container.

Tendo comentado a infraestrutura de marcação para inserção do mapa dinâmico, vejamos a seguir os métodos e propriedades da API do Google.

Devemos inicialmente criar um objeto literal contendo as opções para renderização do mapa. A sintaxe e valores possíveis para as propriedades do objeto são mostrados no exemplo seguir.

```
var opcoes = {  
    zoom: 8,  
    center: latLong,  
    mapTypeId: google.maps.MapTypeId.ROADMAP  
};
```

- `zoom` – define o nível de ampliação do mapa.
- `center` – define o centro do mapa nas coordenadas retornadas por Geolocation.
- `mapTypeId` – define o tipo inicial do mapa e os valores possíveis são: `ROADMAP`, `SATELLITE`, `TERRAIN`, `HYBRID`. Ver 7.5.2.1 subtítulo `maptyle`.

O valor `latLong` para a propriedade `center` é uma variável obtida com a sintaxe mostrada a seguir.

```
var latLong = new google.maps.LatLng(latitude, longitude)
```

Sendo `latitude` e `longitude` os valores retornados pela Geolocalização.

A API do Google nos fornece um objeto denominado `google.maps.Map`. A classe JavaScript que representa o mapa é `Map`. A definição de um mapa é feita com uso dessa classe. Criamos uma instância dessa classe e, em consequência, um mapa conforme a sintaxe mostrada a seguir.

```
google.maps.Map(opcoes)
```

Para inserir o mapa em um nó do DOM a sintaxe é:

```
google.maps.Map(nó, opcoes)
```

Estes são os passos básicos para inserção de um mapa. API do Google Maps prevê uma grande quantidade de métodos e propriedades que possibilitam personalizar seus mapas. Não é do escopo deste livro detalhar aquela API. Se você estiver interessado em aprofundar o assunto, consulte a documentação online no site do Google.

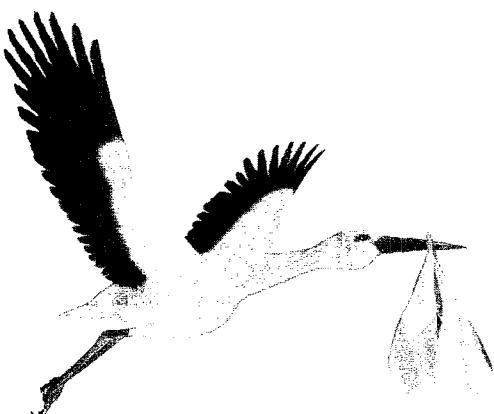
Observe a seguir um exemplo de inserção de mapa em uma página web, disponível no site do livro. Notar que no exemplo usamos, além dos passos básicos explicados, a inserção de um balão de diálogo com uma mensagem, apontando para as coordenadas do dispositivo de usuário retornadas por Geolocation. O balão foi inserido com uso de uma instância de objeto `new google.maps.InfoWindow()` da API do Google (não abordado anteriormente).

```
<script>
var infowindow = new google.maps.InfoWindow();
function mostrarMapa() {
    if (navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(sucesso,erro);
    } else {
        alert("Este navegador não suporta a funcionalidade Geolocation");
    }
}
function erro(err) {
    switch (err.code) {
        case 1 :
            var mensagemErro = "A permissão para obter a sua posição foi negada.";
            break;
        case 2 :
            var mensagemErro = "Não foi possível estabelecer uma conexão para obter a sua posição.";
            break;
        case 3 :
            var mensagemErro = "Tempo esgotado.";
            break;
        default:
            var mensagemErro = "Não foi possível obter sua posição.";
    }
    var codigoErro = err.code;
    var mensagem = "Ocoreu um erro na determinação da posição: <br>";
    mensagem += "Código do erro: " + codigoErro + "<br>";
    mensagem += "Mensagem: " + mensagemErro;
    document.getElementById('msg').innerHTML = mensagem;
}
```

```
function sucesso(position) {  
    var mapa= document.getElementById("mapa");  
    mapa.style.display = "block";  
    var opcoes = {  
        zoom: 15,  
        mapTypeId: google.maps.MapTypeId.ROADMAP  
    }  
    var map = new google.maps.Map(document.getElementById("mapa"), opcoes);  
    var latitude = position.coords.latitude;  
    var longitude = position.coords.longitude;  
    var local = new google.maps.LatLng(latitude,longitude);  
    contentString = "<br>Legal! Achei você usando a funcionalidade<br> Geolocation do W3C e o Google Maps.";  
    map.setCenter(local);  
    infowindow.setContent(contentString);  
    infowindow.setPosition(local);  
    infowindow.open(map);  
}  
</script>
```



[c7-mapa-dinâmico-google.html]



CAPÍTULO 8

Armazenamento de dados

Neste capítulo, mostraremos as funcionalidades para armazenamento de dados no lado do cliente. São previstas duas funcionalidades para esse fim: as APIs Web Storage e Web SQL Database. Os estudos e diretrizes para implementação dessas funcionalidades foram iniciados pelo WHATWG, o Grupo de Trabalho para a HTML5. Trata-se de tecnologias da HTML5 que posteriormente foram desvinculadas do WHATWG e repassadas ao W3C para desenvolvimento, tornando-se especificações independentes. Atualmente encontram-se na fase de Rascunho de Trabalho, sendo, portanto, tecnologias não estáveis e sujeitas a modificações. Web Storage é suportada pelos navegadores modernos, inclusive o Internet Explorer 8, e Web SQL Database não é suportada por navegadores Mozilla que implementam em seu lugar uma tecnologia similar denominada IndexedDB.

8.1 Web Storage

Web Storage é uma API destinada ao armazenamento de dados, persistentes, em clientes web, por exemplo: um navegador.

Na verdade, essa funcionalidade foi criada com a finalidade de corrigir a forma conceitual e precária do funcionamento de cookies de seção.

O conceito de cookie foi criado em 1994 por Lou Montulli da Netscape, que, naquela época, desenvolvia uma aplicação e-commerce para um cliente. Em resumo, trata-se de um mecanismo capaz de armazenar pequenas peças de dados gravadas em um arquivo de texto no computador do usuário. Um cookie é criado e apagado

em uma página web com uso de linguagem de programação no lado do servidor ou com JavaScript (no lado do cliente). Criado um cookie, uma requisição ao servidor por uma página web pode enviar a solicitação do dado armazenado, havendo a necessidade de comunicação entre o cliente web e o servidor para verificação da existência e uso do cookie.

Cookies, apesar de serem largamente empregados nos dias atuais, apresentam limitações que vão desde a reduzida capacidade de armazenagem até severas falhas de segurança. A API Web Storage foi introduzida na HTML5 com a finalidade de corrigir as limitações dos cookies.

As principais limitações com o uso de cookies são:

- Capacidade de armazenamento de apenas algo em torno de 4Kb para cada cookie.
- Armazenamento limitado a, em média, 20 cookies por domínio ou servidor e 300 cookies por navegador. Esses valores podem sofrer variações de acordo com o navegador.
- Inclusão de cookies em cada requisição HTTP implicando tráfego de dados desnecessário e consequente aumento de consumo de banda.
- Restrição ao uso de cookies para armazenamento de dados sensíveis, já que estarão expostos livremente nas transações entre cliente e servidor.
- Vulnerabilidade a ataques maliciosos.

A busca por soluções alternativas para o uso de cookies não começou com a HTML5. A Microsoft introduziu no Internet Explorer 5.5, lançado em julho de 2000, uma tecnologia denominada DHTML Behaviors que, em resumo, permite criar componentes contendo funcionalidades específicas ou comportamentos (behaviors) reusáveis em uma página web.

Um desses comportamentos criado pela Microsoft denomina-se userData Behavior que é uma alternativa proprietária a cookies. Essa alternativa, além de armazenar dados persistentes no lado do cliente, aumentou a capacidade de armazenamento para 64Kb por cookie e 640 Kb por domínio, além de ter criado mecanismos de bloqueio a ataques cross-domínio.

Em 2002 a Adobe criou e implementou no Flash 6 uma funcionalidade denominada LSO – *Local Shared Object* que é uma alternativa a cookies, também conhecida como Flash cookies, que permite armazenar até 100Kb de dados persistentes.

Em 2006 foi a vez da DOJO Toolkit implementar a funcionalidade denominada *AMASS – AJAX Massive Storage System*, inicialmente desenvolvida para o Flash, uma forma de armazenar dados no lado do cliente com capacidade de armazenamento de 100Kb aumentada indefinidamente a partir daí com a devida permissão do usuário.

Em 2007 o Google lançou o plugin para navegadores denominado Gears, que, entre outras funcionalidades, implementou uma API com funcionalidades para incorporar, no lado do cliente, um banco de dados SQL com base em SQLite capaz de armazenar, por domínio e com consentimento do usuário, uma quantidade infinita de dados, em tabelas SQL para banco de dados.

Cada uma das implementações de alternativas para cookies citadas depende de tecnologias proprietárias ou de plugins de terceiros para funcionar. Para resolver esse problema, a HTML5 resolveu prover uma API-padrão para implementação nativa nos navegadores sem dependência de softwares de terceiros. Daí surgiram as duas APIs que estudaremos neste capítulo.

A especificação do W3C para Web Storage descreve dois mecanismos similares ao mecanismo HTTP para cookies de sessão destinados a armazenar dados estruturados no cliente web. Esses mecanismos foram denominados `sessionStorage` e `localStorage` e serão mostrados a seguir.

Embora não definida explicitamente nas especificações, a capacidade de armazenamento local deve ser considerada de até 5Mb, pois é essa a quantidade de memória normalmente alocada pelos navegadores atuais. O navegador Safari, ao atingir esse limite de armazenagem, pergunta ao usuário se ele deseja ampliar o limite. Mas convenhamos que 5Mb é um valor muito alto se comparado aos 100Kb destinados aos cookies.

8.1.1 `sessionStorage`

É um mecanismo que foi projetado com a finalidade de armazenar dados para transação em um documento HTML, isto é, para cada novo documento HTML, é criada uma área de armazenamento de dados independente. Assim, é possível coexistirem múltiplas transações em diferentes janelas ou abas do navegador ao mesmo tempo, cada uma com seus conjuntos de dados independentes. Os dados pertencentes a um documento são excluídos quando o usuário fecha a janela ou a aba do navegador e não são transmitidos e nem se comunicam com outras janelas ou abas.

Esse comportamento resolve uma das limitações impostas pelos cookies: o fato de existirem em diferentes janelas ou abas ao mesmo tempo e somente serem excluídos por ocasião do fechamento do navegador, desde que não tenha sido prevista uma vida útil para eles.

Para observar essa diferença de comportamento na existência de cookies e dados armazenados com uso de `sessionStorage` disponibilizamos no site do livro um exemplo prático, conforme mostrado a seguir. Não se preocupe se a sintaxe para armazenar dados no objeto `sessionStorage`, usada no exemplo, parecer confusa e estranha para você. Adiante estudaremos com detalhes os métodos e propriedades desse objeto.

► JavaScript

```
<script src="cokieutil.js"></script> <!-- Objeto CookieUtil -->
<script>
function salvarCookie(name) {
    var valor = document.forms['cookieform'].cookievalue.value;
    var itemStorage = sessionStorage.setItem('dataStorage', valor);
    var valorItemStorage = sessionStorage.getItem('dataStorage');
    if (!valor)
        alert('Entre um valor para o cookie e dado.');
    else {
        CookieUtil.set(name, valor);
        var mensagem = 'Um cookie denominado "meuCookie" com o valor '+valor+'\n';
        mensagem += 'e um dado com o nome "dataStorage" com o valor '+valorItemStorage+'\n';
        mensagem += 'foram criados'
        alert(mensagem);
    }
}
function lerCookie(name) {
    var valorItemStorage = sessionStorage.getItem('dataStorage');
    var mensagem = 'Um cookie denominado "meuCookie" com o valor '+CookieUtil.get(name)+'\n';
    mensagem += 'e um dado com o nome "dataStorage" com o valor '+valorItemStorage+'\n';
    mensagem += 'foram lidos'
    alert(mensagem);
}
function apagarCookie(name) {
    CookieUtil.unset(name);
    sessionStorage.removeItem('dataStorage');
    alert('Cookie com o nome "meuCookie" e dado com o nome "dataStorage" foram apagados');
}
</script>
```

► HTML

```
<section>
<form name="cookieform" action="#"><p>
    <label>Escolha um valor para o cookie de nome "meuCookie".
    <br>O mesmo valor será usado para o dado de nome "dataStorage":</label><br>
    <input name="cookievalue" />
</p>
</form>
<p><a href="#" onclick="salvarCookie('meuCookie')">Armazenar cookie e dado</a></p>
<p><a href="#" onclick="lerCookie('meuCookie')">Ler cookie e dado</a></p>
<p><a href="#" onclick="apagarCookie('meuCookie')">Apagar cookie e dado</a></p>
</section>
```



[c8-cookies-sessionstorage.html]

8.1.2 localStorage

É um mecanismo que foi projetado com a finalidade de armazenar dados que persistem em diferentes janelas ou abas do navegador. Assim, é possível que uma transação compartilhe em diferentes janelas ou abas do navegador o mesmo conjunto de dados. Os dados são excluídos quando o usuário fecha o navegador e são transmitidos e se comunicam com outras janelas ou abas enquanto o navegador estiver aberto.

Esse mecanismo implementa um comportamento semelhante ao dos cookies que existem em diferentes janelas ou abas ao mesmo tempo e somente são excluídos por ocasião do fechamento do navegador.

O exemplo a seguir, disponível no site do livro, demonstra o funcionamento desse mecanismo de armazenamento de dados. Não se preocupe se a sintaxe para armazenar dados no objeto `localStorage`, usada no exemplo, parecer confusa e estranha para você. Adiante estudaremos com detalhes os métodos e propriedades desse objeto.

► JavaScript

```
<script>
function salvarDado(name) {
    var valor = document.forms['dadoform'].dadovalue.value;
    var itemStorage = localStorage.setItem('dataStorage', valor);
    var valorItemStorage = localStorage.getItem('dataStorage');
    if (!valor)
        alert('Entre um valor para o dado a armazenar.');
    else {
        mensagem = 'Um dado com o nome "dataStorage" com o valor '+valorItemStorage +'\n';
    }
}</script>
```

```

        mensagem += 'foi armazenado';
        alert(mensagem);
    }
}

function lerDados(name) {
    var valorItemStorage = localStorage.getItem('dataStorage');
    mensagem = 'Um dado com o nome "dataStorage" com o valor '+valorItemStorage +'\n';
    mensagem += 'foi lido';
    alert(mensagem);
}
function apagarDados(name) {
    localStorage.removeItem('dataStorage');
    alert('Dado com o nome "dataStorage" foi apagado');
}

```

▷ HTML

```

<section>
    <form name="dadosform" action="#">
        <p><label>Escolha um valor para o dado "meuDado":</label>
        <br><input name="dadovalue" /></p>
    </form>
    <p><a href="#" onClick="salvarDados('meuDado')">Armazenar dado</a></p>
    <p><a href="#" onClick="lerDados('meuDado')">Ler dado</a></p>
    <p><a href="#" onClick="apagarDados('meuDado')">Apagar dado</a></p>
</section>

```

 [c8-localstorage.html]

Para entender a diferença de comportamento entre `sessionStorage` e `localStorage` a especificação do W3C nos fornece dois fragmentos de script que adaptamos e transcrevemos a seguir. Notar que a finalidade dos exemplos mostrados é apenas ilustrar e fazer entender o funcionamento de cada um dos mecanismos. Para tornar o script funcional, será necessário completá-lo.

▷ JavaScript

```

<script>
<label>
    <input type="checkbox" onChange="sessionStorage.seguroViagem = checked">
    Quero fazer o seguro viagem
</label>
...

```

Esse exemplo mostra um controle, do tipo `checkbox`, de um formulário para a compra de passagem contido em uma página web. Se o usuário desejar comprar

com a passagem o seguro-viagem, deverá marcar o checkbox. Usou-se o evento `onchange` para atrelar ao checkbox o armazenamento de um dado denominado `seguroViagem` com o valor `checked`.

Se o usuário abrir outras janelas na mesma transação para a compra da passagem, nas quais no documento renderizado tenha sido inserido uma verificação do tipo:

```
if (session.seguroViagem) {...}
```

não correrá o risco de inadvertidamente contratar o seguro-viagem duas ou mais vezes, pois cada uma das janelas ou abas quando for aberta criará seu próprio objeto `sessionStorage` vazio.

O outro fragmento de script constante da especificação do W3C demonstra o uso de `localStorage` para criar um contador do número de vezes que um usuário carregou uma das páginas de um site.

► JavaScript

```
<p>Você visualizou esta página<span id="count"></span> vezes.</p>
<script>
    if (!localStorage.pageLoadCount)
        localStorage.pageLoadCount = 0;
        localStorage.pageLoadCount += 1;
        document.getElementById('count').textContent = localStorage.pageLoadCount;
</script>
```

Esse exemplo inicialmente verifica a existência de um dado com o nome `pageLoadCount` armazenado no objeto `localStorage` do navegador do usuário. Se não houver, o dado atribui a ele o valor zero e soma uma unidade. A seguir, insere esse valor para ser renderizado no contador. A partir daí, lê o dado armazenado e incrementa uma unidade.

Os objetos `sessionStorage` e `localStorage` admitem os métodos `setItem()`, `getItem()`, `removeItem()`, `clear()` e `key()` destinados a gravar um par nome/valor para um dado, recuperar o valor de um dado, remover o valor de um dado, remover todos os dados gravados anteriormente e recuperar o nome de um dado respectivamente. Contém, também, a propriedade `length`. Vamos estudar a seguir cada um desses métodos e propriedade.

`setItem('nome', 'valor')`

Esse método destina-se a criar um par nome/valor e armazená-lo na área de armazenamento do navegador. A sintaxe para esse método é mostrada a seguir.

```
sessionStorage.setItem('site', 'http://maujor.com');  
e  
localStorage.setItem('site', 'http://maujor.com');
```

Esse método admite uma sintaxe alternativa que é mostrada a seguir.

```
sessionStorage.site = 'http://maujor.com';  
e  
localStorage.site = 'http://maujor.com';
```

getItem('nome')

Esse método destina-se a recuperar o valor de um dado designado nome. A sintaxe para esse método é mostrada a seguir.

```
sessionStorage.getItem('site'); // retorna http://maujor.com  
e  
localStorage.getItem('site'); // retorna http://maujor.com
```

Esse método admite uma sintaxe alternativa que é mostrada a seguir.

```
sessionStorage.site; // retorna http://maujor.com  
e  
localStorage. site; // retorna http://maujor.com
```

removeItem('nome')

Esse método destina-se a apagar da área de armazenamento o par nome/valor de um dado designado nome. A sintaxe para esse método é mostrada a seguir.

```
sessionStorage.removeItem('site'); // remove o par site/http://maujor.com  
ou  
localStorage.removeItem('site'); // remove o par site/http://maujor.com
```

clear()

Esse método não requer parâmetros e se destina a apagar todo o conteúdo da área de armazenamento, isto é, esvazia o objeto `sessionStorage` ou `localStorage`. A sintaxe para esse método é mostrada a seguir.

```
sessionStorage.clear(); // esvazia o objeto sessionStorage  
ou  
localStorage.clear(); // esvazia o objeto localStorage
```

key(*n*)

Esse método destina-se a inspecionar o nome do dado armazenado na posição *n*. A sintaxe para esse método é mostrada a seguir.

```
sessionStorage.key(3); // retorna o nome do quarto dado armazenado
```

ou

```
localStorage.key(0); // retorna o nome do primeiro dado armazenado
```

Podemos usar uma sintaxe alternativa para os métodos que admitem o parâmetro nome, usando o retorno desse método como parâmetro. Observe a seguir a sintaxe alternativa para o método `get()`.

```
sessionStorage.getItem(sessionStorage.key(2)); // retorna o valor do terceiro dado armazenado
```

length

Essa propriedade destina-se a inspecionar a quantidade de dados (pares nome/valor) armazenados nos objetos `sessionStorage` e `localStorage`. Esses objetos podem ser tratados como um array associativo. A sintaxe para esse método é mostrada a seguir.

```
sessionStorage.length; // retorna um número maior ou igual a zero
```

ou

```
localStorage.length; // retorna um número maior ou igual a zero
```

8.1.3 Armazenagem com JSON

Por padrão, o tipo de dado armazenado localmente é `string`. Esse comportamento, se não for considerado, pode trazer resultados inesperados, por exemplo, quando armazenamos números e com eles realizamos operações matemáticas. O exemplo a seguir, disponível no site do livro, esclarece esse comportamento.

► JavaScript

```
<script>
    window.onload = function() {
        var campos = document.getElementsByTagName('input');
        var limpar = document.getElementById('limpar');
        limpar.onclick = function() {
            campos[0].value = 'clique';
            campos[1].value = 'clique';
        }
    }
    sessionStorage.um = 4;
```

```
sessionStorage.dois = 3;
</script>
```

► HTML

```
<section>
  <p>Armazenamento sem tratamento (como string) - Operação de concatenação.<br>
  4 + 3 = <input onclick="this.value = sessionStorage.getItem('um')>
  + sessionStorage.getItem('dois');" value="clique"></p>
  <p>Armazenamento com <code>parseInt()</code> (como número) - Operação de adição.<br>
  4 + 3 = <input onclick="this.value = parseInt(sessionStorage.getItem('um'))>
  + parseInt(sessionStorage.getItem('dois'));" value="clique"></p>
  <button type="button" id="limpar">Limpar</button>
</section>
```



[c8-numeros.html]

Nesse exemplo armazenamos dois números com uso do objeto `sessionStorage`. Os dois números são armazenados como dados do tipo `string`. Assim, se usarmos o operador `+` para os dois dados, estaremos concatenando duas strings. Para realizarmos a operação matemática soma, devemos transformar as strings em números com uso do método `parseInt()`.

Assim, o armazenamento de dados localmente transforma o tipo de objeto para `string`. Esse comportamento acaba trazendo inconvenientes que é o fato da necessidade de transformação do tipo de objeto após ter sido recuperado da armazenagem local. Vimos isso na prática no exemplo anterior, no qual ocorreu a armazenagem do objeto `Number` como `String` e a consequente necessidade do uso do método `parseInt()`, antes de realizar a operação adição. Uma maneira mais imediata de resolver o problema criado com a armazenagem em forma de `string` é usar a tecnologia `JSON`.

`JSON` permite que se use texto para representar objetos `JavaScript` e, em consequência, é possível armazenar localmente objetos e recuperá-los posteriormente com uso dos métodos `JSON.stringify()` e `JSON.parse()`. A sintaxe para armazenar e recuperar objetos com `JSON` é mostrada a seguir.

```
sessionStorage.setItem('ObjetoJson', JSON.stringify(ObjetoJson));      // armazena
JSON.parse(sessionStorage.getItem('ObjetoJson'));           // recupera
```

O exemplo a seguir é idêntico ao exemplo anterior, mas agora estamos usando `JSON` para recuperar o objeto `Number` armazenado como `string`.

► JavaScript

```
<script>
    window.onload = function() {
        var campos = document.getElementsByTagName('input');
        var limpar = document.getElementById('limpar');

        sessionStorage.setItem('a', 4);
        sessionStorage.setItem('b', 3);
        a = sessionStorage.getItem('a');
        b = sessionStorage.getItem('b');
        campos[0].onclick = function() {
            this.value = a+b;
        }

        var numeros = {
            a: 4,
            b: 3
        }
        sessionStorage.setItem('numeros', JSON.stringify(numeros));
        JSON.parse(sessionStorage.getItem('numeros'));
        campos[1].onclick = function() {
            this.value = numeros.a + numeros.b;
        }

        limpar.onclick = function() {
            campos[0].value = 'clique';
            campos[1].value = 'clique';
        }
    }
</script>
```

► HTML

```
<section>
    <p>Armazenar com sessionStorage simples (operação de concatenação):<br>
    4 + 3 = <input value="clique"></p>
    <p>Armazenar com sessionStorage e JSON (operação de adição):<br>
    4 + 3 = <input value="clique"></p>
    <button type="button" id="limpar">Limpar</button>
</section>
```



[c8-numeros-json.html]

8.1.4 Evento storage

O evento `storage` ocorre sempre que há uma alteração na área de armazenamento de dados definida pelo método `localStorage`.

As propriedades do evento `storage` são:

- `key` – é uma string com o nome do dado que foi modificado, adicionado ou removido.
- `oldValue` – o valor de um dado que tenha sido modificado ou null se o dado foi adicionado pela primeira vez.
- `newValue` – o valor adicionado para um dado ou null se o valor foi removido.
- `storageArea` – retorna o objeto `localStorage`.
- `url` – O endereço da página na qual se encontra o método que provocou o evento.

Não é possível cancelar o evento `storage`. Uma vez iniciada uma função callback para o evento, não há como interrompê-la. O evento não se propaga pelo efeito bolha.

Convém ressaltar, com ênfase, que, ao contrário da maioria dos eventos JavaScript, o evento `storage` não é disparado na mesma janela ou aba do navegador onde se encontra o disparador do evento, e sim em outras janelas ou abas abertas que pertençam ao mesmo domínio. Esse comportamento pode parecer muito estranho, mas é assim que foi implementado, embora a especificação não deixe claro que assim deve ser.

Então, quando alteramos a área de armazenamento `localStorage` com uso dos métodos `setItem()`, `removeItem()` ou `clear()` disparamos um evento tipo `storage` em todas as outras janelas ou abas abertas no mesmo domínio.

Quando chamamos o método `setItem()` a propriedade `key` retorna uma string com o nome do dado que acaba de ser armazenado. A propriedade `newValue` retorna o valor do dado que acaba de ser armazenado. A propriedade `oldValue` retorna o valor antigo do nome que foi armazenado, se houver um valor para o nome, caso contrário retorna `null`.

Quando chamamos o método `removeItem()` a propriedade `key` retorna uma string com o nome do dado que acaba de ser removido. A propriedade `oldValue` retorna o valor do dado que acaba de ser removido. A propriedade `newValue` retorna `null`.

Quando chamamos o método `clear()` as propriedades `key`, `oldValue` e `newValue` retornam `null`.

O método `getItem()` não dispara o evento, pois esse método não altera a área de armazenamento.

O exemplo a seguir, disponível no site do livro, demonstra o funcionamento do evento `storage` para cada um dos métodos citados, inspecionando as propriedades do evento.

► JavaScript

```
<script>

function msgEvento(e) {
    if (!e) {e = window.event;}
    var limpar = document.getElementById('limpar');
    var containerMensagem = document.getElementById('mensagem');

    var mensagem = 'Disparado o evento: ' + e.type +'  
';
    mensagem += 'Nome do dado modificado: ' + e.key +'  
';
    mensagem += 'Valor antigo o dado: ' + e.oldValue +'  
';
    mensagem += 'Valor atual do dado modificado: ' + e.newValue +'  
';
    mensagem += 'Área de armazenamento: ' + e.storageArea +'  
';
    mensagem += 'URL das modificações: ' + e.url;
    containerMensagem.innerHTML = mensagem;
    limpar.onclick = function() {
        containerMensagem.innerHTML = '';
    }
}

function salvarDado(name) {
    var valor = document.forms['dadoform'].dadovalue.value;
    if (!valor)
        alert('Entre um valor para o dado a armazenar.');
    else {
        localStorage.setItem(name, valor);
    }
}

function apagarDado(name) {
    localStorage.removeItem(name);
}

function clearDados() {
    localStorage.clear();
}
```

```

if (window.addEventListener) {
    window.addEventListener('storage', msgEvento, false);
} else {
    window.attachEvent('onstorage', msgEvento);
}

```

► HTML

```

<form name="dadoform" action="#">
    <p><label>Escolha um valor para o dado "meuDado":</label><br> <input name="dadovalue"></p>
</form>
<p><a href="#" onClick="salvarDado('meuDado')">Armazenar - setItem()</a></p>
<p><a href="#" onClick="apagarDado('meuDado')">Apagar dado - removeItem()</a></p>
<p><a href="#" onClick="clearDados()">Apagar todos os dados - clear()</a></p>
<p id="mensagem"></p>
<button type="button" id="limpar">Limpar</button>

```

 [c8-evento-storage.html]

8.1.5 Banco de dados

Banco de dados é outra forma de armazenamento de dados no lado do cliente prevista originalmente nas especificações da HTML5 e posteriormente entregue à responsabilidade do W3C.

A primeira especificação para essa modalidade de armazenamento denominou-se Web SQL Storage, que foi implementada nos principais navegadores modernos, exceto no Firefox, pois a fundação Mozilla entendeu que a tecnologia não era segura e inconsistente o suficiente para torná-la imprópria a ser empregada em uma especificação. Sugeriu então ao W3C a adoção de banco de dados com base em IndexedDB.

O W3C reconheceu a argumentação da Mozilla e, em meados de 2010, abandonou a Web SQL Storage e adotou a especificação denominada Indexed Database API, que atualmente se encontra em fase inicial de desenvolvimento e suportada de forma proprietária pelo Firefox4.

A criação de uma tecnologia usando banco de dados para armazenagem no lado do cliente visou a fornecer ao desenvolvedor um mecanismo com capacidade de armazenamento maior que os 5Mb previstos para sessionStorage e localStorage.

Não iremos aprofundar o assunto, pois a especificação, como dito, ainda se encontra em fase inicial e instável, contudo mostraremos os principais fundamentos do armazenamento com uso da API IndexedDB. Os conceitos e códigos que ilustram

as transações com o banco de dados, mostrados a seguir, baseiam-se e foram adaptados do documento denominado IndexedDB primer hospedado no MDN Doc Center da fundação Mozilla.

- A API prevê requisições tanto síncronas quanto assíncronas. As síncronas são previstas para uso em conjunto com a tecnologia WebWorks e as assíncronas para aplicações Web em geral.
- O modelo do banco de dados é o *transactional* significando basicamente que o sistema de gerenciamento do banco é capaz de tratar as transações de forma atômica e independente, além de preservar a consistência do banco mesmo em caso de erros ou falhas.
- O armazenamento de dados baseia-se no princípio chave/valor. Os valores podem ser tanto simples quanto complexos objetos e a chave é uma propriedade do objeto.
- É prevista a possibilidade de requisições durante todo o processo de transação com o banco de dados. Por exemplo: estão disponíveis as propriedades `readyState`, `onsuccess`, `onerror`, `result`, `errorCode` e os métodos `addEventListener()` e `removeEventListener()`.
- Eventos do DOM podem ser usados para notificações do andamento das transações. Por exemplo: estão disponíveis os eventos `success` e `error`.
- A criação de bancos de dados e o consequente armazenamento de dados foram projetados na API para estarem disponíveis offline.

Os principais passos para uma transação completa com um banco de dados IndexedDB são mostrados a seguir.

- Iniciar uma transação.
- Fazer a requisição para realizar uma operação com o banco, por exemplo, adicionar ou requisitar um dado armazenado.
- Aguardar a finalização e resultado da requisição monitorando o apropriado evento do DOM.
- Manipular o resultado da requisição.

A sintaxe para iniciar uma transação é mostrada a seguir.

```
var request = windowIndexedDB.open('Catalogo', 'Catalogo de email e telefone');
```

O objeto `indexedDB` admite um método denominado `open()` que, por sua vez, admite dois parâmetros. Os parâmetros são: o nome do banco de dados e uma breve descrição do banco. Esse método destina-se tanto a criar um banco novo quanto a abrir um banco existente. Em nosso exemplo, o banco criado foi atribuído à variável `request` que passa a representar o banco.

Convém ressaltar que bancos de dados locais são criados e atrelados a domínios. Assim, poderemos ter dois bancos de dados com o mesmo nome e completamente independentes desde que criados para diferentes domínios, ainda que páginas dos dois domínios estejam abertas em diferentes janelas ou abas do mesmo navegador.

Uma vez que as especificações para `indexedDB` ainda estão em fase inicial, a fundação Mozilla implementou as funcionalidades para esse banco de dados usando o prefixo `moz` que adotaremos nos exemplos a seguir com a finalidade de ilustrar o funcionamento da API. Assim, nosso código para a criação de um banco de dados será escrito com a sintaxe mostrada a seguir.



Os exemplos disponíveis no site do livro deverão ser visualizados no navegador Firefox4+.

```
window.indexedDB = window.mozIndexedDB;  
var request = window.indexedDB.open('Catalogo', 'Catalogo de email e telefone');
```

Uma vez aberto ou criado o banco de dados, o próximo passo será verificar se a requisição foi bem-sucedida ou falhou, como mostrado a seguir.

► JavaScript

```
<script>  
    window.onload = function() {  
        window.indexedDB = window.mozIndexedDB;  
        var mensagem = document.getElementById('mensagem');  
        var limpar = document.getElementById('limpar');  
        var request = window.indexedDB.open('Catalogo', 'Catalogo endereços de email e telefone');  
        request.onsuccess = function(event) {  
            mensagem.innerHTML = 'Banco de dados "Catalogo" criado/aberto com sucesso.';  
            limpar.style.display = 'block';  
        }  
        request.onerror = function(event) {  
            mensagem.innerHTML = 'Falha ao criar/abrir banco de dados.';  
            limpar.style.display = 'block';  
        }  
    }</script>
```

```
limpar.onclick = function() {
    mensagem.innerHTML = '';
    limpar.style.display = 'none';
}
}

</script>
```

► HTML

```
<section>
    <p id="mensagem"></p>
    <button type="button" id="limpar" style="display:none;">Remover mensagem</button>
</section>
```



[c8-indexedDB-ex1.html]

Nesse exemplo, disponível no site do livro, você poderá visualizar a criação/abertura de um banco de dados local, bem como a chamada de funções callback para sucesso e falha. Notar que o navegador apresenta uma mensagem solicitando a permissão do usuário para criar o banco de dados.

Para melhor entender o funcionamento do script, sugiro a seguinte sequência de ações no arquivo exemplo.

Carregue o arquivo e na mensagem do Firefox solicitando permissão para armazenar dados abra o menu Permitir e escolha Nunca para este site, como mostrado na figura 8.1 a seguir.

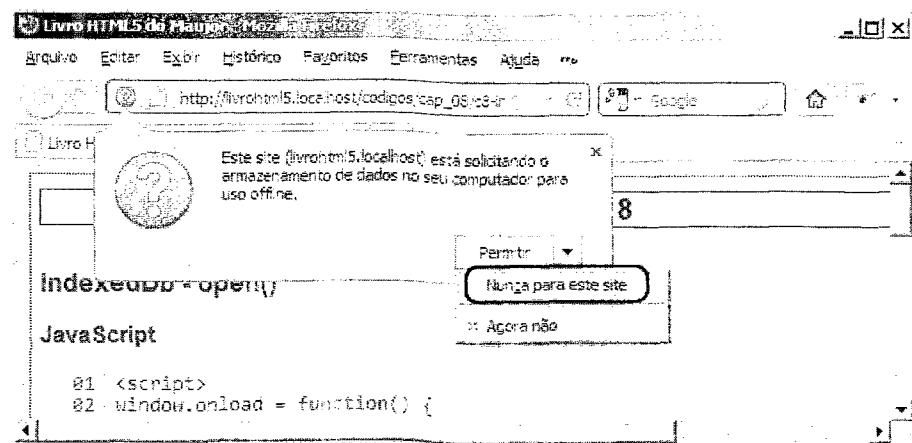


Figura 8.1 – Permissão para armazenar dados no Firefox.

Carregue o arquivo e na mensagem do Firefox solicitando permissão para armazenar dados abra o menu **Permitir** e escolha **Nunca para este site**, como mostrado na figura 8.2.

Com essa ação, você provocará um erro na criação do banco de dados e a função callback para o evento `onerror` será executada mostrando uma mensagem de erro ao usuário, conforme consta no código mostrado.

Clique o botão “Remover mensagem” e a seguir recarregue a página (`Ctrl+F5`). A mensagem de erro reaparece e você não mais conseguirá criar o banco de dados, recarregando a página ou mesmo em outra janela do navegador, pois o Firefox bloqueou qualquer tentativa de armazenamento de dados vindo do domínio que contém o arquivo. Para desbloquear o armazenamento, abra o menu do navegador **Ferramentas > Propriedades da página > Permissões**. Na janela que se abre, marque a caixa **Perguntar** em **Preservar armazenamento offline**, conforme mostrado na figura 8.2.

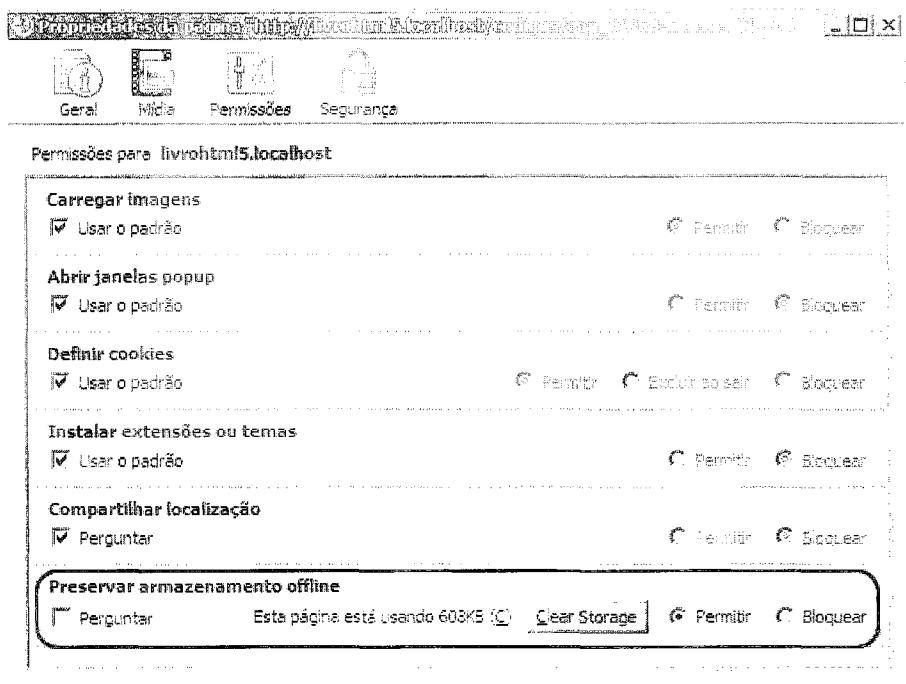


Figura 8.2 – Ajuste de permissões para armazenar dados no Firefox.

Recarregue a página e repita as operações descritas anteriormente, agora permitindo a armazenagem. Nesse caso, a função callback para o evento `onsuccess` sera executada mostrando uma mensagem de sucesso na criação do banco de dados, conforme consta no código mostrado.

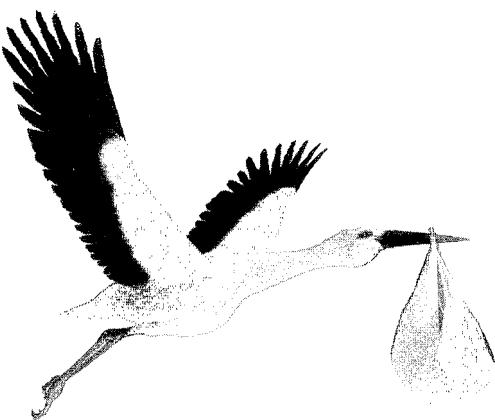
Convém notar que, se você estiver navegando no modo “Navegação privada”, não será permitida a armazenagem de dados localmente em nenhuma hipótese, mesmo que você habilite a permissão no submenu **Permissões**, como mostrado na figura 8.2. Para iniciar a navegação privada no Firefox, tecle **Ctrl+Shift+P**.

Não iremos prosseguir com esse assunto, pois as especificações do W3C para implementação da funcionalidade IndexedDB API ainda estão no início. O propósito do exemplo mostrado foi tão somente mostrar que a implementação de banco de dados local é uma realidade em andamento a ser implementada no futuro que tal como outras funcionalidades da HTML5 facilitará e muito o desenvolvimento de aplicações web ágeis e seguras. Se você estiver interessado em aprofundar seus conhecimentos em relação à implementação proprietária da Mozilla, que certamente influenciará a especificação final do W3C, consulte https://developer.mozilla.org/en/IndexedDB/IndexedDB_primer.

8.2 Verificando suporte

Nos exemplos mostrados neste capítulo não usamos a verificação de suporte nos navegadores para as funcionalidades descritas. Como vimos em capítulos anteriores, o uso da biblioteca *Modernizr* fornece uma maneira simples de se processar essa verificação. A seguir, mostramos a sintaxe para a verificação com uso da biblioteca citada.

```
if (!Modernizr.sessionstorage) {  
    // código alternativo para sessionStorage  
}  
if (!Modernizr.localstorage) {  
    // código alternativo para localStorage  
}  
if (!Modernizr.indexeddb) {  
    // código alternativo para indexedDB  
}
```



CAPÍTULO 9

APIs para comunicação

Neste capítulo, mostraremos as funcionalidades das APIs para comunicação previstas nas especificações do W3C. Denominamos APIs para comunicação aquelas previstas nas seguintes especificações: HTML5 Web Messaging, API Web Sockets e API Web Workers.

9.1 Web Messaging

Web Messaging é uma API da HTML5 destinada a definir funcionalidades que possibilitam a comunicação entre aplicações web hospedadas em diferentes origens. Com uso das técnicas existentes, esse tipo de comunicação não é possível por razões de segurança e privacidade. Trata-se da chamada “política da mesma origem”.

Basicamente, a política da mesma origem baseia-se em um conceito de segurança que não permite que programação em linguagem client-side proveniente de uma origem (por exemplo: origem B) interfira na programação, estrutura, apresentação ou qualquer outra característica do documento no qual estão hospedados e que pertencem a uma outra origem (por exemplo: origem A). Em resumo, a um script proveniente de uma origem não é permitido acessar o DOM de um documento com origem diferente daquela do script.

Se a sua página web hospeda um widget de terceiro, por exemplo, um bloco de conteúdo do Adsense, ao script que insere o bloco é vedado acesso a qualquer parte da página, exceto, é claro, a própria inserção do widget.

Um documento pode ser proveniente de três tipos distintos de origem: *PROTOCOL*, *HOST* e *PORT*.

A tabela a seguir, extraída do MDN Doc Center da fundação Mozilla e devidamente adaptada, esclarece a política da mesma origem. Suponha que a origem seja o URL <http://maujor.com/tutorial/widget-css.php>

URL	Mesma origem?	Razão
http://maujor.com/dicas/centrar.php	Sim	-
http://maujor.com/	Sim	-
https://maujor.com/area-restrita.php	Não	protocolo diferente
http://maujor.com:81/w3c/css2.php	Não	porta diferente
http://fw.maujor.com/bezier.php	Não	host diferente

A API, ao mesmo tempo que oferece a possibilidade de transpor as restrições impostas pela política da mesma origem, também fornece mecanismos destinados a bloquear acessos maliciosos provenientes de diferentes origens, ou seja, os “ataques cross-site”.

window.postMessage(mensagem, destino, [portas])

O método `window.postMessage()` destina-se a criar uma mensagem a ser enviada para um objeto `window` em uma origem diferente daquela que envia mensagem. Por padrão, o tipo de mensagem enviada é uma `string`. Contudo, podemos enviar um objeto com uso dos métodos `JSON.stringify()` e `JSON.parse()` tal como mostramos em 8.1.3 para o armazenamento local.

Esse método admite três parâmetros: dois obrigatórios e um opcional. O parâmetro `mensagem` é uma `string` contendo a mensagem a ser passada, o parâmetro `destino` define o endereço para o qual a mensagem está sendo enviada e o parâmetro `portas`, que é opcional, define um `array` de portas válidas para o destino da mensagem.

O parâmetro `destino` pode ser um URL absoluto definindo um único endereço de destino da mensagem, um caractere curinga (*) definindo qualquer destino ou um caractere barra (/) que restringe o destino da mensagem à sua origem, isto é, adota a política da mesma origem.

Para recebimento da mensagem, a API prevê o evento `message` a ser disparado no documento destino da mensagem. Esse evento, quando disparado, chama uma função callback (manipuladora do evento), conforme sintaxe geral mostrada a seguir.

```
<script>
  if (window.addEventListener) {
    window.addEventListener('message', receberMensagem, false);
  } else {
    window.attachEvent("onmessage", receberMensagem);
```

```

};

function receberMensagem(e) {
    // faz alguma coisa com a mensagem recebida
}
</script>

```

message

Conforme mostrado, o documento destino monitora o evento `message` que, ao ser disparado, chama a função `recepberMensagem(e)` destinada a processar a mensagem recebida.

Essa função recebe um parâmetro que retorna um objeto-evento com as seguintes propriedades:

- `e.data` – retorna o texto da mensagem.
- `e.origin` – retorna a origem da mensagem.
- `e.lastEventId` – retorna a string identificadora do último evento.
- `e.source` – retorna `WindowProxy` do destino da mensagem.
- `e.ports` – retorna o array das portas enviadas com a mensagem.

O exemplo a seguir, disponível no site do livro, demonstra o funcionamento da API.

Suponha que um documento A, hospedado em `http://livrohtml5.com.br`, contém um `iframe` que, por sua vez, contém um documento B, hospedado em `http://maujor.com`. O envio de uma mensagem de A para B se faz com um script usando o método `window.postMessage()` inserido em A, definindo a mensagem e o destino. Para que o documento B receba a mensagem de A, deverá conter um script que registre um manipulador do evento denominado `message` definido na API.

Os scripts e marcações nos dois documentos são mostrados a seguir.

No documento A:

► JavaScript

```

<script>
window.onload = function() {
    var objetoIframe = document.getElementsByTagName('iframe')[0];
    var btnEnviar = document.getElementsByTagName('button')[0];

```

```
btnEnviar.onclick = function() {
    var textoMensagem = document.getElementsByTagName('input')[0].value;
    if (textoMensagem == '') {
        alert('Digite a mensagem a ser enviada');
    } else {
        objetoIframe.contentWindow.postMessage(textoMensagem, 'http://maujor.com');
    }
}
</script>
```

► HTML

```
<section>
    <p><label>Mensagem: <input type=text></label>
    <button type=button>Enviar mensagem</button></p>
    <iframe src="http://maujor.com/messaging.html"></iframe>
</section>
```

No documento B:

► JavaScript

```
<script>
    if (window.addEventListener) {
        window.addEventListener('message', receberMensagem, false);
    } else {
        window.attachEvent("onmessage", receberMensagem);
    };
    function receberMensagem(e) {
        var mensagem;
        var containerMensagem = document.getElementById('recebe-mensagem');
        if (e.origin == 'http://livrohtml5.com.br') {
            mensagem = 'Recebida a seguinte mensagem: <br>';
            mensagem += 'Texto: ' + e.data + '<br>';
            mensagem += 'Origem: ' + e.origin;
            containerMensagem.innerHTML = mensagem;
        } else {
            containerMensagem.innerHTML = 'Tentativa de envio de mensagem de uma origem não autorizada';
        }
    }
</script>
```

► HTML

```
<p id="recebe-mensagem "></p>
```



[c9-messaging.html]

O envio de uma mensagem com uso de Web Messaging, como mostrado no exemplo, faz-se com a transformação da mensagem em string, por padrão. Se você pretende enviar dados a serem manipulados individualmente quando chegarem ao destino, esse comportamento de transformação em string inviabilizará seu projeto, pois enviando os dados encadeados em uma única string, não conseguirá separá-los e nem mesmo distingui-los no destino.

Felizmente a tecnologia JSON oferece a funcionalidade de transformar um objeto em string com uso do método `JSON.stringify()` e posteriormente recuperar o objeto com uso do método `JSON.parse()`. Usando esses dois métodos, viabilizamos o envio de dados de qualquer natureza com uso de Web Messaging.

O exemplo a seguir, disponível no site do livro, permite que você visualize o envio de dados com uso de JSON.

No documento que envia a mensagem:

► JavaScript

```
<script>
    window.onload = function() {
        var objetoIframe,btnEnviar,nomeUsuario,horaMensagem,TextoMensagem,dadosEnviar,stringEnviar;
        objetoIframe = document.getElementsByTagName('iframe')[0];
        btnEnviar = document.getElementsByTagName('button')[0];
        btnEnviar.onclick = function() {
            now = new Date();
            hour = now.getHours();
            minute = now.getMinutes();
            second = now.getSeconds();

            horaMensagem = hour + ':' + minute + ':' + second + 'h';
            nomeUsuario = document.getElementsByTagName('input')[0].value;
            textoMensagem = document.getElementsByTagName('textarea')[0].value;
            if (textoMensagem == '' || nomeUsuario == '') {
                alert('Os campos nome e mensagem devem ser preenchidos');
            } else {
                dadosEnviar = {
                    nome: nomeUsuario,
                    texto: textoMensagem,
                    hora: horaMensagem
                }
                stringEnviar = JSON.stringify(dadosEnviar);
            }
        }
    }
</script>
```

```
        objetoIframe.contentWindow.postMessage(stringEnviar, 'http://livrohtml5.com.br');
```

```
    }
```

```
}
```

```
}
```

```
</script>
```

► HTML

```
<section>
```

```
    <p><label>Seu nome:<br><input type=text></label></p>
```

```
    <p><label>Mensagem:<br><textarea></textarea></label></p>
```

```
    <p><button type=button>Enviar mensagem</button></p>
```

```
    <iframe src="json.html"></iframe>
```

```
</section>
```

No documento que recebe a mensagem (inserido em um iframe):

► JavaScript

```
<script>
```

```
if (window.addEventListener) {
```

```
    window.addEventListener('message', receberMensagem, false);
```

```
} else {
```

```
    window.attachEvent("onmessage", receberMensagem);
```

```
};
```

```
function receberMensagem(e) {
```

```
    var containerMensagem, msgRecebida, nomeRecebido, textoRecebido, para, span, conteudoSpan, conteudoPara;
```

```
    containerMensagem = document.getElementById('recebe-mensagem');
```

```
    if (e.origin == 'http://livrohtml5.com.br') {
```

```
        msgRecebida = JSON.parse(e.data);
```

```
        nomeRecebido = msgRecebida.nome;
```

```
        textoRecebido = msgRecebida.texto;
```

```
        horaRecebida = msgRecebida.hora;
```

```
        para = document.createElement('p');
```

```
        span = document.createElement('span');
```



```
        conteudoSpan = document.createTextNode(nomeRecebido);
```

```
        conteudoPara = document.createTextNode(' ('+horaRecebida+') '+ textoRecebido);
```



```
        span.appendChild(conteudoSpan);
```



```
        para.appendChild(span);
```

```
        para.appendChild(conteudoPara);
```

```
        containerMensagem.appendChild(para);
```

```
    } else {
```

```

    containerMensagem.innerHTML = 'Tentativa de envio de mensagem de uma origem não autorizada';
}
</script>

```

▷ **HTML**

```

<section>
  <div id="recebe-mensagem"></div>
</section>

```



[c9-json.html]

Notar que enviamos três dados: nome, mensagem e hora. A hora poderia ter sido aferida no documento de destino; contudo, optamos pela tomada da hora na origem apenas para ilustrar o envio de mais um dado.

9.2 Web Socket

Web Socket é uma API originariamente incluída na especificação para a HTML5 com o nome de TCP Connection e posteriormente desvinculada da HTML5 e entregue para desenvolvimento pelo W3C. Destina-se a definir um protocolo e suas funcionalidades que possibilitem a comunicação de um cliente web com um servidor remoto. A API criou os protocolos Transmission Control Protocol (TCP) para *socket*: denominados *ws:* e *wss:* destinados a estabelecer comunicação com o servidor com uso de conexão normal e segura, a exemplo dos protocolos *http:* e *https:*.



Em linguagem de computadores *socket* é um ponto abstrato localizado nas terminações de um caminho de comunicação bidirecional estabelecido por um protocolo de transmissão.

Uma vez criada a conexão, ela permanece aberta possibilitando comunicação permanente entre o cliente e o servidor. A conexão somente será fechada por determinação explícita do desenvolvedor. Para criar um *socket* de comunicação, a sintaxe geral é mostrada a seguir.

```
var wsConn = new WebSocket('ws://jogo.exemplo.com:12010/atualizacoes');
```

Uma conexão com *socket* é *full-duplex*, ou seja, permite comunicação simultânea nas duas direções: cliente servidor e servidor cliente. Em uma conexão convencional, também conhecida como *half-duplex*, com uso do protocolo *http*, por exemplo, cada solicitação do cliente ao servidor e respectiva resposta é independente e implica o envio e tráfego de cabeçalhos *HTTP*. Para uma aplicação que necessite consultar o servidor repetidamente para verificar atualizações em tempo real, o tráfego envolvido

na comunicação pode requerer um alto consumo de banda com respectivo aumento da taxa de transferência. É o caso, por exemplo, de transmissões de eventos ao vivo, de jogos multiplayers e de salas de bate-papo. A API Web Socket reduz significativamente esse consumo de banda e consequente taxa de transferência.

Outra vantagem da API é a redução drástica da latência na comunicação. Latência em comunicação bidirecional é o tempo de espera pela resposta.

O gráfico mostrado na figura 9.1 ilustra os esquemas de comunicação convencional (*half-duplex*) e com socket (*full-duplex*).

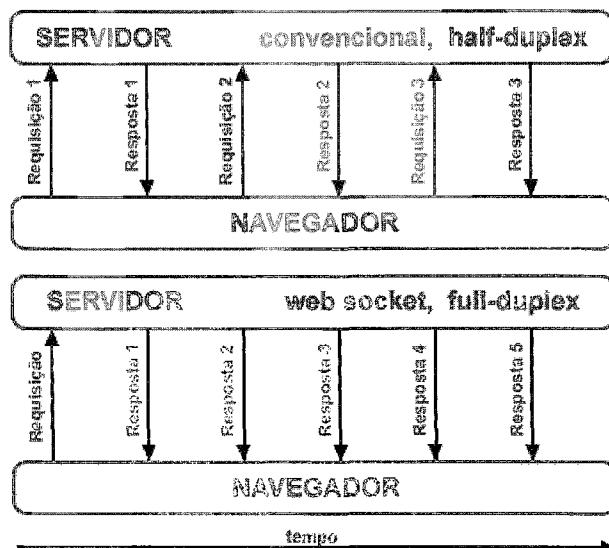


Figura 9.1 – Comunicação *half-duplex* x *full-duplex*.

Estudos e testes conduzidos pela Corporação Kaazing, que está envolvida com o desenvolvimento da tecnologia de comunicação com *socket*, dão conta que: “HTML5 Web Sockets reduz o tráfego de cabeçalhos HTTP em 500:1. Dependendo do tamanho dos cabeçalhos HTTP a redução pode chegar a 1000:1. A redução da latência é de 3:1”.

Em resumo: Web Sockets pode tornar uma aplicação mais rápida, mais eficiente e mais escalável.

O exemplo a seguir, adaptado do exemplo fornecido pela especificação do W3C para Web Socket, ilustra a sintaxe e dá uma ideia do funcionamento de uma requisição com uso dessa tecnologia. O código mostra o envio, para o servidor, de atualizações de um jogo em andamento no navegador do usuário.

```

var wsConn = new WebSocket('ws://jogo.exemplo.com:12010/atualizacoes');
wsConn.onopen = function () {
    setInterval(function() {
        if (wsConn.bufferedAmount == 0)
            wsConn.send(getUpdateData());
    }, 50);
};

```

O exemplo inicia com a abertura de uma linha de comunicação com um arquivo de atualizações de um jogo. Aberta a comunicação, uma função callback é chamada a cada 50ms. A função verifica a quantidade de informação sobre a atualização armazenada em buffer e, como resultado da verificação, envia os dados de atualização coletados pela função `getUpdateData()`.

Convém notar que a tecnologia se encontra ainda em fase de desenvolvimento e seu pleno uso nos dias atuais requer bibliotecas para configurar o suporte ao servidor. A biblioteca mais popular é a `socket.io` que fornece funcionalidades de implementação da tecnologia tanto no lado do cliente quanto no servidor, para Java e Node.js. Não é do escopo deste livro detalhar tais implementações.

Mostraremos a seguir a título de informação as funcionalidades disponíveis na API para Web Sockets.

9.2.1 Propriedades e métodos

`WebSocket(url [,protocolos])`

Trata-se de um construtor que admite um ou dois argumentos. O primeiro argumento, `url`, define o URL com o qual será estabelecida a comunicação. O segundo argumento, `protocolos`, é facultativo e quando presente define um nome ou um array de nomes de subprotocolos válidos para estabelecimento da comunicação.

`send(dado)`

Esse método destina-se a enviar dados para o servidor. Os dados a enviar são, por padrão, em formato de string e tal como ocorre com Web Messaging o uso dos métodos `JSON.stringify()` e `JSON.parse()` viabiliza o envio de objetos.

A sintaxe para uso desse método é mostrada a seguir.

```

objetoDados = {
    nome: 'Maujor',
    mensagem: 'Bom dia!'
};

```

```
stringDados = JSON.stringify(objetoDados);
send(stringDados);
...
```

Para que os dados sejam enviados, é necessário que o atributo `readyState` tenha assumido o valor `OPEN` e ainda não tenha entrado no valor `CLOSING`. Além disso, é necessário que o buffer não esteja cheio.

readyState

Trata-se de uma propriedade destinada a monitorar o estado da comunicação podendo assumir os seguintes valores:

- `CONNECTING` (0) – a conexão não foi estabelecida.
- `OPEN` (1) – a conexão foi estabelecida e a comunicação é possível.
- `CLOSING` (2) – a conexão está em processo de fechamento.
- `CLOSED` (3) – a conexão foi fechada e não poderá ser aberta.

bufferedAmout

Esse atributo retorna o número de bytes contido no texto UTF-8 a ser enviado como dado. Evocar o método `send()` com o buffer cheio fecha a conexão aberta, por isso é essencial que se verifique a situação do buffer antes de enviar um dado.

A sintaxe para uso desse atributo é mostrada a seguir.

```
objetoDados = {
    nome: 'Maujor',
    mensagem: 'Bom dia!'
};

stringDados = JSON.stringify(objetoDados);
if (wsConn.bufferedAmount == 0) {
    send(stringDados);
};
...
```

9.2.2 Eventos

onopen

Esse atributo, que define o evento `open`, está disponível para todos os objetos implementados pela interface `WebSocket`. O evento é disparado quando a comunicação é estabelecida.

A sintaxe para uso desse atributo é mostrada a seguir.

```
var wsConn = new WebSocket('ws://exemplo.com:12345/ex');
wsConn.onopen = function (e) {
    objetoDados = {
        nome: 'Maujor',
        mensagem: 'Bom dia!'
    };
    stringDados = JSON.stringify(objetoDados);
    if (wsConn.bufferedAmount == 0) {
        send(stringDados);
    };
    ...
}
```

onmessage

Esse atributo, que define o evento `message`, está disponível para todos os objetos implementados pela interface `WebSocket`. O evento é disparado quando os dados enviados chegam ao destino.

A sintaxe para uso desse atributo é mostrada a seguir.

```
var wsConn = new WebSocket('ws://exemplo.com:12345/ex');
wsConn.onmessage = function (e) {
    objetoDados = {
        nome: 'Maujor',
        mensagem: 'Bom dia!'
    };
    stringDados = JSON.stringify(objetoDados);
    if (wsConn.bufferedAmount == 0) {
        send(stringDados);
    };
    ...
wsConn.onmessage = function (e) {
    dadosRecebidos = JSON.parse(e.data);
    alert('Nome: ' + dadosRecebidos.nome + 'Mensagem: ' + dadosRecebidos.mensagem);
```

onclose

Esse atributo, que define o evento `close`, está disponível para todos os objetos implementados pela interface `WebSocket`. O evento é disparado quando a conexão é encerrada.

A sintaxe para uso desse atributo é mostrada a seguir.

```
var wsConn = new WebSocket('ws://exemplo.com:12345/ex');
wsConn.onmessage = function (e) {
    objetoDados = {
        nome: 'Maujor',
        mensagem: 'Bom dia!'
    };
    stringDados = JSON.stringify(objetoDados);
    if (wsConn.bufferedAmount == 0) {
        send(stringDados);
    };
    ...
    wsConn.onmessage = function (e) {
        dadosRecebidos = JSON.parse(e.data);
        alert('Nome: ' + dadosRecebidos.nome + ' Mensagem: ' + dadosRecebidos.mensagem);
    ...
    wsConn.onclose = function (e) {
        alert('A conexão foi encerrada');
    };
}
```

onerror

Esse atributo, que define o evento `error`, está disponível para todos os objetos implementados pela interface `WebSocket`. O evento é disparado quando ocorre um erro.

A sintaxe para uso desse atributo é mostrada a seguir.

```
var wsConn = new WebSocket('ws://exemplo.com:12345/ex');
wsConn.onerror = function(error) {
    alert('Ops! ocorreu o erro: ' + erro);
};
```

9.3 Web Workers

Web Workers é uma API não incluída na especificação para a HTML5 e desenvolvida pelo W3C. Trata-se de mais uma tecnologia que começou a ser implementada pelos navegadores com as funcionalidades da HTML5 e ficou a ela vinculada embora dela não fazendo parte.

Uma das características da linguagem JavaScript é a sua incapacidade de executar dois ou mais scripts simultaneamente. Trata-se de uma linguagem que no jargão técnico é chamada de *single-thread* que, em tradução livre, significa “uma só linha”, significando que sua execução segue por um caminho reto que comporta um e somente um script. Não há lugar para mais de um script em execução.

Uma consequência imediata dessa característica da linguagem e uma experiência pela qual provavelmente você já tenha passado é que nenhuma interação com uma página web é possível enquanto um script longo está sendo executado, pois a página simplesmente “congela”.

Scripts que demandam muito tempo para execução são detectados pelos mecanismos internos dos navegadores e, depois de passado determinado tempo, acabam por fazer com que uma caixa de diálogo seja apresentada ao usuário informando a demora e perguntando se ele quer que interrompa ou continue a execução do script.

A figura 9.2 mostra a caixa de diálogo, para interrupção de script longo, no navegador Firefox4.

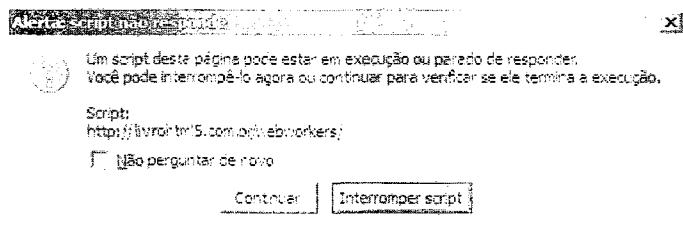


Figura 9.2 – Caixa de diálogo para script longo.

Algumas técnicas têm sido usadas para contornar essa restrição, como uso de métodos `setInterval()`, `setTimeout()` e do objeto `XMLHttpRequest`, contudo são soluções que nem sempre simulam a execução de múltiplos scripts simultaneamente.

Web Workers é uma tecnologia que se destina a minimizar os problemas criados pela característica *single-thread* da linguagem JavaScript, criando funcionalidades capazes de permitir a execução de mais de um script simultaneamente em uma aplicação. Em outras palavras, é possível criar múltiplos threads em uma aplicação e a linguagem assume características que a coloca lado a lado com as modernas linguagens *multi-thread*.

Observe na figura 9.3 a seguir um diagrama mostrando a diferença entre execução de tarefas com JavaScript tradicional e com Web Workers.

Notar que com Web Workers é possível realizar as tarefas B e D ao mesmo tempo que são realizadas as tarefas A e C. O cálculo de A e B começa ao mesmo tempo e, ao terminar, a tarefa C que depende de A e B é iniciada com a tarefa D. Terminadas estas, o cálculo final é processado.

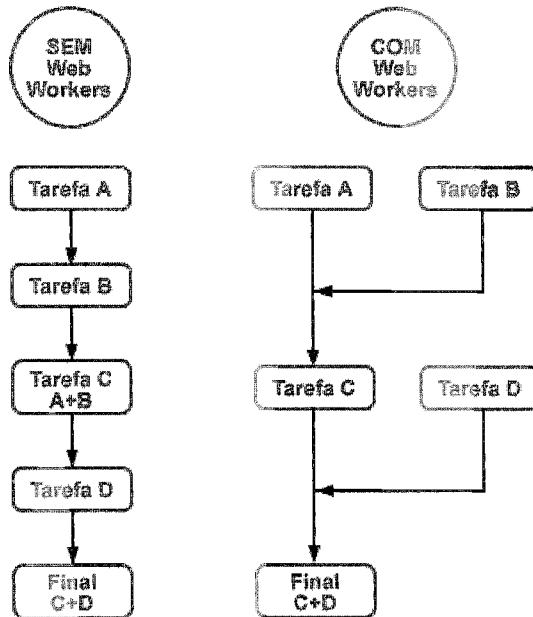


Figura 9.3 – Diagrama comparativo JavaScript x Web Workers.

Por outro lado, com JavaScript tradicional, todas as tarefas são realizadas em um processo semelhante a um passo a passo com cada uma das tarefas iniciando somente quando a anterior for concluída.

Porém, Web Workers tem acesso apenas aos seguintes conjuntos de funcionalidades da JavaScript.

- Objeto `navigator`.
- Objeto `location` (somente leitura).
- Objeto `XMLHttpRequest`.
- Objetos JavaScript – `Object`, `Array`, `Date`, `Math`, `String` etc.
- Método `setInterval()`.
- Método `clearInterval()`.
- Método `setTimeout()`.
- Método `clearTimeout()`.
- Interface `application Cache`.
- Importação de scripts com uso do método `importScripts()`.
- Criação de sub threads.

Web Workers não tem acesso ao DOM nem aos objetos `window`, `document` e `parent`. Por exemplo, não é possível o uso dos métodos `alert()` e `getElementById()` quando se usa Web Workers.

Segundo o W3C, a finalidade da especificação para Web Workers é definir uma API que possibilite aos autores de aplicações web criar mecanismos (chamados workers) capazes de executar scripts em background e em paralelo com a página web principal.

A execução de workers em uma aplicação web não interfere com os scripts da GUI (Graphical User Interface ou interface gráfica do usuário); portanto, não impede a interação do usuário e muito menos “congela” uma página web.

Para demonstrar esse comportamento, criamos um exemplo, disponível no site do livro. Nele um script destinado a calcular a soma dos números naturais de 0 até 100.000.000 é disparado após o usuário clicar em um botão. O script demora alguns segundos para ser executado e durante esse tempo é solicitado ao usuário iniciar uma ação qualquer de interação com a página web, por exemplo: selecionar textos ou mesmo selecionar o endereço na barra de endereços do navegador. O usuário constatará que a página permanece congelada enquanto o script é executado e as interações não são possíveis.

A seguir, ao usuário é solicitada a mesma ação inicial, clicando em botão para acionar o script com uso de Web Workers. Dessa vez será constatado que a página não congela e a interação com a página é possível. A seguir, o exemplo, mas, em um primeiro momento, não se preocupe em entender as funcionalidades da API usadas no exemplo. Na sequência, elas serão explicadas.

► Javascript na página

```
<script>
//<![CDATA[
function init() {
    if (!Modernizr.webworkers) {alert('Lamento, seu navegador não suporta Web Workers');}
    var i,r,btnSemWorker,btnComWorker,campoResultado,btnLimpar;
    btnSemWorker = document.getElementsByTagName('button')[0];
    btnComWorker = document.getElementsByTagName('button')[1];
    campoResultado = document.getElementsByTagName('input')[0];
    btnLimpar = document.getElementsByTagName('button')[2];
    campoResultado.style.color = 'red';
    // sem worker
    function calcula() {
        campoResultado.value = 'Calculando...';
        setTimeout(function() {

```

```
r = 0
for (i=0; i<=1e+8; i++) {r += i;}
campoResultado.value = r;
},100)
};

if (window.addEventListener) {
    btnSemWorker.addEventListener('click', calcula, false);
    btnLimpar.addEventListener('click', function() {
        campoResultado.value = '';
    }, false);
} else {
    btnSemWorker.attachEvent('onclick', calcula);
    btnLimpar.attachEvent('onclick', function() {
        campoResultado.value = '';
    });
}

// com worker
function calcWorker() {
    campoResultado.value = 'Calculando...';
    var worker = new Worker('calcula.js');
    worker.onmessage = function (evt) {
        campoResultado.value = evt.data;
    };
};

if (window.addEventListener) {
    btnComWorker.addEventListener('click', calcWorker, false);
} else {
    btnComWorker.attachEvent('onclick', calcWorker);
};

}

window.onload = init;
// ]]>
</script>
```

► Javascript no arquivo do Worker

```
var r=0;
for (i=0; i<=1e+8; i++) {r += i;}
postMessage(r);
```

► HTML

```
<section>
<h3>Cálculo da soma dos números naturais de 0 a 100.000.000</h3>
<p>Iniciar cálculo: <button type=button>Sem Worker</button> <button type=button>Com Worker
</button></p>
```

```
<p><label>Resultado:<br><input type=text></label></p>
<button type=button>Limpar</button>
</section>
```



[c9-webworkers-ex1.html]

Em uma primeira análise, somos levados a acreditar que podemos criar workers indiscriminadamente para executar várias tarefas JavaScript pesadas. Contudo, ao criar workers, o desenvolvedor deve considerar que nem sempre é apropriado um elevado número de workers na página. A especificação do W3C ilustra esse cuidado citando que não é apropriada a criação de um worker para cada pixel de uma imagem com tamanho de quatro megapixel. Afirma ainda que, em geral, workers a princípio devem ter uma vida longa, além de um desempenho e alocação de memória que justifiquem seu uso.

Em linhas gerais, exemplos da conveniência de uso de workers incluem: processamento de objetos JSON extensos, manipulação de banco de dados hospedados no lado do cliente, processamento de tarefas relativas a imagens, áudio e vídeo, cálculos matemáticos dependentes de longas interações.

Vejamos a seguir a sintaxe geral para a criação e uso de workers, mostrando alguns exemplos e estudando métodos e propriedades dessa API.

new Worker('arquivo.js')

Essa é a sintaxe para criar um worker. Usa o operador `new` para criar um objeto que na API foi chamado de `Worker`. O parâmetro único aponta para um arquivo JavaScript contendo o script para a realização da tarefa. Esse objeto armazena todas as funcionalidades da API. A sintaxe JavaScript permite que você escolha livremente o nome de uma variável para armazenar um objeto recém-criado. Observe os códigos a seguir.

```
var worker = new Worker('tarefa-um.js');
var xpto = new Worker('blablabla.js');
```

Na primeira linha criamos um `worker` para executar a tarefa codificada no arquivo `tarefa-um.js` e na segunda outro `worker` denominado `xpto` para executar a tarefa codificada no arquivo `blablabla.js`.

Notar que as tarefas a serem executadas em cada um dos workers de uma página devem ser definidas em arquivos externos.

postMessage('mensagem')

Esse método destina-se a estabelecer comunicação entre o script da página principal e o script no arquivo que contém a tarefa para o worker, ou, dito de maneira simplificada, a comunicação entre a página e o worker e vice-versa. O parâmetro único desse método, mensagem, pode ser uma string ou um objeto JSON a ser passado da página para o worker e vice-versa.

message

Esse evento destina-se a monitorar a chegada de uma mensagem tanto no Worker quanto na página. A sintaxe geral de uso desse evento é mostrada a seguir.

► Javascript para receber mensagem no Worker

```
onmessage = function(e) {  
    var mensagemRecebida = e.data;      // armazena a mensagem recebida  
    ...  
}
```

ou se você preferir a codificação W3C

```
self.addEventListener('message', function(e) {  
    var mensagemRecebida = e.data;      // armazena a mensagem recebida  
    ...  
}, false)
```

Nessa sintaxe usamos a palavra-chave `self` que se refere ao Worker. Alternativamente podemos usar a palavra-chave `this` que também se refere a Worker, como mostrado a seguir.

```
this.addEventListener('message', function(e) {  
    var mensagemRecebida = e.data;      // armazena a mensagem recebida  
    ...  
}, false)
```

O exemplo a seguir, disponível no site do livro, demonstra o uso do método `postMessage()` e do evento `message` para a troca de mensagens entre a página e o Worker. Nele o usuário digita uma breve mensagem, em uma caixa de texto, para ser enviada a um Worker. Este recebe a mensagem, processa e retorna a mensagem processada para a página. O processamento do texto da mensagem recebida pelo Worker consiste simplesmente em compor uma mensagem de retorno para a página na qual consta a mensagem recebida.

O exemplo não apresenta valor prático algum e tem por finalidade, apenas, demonstrar a mecânica e sintaxe para a troca de mensagens entre a página e o Worker.

► Javascript na página

```
<script>
    function init() {
        if (!Modernizr.webworkers) {alert('Lamento, seu navegador não suporta Web Workers');}
        var campoMensagem,btnLimpar,btnIniciarWorker,campoResultado;
        campoMensagem = document.getElementsByTagName('input')[0];
        btnLimpar = document.getElementsByTagName('button')[0];
        btnIniciarWorker = document.getElementsByTagName('button')[1];
        campoResultado = document.getElementById('msg');
        function comunicarWorker() {
            if (campoMensagem.value == '') {
                alert('Digite sua mensagem para o Worker');
                return;
            }
            campoResultado.style.display = 'block';
            var textoMensagem = campoMensagem.value;
            var worker = new Worker('exemplo2.js');
            worker.postMessage(textoMensagem); // envia mensagem para o Worker
            worker.onmessage = function (e) { // recebe mensagem do Worker
                campoResultado.innerHTML = e.data;
            };
        };
        if (window.addEventListener) {
            btnIniciarWorker.addEventListener('click', comunicarWorker, false);
            btnLimpar.addEventListener('click', function() {
                campoMensagem.value = '';
                campoResultado.style.display = 'none';
            }, false);
        } else {
            btnLimpar.attachEvent('onclick', function() {
                campoMensagem.value = '';
                campoResultado.style.display = 'none';
            });
            btnIniciarWorker.attachEvent('onclick', comunicarWorker);
        };
    }
    window.onload = init;
</script>
```

▷ Javascript no arquivo do Worker

```
onmessage = function(e) {  
    var mensagemRecebida = e.data; // recebe mensagem da página  
    var mensagemRetornada = 'O Worker acusa o recebimento da seguinte mensagem:  
    <br><span class="dest">' + mensagemRecebida + '</span>';  
    postMessage(mensagemRetornada); // envia mensagem para a página  
}
```

▷ HTML

```
<section>  
    <h3>Comunicação da página com o Worker e vice-versa</h3>  
    <p><label>Entre uma mensagem:<br><input type=text></label></p>  
    <button type=button>Limpar</button></p>  
    <p><button type=button>Iniciar Worker</button>  
    <p id=msg></p>  
</section>
```

 [c9-webworkers-ex2.html]

Error

Esse evento destina-se a monitorar a existência de erros no Worker. As propriedades desse evento são: `lineno`, `message` e `filename` que retornam respectivamente a linha onde ocorreu o erro, a mensagem de erro e o arquivo onde se encontra o erro.

O exemplo a seguir é uma extensão do exemplo anterior no qual foi adicionado o monitoramento de erros no arquivo do worker. Nesse arquivo foi cometido um erro proposital marcado em destaque no código a seguir.

▷ Javascript na página

```
<script>  
    function init() {  
        if (!Modernizr.webworkers) {alert('Lamento, seu navegador não suporta Web Workers');}  
        var campoMensagem,btnLimpar,btnIniciarWorker,campoResultado;  
        campoMensagem = document.getElementsByTagName('input')[0];  
        btnLimpar = document.getElementsByTagName('button')[0];  
        btnIniciarWorker = document.getElementsByTagName('button')[1];  
        campoResultado = document.getElementById('msg');  
        function comunicarWorker() {  
            if (campoMensagem.value == '') {  
                alert('Digite sua mensagem para o Worker');  
                return;  
            }  
        }
```

```

campoResultado.style.display = 'block';
var textoMensagem = campoMensagem.value;
var worker = new Worker('exemplo3.js');
worker.onerror = function (e) {
    linhaErro = e.lineno;
    msgErro = e.message;
    arquivoErro = e.filename;
    erro = 'Ops! Ocorreu o seguinte erro:<br>';
    erro += 'Linha: ' + linhaErro + '<br>';
    erro += 'Erro: ' + msgErro + '<br>';
    erro += 'Arquivo: ' + arquivoErro;
    campoResultado.innerHTML = erro;
};
worker.postMessage(textoMensagem); // envia mensagem para o Worker
worker.onmessage = function (e) { // recebe mensagem do Worker
    campoResultado.innerHTML = e.data;
};
};

if (window.addEventListener) {
    btnIniciarWorker.addEventListener('click', comunicarWorker, false);
    btnLimpar.addEventListener('click', function() {
        campoMensagem.value = '';
        campoResultado.style.display = 'none';
    }, false);
} else {
    btnLimpar.attachEvent('onclick', function() {
        campoMensagem.value = '';
        campoResultado.style.display = 'none';
    });
    btnIniciarWorker.attachEvent('onclick', comunicarWorker);
};

}

window.onload = init;
</script>

```

► Javascript no arquivo do Worker

```

onmessage = function(e) {
    var mensagemRecebida = e.data; // recebe mensagem da página
    var mensagemRetornada = 'O Worker acusa o recebimento da seguinte mensagem:
<br><span class="dest">' + mensagemRecebida + '</span>';
    postMessage(MensagemRetornada); // Aqui há um erro - mensagemRetornada e não MensagemRetornada
}

```

► HTML

```
<section>
<h3>Comunicação da página com o Worker e vice-versa <br>
<small>Mostra o uso do evento error. O script do worker contém erro</small></h3>
<p><label>Entre uma mensagem:<br><input type=text></label>
<button type=button>Limpar</button></p>
<p><button type=button>Iniciar Worker</button>
<p id="msg"></p>
</section>
```



[c9-webworkers-ex3.html]

Opcionalmente podemos monitorar erros dentro do próprio arquivo do worker e não na página como fizemos no exemplo mostrado anteriormente. Observe o código a seguir que demonstra essa opção.

```
onerror = function (e) {
    linhaErro = e.lineno;
    msgErro = e.message;
    arquivoErro = e.filename;
    erro = 'Ops! Ocorreu o seguinte erro:<br>';
    erro += 'Linha: ' + linhaErro + '<br>';
    erro += 'Erro: ' + msgErro + '<br>';
    erro += 'Arquivo: ' + arquivoErro;
    postMessage(erro); // enviamos a mensagem de erro para a página. Lá o script fará sua
                      // apresentação ao usuário.
};
```

importScripts(arquivo1[,arquivo2,arquivo3,...,arquivon])

Esse método destina-se a importar arquivos JavaScript para dentro de um worker. Admite um arquivo como parâmetro obrigatório, seguido de um número ilimitado de arquivos, isto é, podemos importar para o worker tantos arquivos externos quanto forem convenientes. É válida a importação de qualquer tipo de arquivo JavaScript, inclusive bibliotecas inteiras, por exemplo, a biblioteca jQuery.

Observe a seguir a sintaxe para uso desse método.

```
var worker = new Worker('tarefa-um.js'); // criamos um worker usando o arquivo tarefa-um.js
```

Suponha que para realizar seus cálculos o worker faça uso da biblioteca jQuery e de um script que o desenvolvedor resolveu separar em dois blocos independentes para a facilidade de manutenção. O arquivo *tarefa-um.js* é escrito conforme mostrado a seguir.

```
importScript('../scripts/jquery-1.6.js');
importScript('../scripts/w11.js');
importScript('../scripts/w12.js');
```

Alternativamente poderíamos escrever.

```
importScript('../scripts/jquery-1.6.js', '../scripts/w11.js', '../scripts/w12.js');
```

terminate()

Esse método destina-se a encerrar um worker. É usado no script da página.

Observe a seguir a sintaxe para uso desse método.

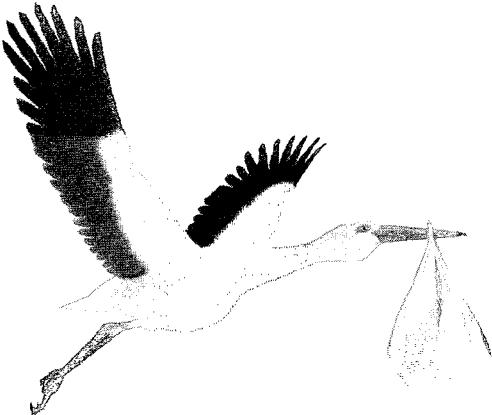
```
var meuWorker = new Worker('tarefa-um.js'); // criamos um worker usando o arquivo tarefa-um.js
...
meuWorker.terminate(); // encerramos o worker
```

close()

Esse método é semelhante ao método `terminate()` e destina-se a encerrar um worker. A diferença é que deverá ser usado no script do worker.

Observe a seguir a sintaxe para uso desse método.

```
self.close(); // A palavra chave self (ou, opcionalmente this) refere-se ao próprio o worker
```



CAPÍTULO 10

Offline

Neste capítulo, mostraremos as funcionalidades previstas nas especificações da HTML5 denominada Offline Web Application que, como o nome sugere, destina-se a criar aplicações web para funcionamento offline.

10.1 Introdução

A especificação para a criação de aplicações web offline, em sua parte introdutória, diz o seguinte:

Com a finalidade de possibilitar aos usuários a continuar interagindo com uma aplicação ou documento, mesmo que sua conexão com a Internet tenha caído ou esteja indisponível, os autores devem criar um *manifesto* que relaciona os arquivos necessários para o funcionamento offline da aplicação. O navegador do usuário ao tomar conhecimento da existência de um *manifesto* criará uma cópia dos arquivos listados para uso offline.

A especificação prevê o relacionamento de três tipos de arquivos a serem usados na criação de uma aplicação offline e constantes do *manifesto*. São eles:

- arquivos necessários para uso offline;
- arquivos não necessários para uso offline;
- arquivos alternativos para uso offline.

O navegador ou dispositivo do usuário criará uma cópia em cache dos arquivos necessários para uso offline e deles fará uso quando o usuário abrir a aplicação e estiver offline ou quando estando online a conexão cair.

A relação dos arquivos não necessários para uso offline é opcional e destina-se a garantir que as requisições do navegador para os arquivos nela listados ocorram somente quando o usuário estiver online. Por exemplo: em uma aplicação que contenha uma funcionalidade para atendimento online (chat), os arquivos necessários ao funcionamento do chat não devem ser armazenados localmente, pois é óbvio que a funcionalidade não funcionará offline.

Arquivos alternativos são arquivos armazenados em cache para serem usados como alternativa aos arquivos não disponíveis offline. Por exemplo: se o usuário tenta usar o chat offline, é chamado um arquivo alternativo contendo uma mensagem informando que ele precisa estar conectado à Internet para usar o chat.

10.2 Manifesto

Manifesto é um arquivo de texto puro codificado em UTF-8 que informa ao navegador quais são os arquivos necessários e não necessários para uso offline. Um manifesto contém um título e três seções nas quais são listados os três tipos de arquivos estudados no item anterior. Observe a seguir a sintaxe para um arquivo manifesto típico.

```
CACHE MANIFEST
```

```
# 03/04/2011-v1
```

```
CACHE:
```

```
# lista dos arquivos para uso offline  
index.html  
/estilos/main.css  
/scripts/jquery.js
```

```
NETWORK:
```

```
# lista dos arquivos não necessários para uso offline  
/login.php  
http://facebook/api/comments
```

```
FALLBACK:
```

```
# lista dos arquivos alternativos para uso offline  
/login.php /offline/login.html  
http://facebook/api/comments /offline/facebook-comentarios.html  
*.html /offline/offline.html  
# Exemplo de arquivo manifesto  
# para uso em aplicações offline
```

As diretrizes que regem a criação e funcionamento de um arquivo manifest são as seguintes:

- o título CACHE MANIFEST deve ser, obrigatoriamente, a primeira linha do arquivo;
- as três seções CACHE, NETWORK E Fallback são facultativas, podem ser escritas em qualquer ordem e podem constar mais de uma vez no manifesto;
- a seção CACHE pode ser omitida e os arquivos nela listados podem vir logo abaixo do título. Tudo se passa como se CACHE MANIFEST assumisse as funções de título e seção CACHE cumulativamente;
- linhas em branco são ignoradas;
- linhas de comentário devem iniciar com o sinal “jogo da velha” (#);
- a quantidade máxima de dados em cache por site não deve ultrapassar 5MB;
- se o carregamento do arquivo manifesto ou de qualquer um dos arquivos nele contido falhar, todo o processo de armazenamento falha. Caso existam dados anteriormente armazenados em cache, esses dados serão usados para navegação offline.

A seção NETWORK admite duas sintaxes. A primeira sintaxe lista explicitamente os arquivos não necessários para uso offline, como mostrado no exemplo anterior. A segunda sintaxe usa o caractere curinga (*) para informar que todos os arquivos não listados na seção CACHE não devem ser armazenados para uso offline, conforme mostrado a seguir.

NETWORK:

*

Na seção Fallback onde são listados os arquivos alternativos, cada linha da listagem contém dois arquivos separados por um espaço em branco. O primeiro é o arquivo online e o segundo o arquivo alternativo a ser armazenado.

No nosso exemplo a primeira linha daquela seção instrui o navegador em modo offline a usar o arquivo `login.html` armazenado em cache se o usuário tentar acesso ao arquivo `login.php` para se logar offline.

Na terceira linha qualquer acesso offline a um arquivo `.html` que não esteja armazenado para uso offline acessará o arquivo `offline.html` armazenado em cache. Possivelmente esse arquivo conterá uma mensagem ao usuário informando que o acesso à página que ele acaba de requisitar somente é possível quando online.



É evidente, contudo, não é demais ressaltar que todo o processo de armazenamento de arquivos em cache é iniciado a partir de um primeiro acesso online à página que contém o arquivo manifesto.

Um arquivo manifesto pode ser gravado com qualquer extensão, porém é de boa prática não escolher uma extensão reservada a outros tipos de arquivo (por exemplo: .html, .doc, .jpg, .xls) ou, melhor ainda, adote a extensão .manifest que, embora não obrigatória, já está consagrada pelo uso.

Um arquivo manifesto deve ser servido com o MIME type `text/cache-manifest`, assim é necessário que o seu servidor esteja configurado de modo a reconhecer a extensão do arquivo manifesto como pertencente a um documento criado para esse MIME type. Supondo que adotamos a extensão .manifest em um servidor Apache, o arquivo de configuração do Apache ou o arquivo .htaccess deverá ter uma linha, como mostrado a seguir.

```
AddType text/cache-manifest .manifest
```

Já sabemos como criar e qual a finalidade de um arquivo manifesto, vamos estudar a seguir como fazer funcionar o arquivo.

A primeira providência é informar à página web que existe um arquivo manifesto para fazê-la usável offline. Para isso foi criado pela HTML5 o atributo `manifest` para o elemento `html` e a declaração se faz conforme mostrado a seguir.

```
<html manifest="/offline/exemplo.manifest" lang="pt-br">
```

O valor do atributo é um URL relativo ou absoluto no qual se encontra hospedado o arquivo manifesto.

É necessário que o atributo `manifest` esteja presente no elemento `html` da página para que ela seja incluída no cache para uso offline, exceto se a página estiver relacionada explicitamente na seção CACHE do manifesto. Apesar de essa exceção ser uma recomendação da especificação, é de boa prática usar o atributo `manifest` mesmo naquelas páginas.

10.2.1 Atualização do manifesto

Suponha que a sua aplicação foi armazenada para uso offline conforme instruções contidas no arquivo manifesto. Passado algum você modificou, atualizando, um ou mais arquivos que tenham sido armazenados no cache do navegador do usuário. Se o usuário carregar sua aplicação online, os arquivos desatualizados não serão

automaticamente substituídos no cache do navegador e o usuário continuará com uma versão desatualizada quando voltar para a navegação offline.

Esse comportamento é importante, pois, uma vez armazenados em cache os arquivos para uso offline, eles não serão atualizados se não ocorrer uma das seguintes hipóteses:

- o usuário apaga os dados armazenados em cache para a aplicação;
- o desenvolvedor atualiza o arquivo manifesto;
- o desenvolvedor usa o objeto `applicationCache` para verificar o status do cache e atualizá-lo.

A primeira hipótese deve ser imediatamente descartada, pois admitir que os usuários farão limpeza do cache do navegador toda vez que carregarem uma aplicação para uso offline é inadmissível. Então vamos estudar as duas outras hipóteses.

Atualizar o arquivo manifesto é uma opção que tem sido adotada na maioria das vezes. Qualquer alteração feita no arquivo é suficiente para provocar uma atualização dos dados armazenados quando o usuário retorna à aplicação online. Até mesmo a mudança de um comentário no arquivo provoca a atualização. E é exatamente esse tipo de mudança que vem sendo empregado.

Uma opção muito conveniente é criar uma linha de comentário informando a data da última modificação do arquivo ou sua versão ou ainda ambas. Toda vez que atualizarmos arquivos da aplicação alteramos a data ou versão lançada no arquivo manifesto. O exemplo mostrado anteriormente contém a linha de comentário destinada a ser modificada sempre que houver uma atualização na aplicação. Transcrevemos a seguir parte daquele exemplo.

```
CACHE MANIFEST  
# 03/04/2011-v1  
...
```

A última opção é fazer a atualização do cache com uso de script. O script que faz a atualização automática será mostrado em 10.3.3.

10.3 API Application cache

Application cache, uma API prevista nas especificações para HTML5, que define métodos, propriedades e eventos destinados a monitorar e manipular o estado dos

dados armazenados para uma aplicação offline. A interface dessa API fornece um objeto denominado `applicationCache` contendo todas as funcionalidades da API.

A sintaxe para armazenar o objeto em uma variável e suas respectivas funcionalidades é mostrada a seguir.

```
var appCache = window.applicationCache;  
appCache.status  
appCache.update()  
appCache.swapCache()
```

Assim, o objeto `applicationCache` tem uma propriedade e dois métodos estudados a seguir.

10.3.1 Propriedades e métodos

status

Trata-se de uma propriedade destinada a monitorar o estado dos dados em cache podendo assumir os seguintes valores:

- **UNCACHED** (0) – nada existe armazenado em cache.
- **IDLE** (1) – estão sendo usados os últimos dados armazenados em cache.
- **CHECKING** (2) – verificando existência de atualizações para armazenar no cache.
- **DOWNLOADING** (3) – fazendo download de atualizações de dados.
- **UPDATEREADY** (4) – foi feito o download de atualizações e estas estão aguardando que o usuário recarregue a página ou que o método `swapCache()` seja chamado para atualizar o cache.
- **OBSOLETE** (5) – O cache foi marcado como obsoleto. Acontece quando o arquivo manifesto não é encontrado.

Observe a seguir um exemplo da sintaxe de uso dessa propriedade.

```
var appCache = window.applicationCache;  
if (appCache.status == appCache.UPDATEREADY) {  
    // código para atualizar o cache  
}
```

update()

Esse método informa ao navegador que existe um arquivo atualizado. Convém deixar claro que ao se evocar o método o download será feito; contudo, são necessárias outras ações para que o cache seja atualizado, como veremos a seguir. O uso desse método pode ser dispensado, pois a existência de arquivos atualizados e seus downloads são feitos sempre que a página é carregada.

swapCache()

Esse método informa ao navegador que, uma vez feito o download de um manifesto atualizado, os dados deverão substituir o cache atual pelo atualizado. O uso desse método deve seguir-se ao uso do evento `updateready`. Convém notar que apesar de o cache ter sido atualizado seu efeito sobre a página aberta somente será sentido se o usuário recarregar a página ou fechar o navegador e voltar à página.

Um script completo que faz a atualização e a coloca em uso será mostrado adiante, pois nele usaremos um manipulador de eventos e precisamos estudar primeiro os eventos da API.

10.3.2 Eventos

onchecking

Esse atributo, que define o evento `checking`, é disparado quando se constata a existência de atualizações de cache ou a tentativa de download do manifesto pela primeira vez que a página é carregada no navegador. Esse é sempre o primeiro evento da sequência de eventos.

A sintaxe para uso desse atributo é mostrada a seguir.

```
var appCache = window.applicationCache;  
appCache.addEventListener('checking', funcao, false);  
...
```

onupdate

Esse atributo, que define o evento `update`, é disparado quando se constata a não existência de atualizações.

A sintaxe para uso desse atributo é mostrada a seguir.

```
var appCache = window.applicationCache;  
appCache.addEventListener('update', funcao, false);  
...
```

downloading

Esse atributo, que define o evento `downloading`, é disparado quando se constata a existência de atualizações e o navegador está buscando por elas ou no início do download do manifesto pela primeira vez.

A sintaxe para uso desse atributo é mostrada a seguir.

```
var appCache = window.applicationCache;  
    appCache.addEventListener('downloading', funcao, false);  
    ...
```

progress

Esse atributo, que define o evento `progress`, é disparado quando se inicia o download de cada um dos arquivos de atualizações.

A sintaxe para uso desse atributo é mostrada a seguir.

```
var appCache = window.applicationCache;  
    appCache.addEventListener('progress', funcao, false);  
    ...
```

cached

Esse atributo, que define o evento `cached`, é disparado quando atualizações foram baixadas e se encontram em condições de ir para o cache.

A sintaxe para uso desse atributo é mostrada a seguir.

```
var appCache = window.applicationCache;  
    appCache.addEventListener('cached', funcao, false);  
    ...
```

updateready

Esse atributo, que define o evento `updateready`, é disparado quando atualizações foram baixadas, encontram-se em condições de ir para o cache e pode-se chamar o método `swapCache()` para fazer a troca do cache atual pelo cache recém-atualizado.

A sintaxe para uso desse atributo é mostrada a seguir.

```
var appCache = window.applicationCache;  
    appCache.addEventListener('updateready', funcao, false);  
    ...
```

obsolete

Esse atributo, que define o evento `obsolete`, é disparado quando a requisição para o arquivo manifesto resulta em um erro 404 – arquivo não encontrado ou 403 –, arquivo em endereço de acesso proibido.

A sintaxe para uso desse atributo é mostrada a seguir.

```
var appCache = window.applicationCache;  
appCache.addEventListener('obsolete', funcao, false);  
...
```

error

Esse atributo, que define o evento `error`, é disparado quando a requisição para o arquivo manifesto resulta em um dos seguintes erros:

- erro 404 ou 410 – arquivo removido permanentemente;
- a página que solicitou o manifesto não carregou adequadamente;
- ocorreu um erro fatal na busca por arquivos atualizados;
- o manifesto sofreu alteração enquanto estava sendo atualizado no navegador.

A sintaxe para uso desse atributo é mostrada a seguir.

```
var appCache = window.applicationCache;  
appCache.addEventListener('onerror', funcao, false);  
...
```

Ocorrendo um erro qualquer, o processo de requisição falha e a aplicação usa a antiga versão em cache se houver uma.

10.3.3 Atualizando o cache

Vimos que uma das três maneiras de se atualizar o cache é com uso de script. Veremos a seguir um script que, valendo-se dos métodos, propriedades e eventos estudados nos itens anteriores, faz a atualização automática do cache. Na verdade, a atualização imediata depende da autorização do usuário.

```
var appCache = window.applicationCache;  
window.addEventListener('load', function(e) {  
    appCache.addEventListener('updateready', function(e) {  
        if (appCache.status == appCache.UPDATEREADY) {  
            appCache.swapCache();
```

```
if (confirm('Existem atualizações para este site. Deseja carregar as atualizações?')) {
    window.location.reload();
}
} else {
    alert('Você optou por usar arquivos desatualizados neste site.');
}
}, false);
}, false)
```

10.4 Verificando suporte

Nos scripts desenvolvidos para uso dessa API, é importante a verificação de suporte nos navegadores para as funcionalidades descritas. Como vimos em capítulos anteriores, o uso da biblioteca *Modernizr* fornece uma maneira simples de se processar essa verificação. A seguir, mostramos a sintaxe para a verificação com uso da biblioteca citada.

```
if (!Modernizr.applicationcache) {alert('Lamento seu navegador não suporta navegação offline');
ou ainda
if (!window.applicationCache) {alert('Lamento seu navegador não suporta navegação offline');
```

10.5 Exemplo prático

Para demonstrar o uso das funcionalidades oferecidas pela API Application cache, desenvolvemos um exemplo simples que, embora não tenha valor prático algum, presta-se para demonstrar a sintaxe e uso da API.

O exemplo é constituído de três arquivos, a saber:

- **index.html** – uma página com o layout e estilização típicos das páginas-exemplo do livro.
- **offline.css** – uma folha de estilos definindo fundo e cor de fontes para o elemento body.
- **exemplo.manifest** – um arquivo manifesto.

O arquivo manifesto contém instruções para armazenagem da marcação HTML da página *index.html* e do arquivo *offline.css*. Assim, quando o usuário visualizar a página offline, ela será apresentada com todo o seu conteúdo HTML sem estilização em fundo e fontes na cor definida no arquivo *offline.css*.

O exemplo está disponível no site do livro e os códigos são mostrados a seguir.

► index.html

```
<!DOCTYPE html>
<html manifest="exemplo.manifest" lang="pt-br">
<head>
<meta charset="utf-8">
<title>Livro HTML5 do Maujor</title>
<link href="../estilos/main.css" rel="stylesheet" type="text/css">
<!-- code highlighter -->
...
<!-- fm code highlighter -->
<script>
if (!Modernizr.applicationcache) {alert('Lamento, seu navegador não suporta navegação offline');}
var appCache = window.applicationCache;
window.addEventListener('load', function(e) {
    appCache.addEventListener('updateready', function(e) {
        if (appCache.status == appCache.UPDATEREADY) {
            appCache.swapCache();
            if (confirm('Existem atualizações para este site. Deseja carregar as atualizações?')) {
                window.location.reload();
            }
        } else {
            alert('Você optou por usar arquivos desatualizados neste site.');
        }
    }, false);
}, false)
</script>
</head>
<body>
<div id="tudo">
    <header>
        <h1>Capítulo 10</h1>
        <h2>Application cache</h2>
    </header>
    <section>
        ...
    </section>
    <footer>
        ...
    </footer>
</div>
</body>
</html>
```

► offline.css

```
body {  
    background-color:black;  
    color:white;  
}
```

► exemplo.manifest

CACHE MANIFEST

#05/04/2011:v1

CACHE:

index.html
offline.css

NETWORK:

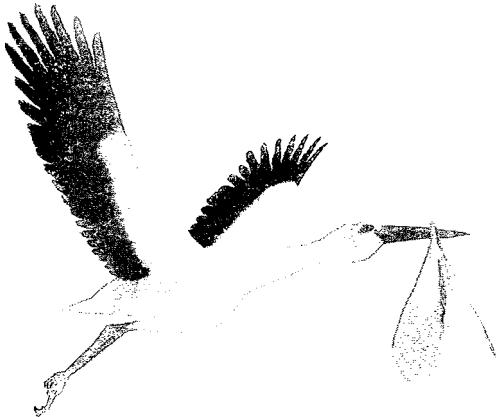
./estilos/main.css

FALLBACK:

./estilos/main.css offline.css;



[c10-index.html]



CAPÍTULO 11

Acessibilidade

Neste capítulo, mostraremos as funcionalidades previstas nas especificações da HTML5 desenvolvidas pelo W3C e destinadas a incrementar a acessibilidade aos conteúdos web. Veremos a WAI-ARIA, sigla para Web Accessibility Initiative – Accessible Rich Internet Applications, que, em tradução adaptada, significa: Generalidades para a acessibilidade ao conteúdo web – Aplicações de internet rica e acessíveis. Estudaremos também o uso de Microdados, um mecanismo para incorporar dados na marcação HTML, que se destinam a ser acessados por máquinas.

11.1 WAI-ARIA

WAI é um Grupo de Trabalho do W3C e tem por objetivo o desenvolvimento de estratégias, diretrizes e fontes de consulta destinadas a prover meios para a criação de uma web acessível para pessoas com necessidades especiais.

ARIA é a sigla para Aplicações de Internet Rica que em definição informal são interfaces gráficas que oferecem possibilidade de interação com o usuário com base em funcionalidades instaladas ao lado do cliente.

O primeiro Rascunho de Trabalho para a especificação ARIA foi lançado pelo W3C em 4 de fevereiro de 2008 e atingiu o estágio de Candidata à Recomendação em 18 de janeiro de 2011. Na especificação está dito o seguinte, em tradução adaptada:

A acessibilidade ao conteúdo web rico, tais como widgets, estruturas em geral e comportamentos dependem da inclusão de informações semânticas a serem interpretadas por tecnologias assistivas, tais como leitores de tela, para que estas possam transmitir ao usuário com necessidades especiais seu real significado.

Esta especificação fornece diretrizes sobre a definição da função, estado e propriedade dos elementos de marcação com a finalidade de incrementar a acessibilidade e interoperabilidade de conteúdos e aplicações web.

Aplicações web complexas podem tornar-se inacessíveis se as tecnologias assistivas não forem capazes de determinar a semântica das estruturas que a compõem ou se o usuário não puder navegar por toda a aplicação de uma forma consistente.

A WAI-ARIA divide a semântica para fins de acessibilidade em:

- *role* para definir a finalidade do elemento na interface;
- *state* propriedade dinâmica que define as características de um objeto sujeito à mutação em resposta a uma ação do usuário ou a um processamento automático;
- *propertie* tem o mesmo significado de *state*. A diferença é que *propertie* define ocorrência de mudança de estado de forma não tão frequente como ocorre com *state*. Contudo, há exceções e na prática é comum referir-se genericamente a *state* e *propertie* como *atributos*.

11.1.1 Role

Para definir a finalidade de um elemento estrutural adicionando-lhe semântica para uso por uma tecnologia assistiva, usamos o atributo *role*. A sintaxe para definição desse atributo é mostrada a seguir.

```
<div id="principal" role="main">  
  ...  
</div>
```

Um desenvolvedor, humano, que entenda a língua portuguesa ao ler a marcação mostrada concluirá que o `div#principal` é um container para o conteúdo principal da aplicação. Não se trata de um menu ou de um rodapé. Lembre-se de que as boas práticas de escrita de marcação recomendam o uso de nomes para ID's e classes que sejam semânticos, isto é, que indiquem a finalidade estrutural do elemento ao qual estão inseridas.

Por outro lado, um leitor de tela ao ler a marcação mostrada encontrará a palavra `principal` como valor do ID e nada concluirá, pois o idioma que uma tecnologia assistiva entende não é uma linguagem humana, e sim uma linguagem de máquina previamente definida em uma taxionomia própria para a linguagem de marcação.

Faz parte daquela taxionomia o atributo role e seu valor main que são interpretados por uma máquina como sendo uma região da aplicação na qual se encontra o conteúdo principal da aplicação. Não se trata de uma região contendo um menu ou um rodapé.

O valor prático, para tecnologias assistivas, de uma marcação como a mostrada é imediato. Suponha o cenário no qual o desenvolvedor prevê na marcação um link do tipo “Pular navegação” para evitar que um leitor de tela tenha de passar por todo o processo de leitura dos itens de um vasto menu antes de chegar ao conteúdo principal. Com o uso das funcionalidades da ARIA, esses tipos de links, tão comuns em documentos atuais, não mais serão necessários, pois a existência de uma informação semântica indicando onde se encontra o conteúdo principal será suficiente para que o leitor de tela encontre aquele conteúdo.

Como foi dito, existe uma gramática destinada à máquina definindo a finalidade e função de cada uma das funcionalidades da ARIA. Vamos a seguir fornecer uma visão geral dessa gramática. A especificação categoriza roles em quatro áreas:

- Abstract roles
- Widget roles
- Document Structure roles
- Landmark roles

11.1.1.1 Abstract roles

Os roles abstratos constituem a base para a construção dos demais roles da WAI-ARIA. São usados exclusivamente para definir a semântica relacionada à natureza dos elementos ou seções da aplicação, ou seja, destinam-se à construção da ontologia dos roles. Assim, não devem ser usados pelos autores para definir a semântica de conteúdos.

A título de informação, os roles abstratos e suas finalidades são descritos a seguir.

- **command** – uma estrutura destinada a executar uma ação, mas que não recebe nenhum tipo de dado.
- **composite** – uma estrutura contendo elementos de navegação ou elementos-filho que a compõem.
- **input** – uma estrutura genérica destinada a receber dados fornecidos pelo usuário.

- **landmark** – uma região da página na qual se supõe necessidade de rápido acesso. Por exemplo, uma estrutura de navegação ou de busca.
- **range** – um controle destinado a receber um dado fornecido pelo usuário e que deve estar compreendido entre determinados valores.
- **roletype** – um role-base do qual todas suas estruturas descendentes herdam a semântica.
- **section** – uma unidade de estrutura de conteúdos renderizável em uma aplicação ou documento.
- **sectionhead** – uma estrutura que serve de rótulo ou sumário para uma seção.
- **select** – uma estrutura que permite ao usuário escolher um valor constante de uma lista de opções.
- **structure** – um elemento estrutural do documento.
- **widget** – um componente interativo de uma interface gráfica.
- **window** – o navegador ou uma janela da aplicação.

11.1.1.2 Widget roles

Os roles para Widgets devem ser usados, pelos autores, para definir a semântica de componentes de um Widget da interface do usuário.

Os valores possíveis para essa categoria de roles são mostrados a seguir.

- **alert** – uma mensagem contendo uma informação importante ao usuário.
- **alertdialog** – uma janela de diálogo contendo uma mensagem de alerta na qual o foco inicial é dado a um elemento específico da janela.
- **button** – um controle no qual o usuário realiza um clique ou pressiona para desencadear uma ação.
- **checkbox** – um controle do tipo **checkbox** com três estados possíveis: **true**, **false** e **mixed**.
- **dialog** – uma janela contendo uma aplicação que interrompe o processamento da aplicação principal e destina-se a solicitar uma informação ou uma resposta ao usuário.

- **gridcell** – uma célula de um grid ou de uma árvore de grid.
- **link** – uma referência interativa a um destino externo ou interno na própria aplicação que, quando ativado, faz com que o agente de usuário navegue para o destino.
- **log** – uma região na qual se armazenam informações essenciais que não são sobrescritas por informações que se atualizam.
- **marquee** – uma região na qual se armazenam informações não essenciais que não são atualizadas com muita frequência.
- **menuitem** – uma das opções de um grupo de escolhas contidas em um menu ou barra de menu.
- **menuitemcheckbox** – um item de menu do tipo `checkbox` com três estados possíveis: `true`, `false` e `mixed`.
- **menuitemradio** – um item de menu do tipo `radiobutton` no qual é possível apenas uma escolha.
- **option** – uma das opções de um elemento de seleção.
- **progressbar** – um elemento destinado a mostrar o progresso de uma tarefa de longa duração.
- **radio** – um controle do tipo `radiobutton` de um grupo no qual é possível apenas uma opção de escolha.
- **scrollbar** – um objeto gráfico destinado a controlar o rolamento de um conteúdo em determinada área de apresentação.
- **slider** – um controle para a seleção de determinado valor contido em uma faixa de valores.
- **spinbutton** – um controle para a seleção de determinado valor contido em uma faixa de valores com incrementação discreta, ou seja, valores em crescimento contínuo.
- **status** – conteúdo para uma mensagem ao usuário cuja importância não justifica o uso de uma janela de alerta. Normalmente, mas não necessariamente, uma barra de status.

Fazem parte ainda dessa categoria os roles destinados a marcar containers de gerenciamento dos Widgets. Os valores possíveis para esses roles são mostrados a seguir.

- **combobox** – uma caixa de seleção, em geral similar a uma caixa de texto na qual o usuário escolhe uma opção ou digita um item de sua escolha.
- **grid** – um controle interativo contendo células de dados tabulares com linhas e colunas semelhante a uma tabela.
- **listbox** – uma estrutura que permite a escolha de um ou mais itens de uma lista de opções.
- **menu** – uma estrutura que oferece opções de escolha ao usuário.
- **menubar** – uma barra de menu normalmente posicionada na horizontal.
- **radiogroup** – um grupo de botões do tipo radio.
- **tablist** – uma lista de abas.
- **tree** – um tipo de lista contendo subníveis aninhados que retraem e expandem.
- **treemap** – um grid no qual as linhas retraem e expandem.

11.1.1.3 Document Structure roles

Os roles para Document Structure devem ser usados, pelos autores, para definir a semântica de componentes estruturais destinados a organizar os conteúdos na página. Normalmente não são usados em estruturas interativas.

Os valores possíveis para essa categoria de roles são mostrados a seguir.

- **article** – uma estrutura da página cujo conteúdo é independente do documento, página ou site.
- **columnheader** – uma célula contendo o cabeçalho da coluna.
- **definition** – uma estrutura contendo a definição de um termo ou conceito.
- **directory** – uma lista de referência aos componentes de um grupo tal como uma tabela índice.
- **document** – uma região contendo informações relacionadas que fazem parte do conteúdo do documento. Contrário à região de uma aplicação (ver role application em Landmark Role).
- **group** – um conjunto de objetos da interface que para as tecnologias assistivas não devem ser incluídos em um índice ou uma tabela sumário da página.

- heading – o cabeçalho de uma seção da página.
- img – um container para uma coleção de objetos que formam uma imagem.
- list – um grupo de itens de uma lista não interativa.
- listitem – um item de uma lista.
- math – um conteúdo representando uma expressão matemática.
- note – uma seção cujo conteúdo é uma anotação sobre o conteúdo principal.
- presentation – um elemento cuja função semântica nativa não está mapeada pela API de acessibilidade.
- region – uma seção da página importante o bastante para ser incluída no índice.
- row – uma linha de células em um grid.
- rowheader – uma célula contendo o cabeçalho da linha.
- separator – um divisor que separa seções de conteúdos ou grupos de itens de menu.
- toolbar – uma coleção de botões apresentada em forma visual compacta.

11.1.1.4 Landmark roles

Os Landmark roles devem ser usados, pelos autores, para definir a semântica de componentes estruturais aos quais se supõe que o usuário deseje um rápido acesso, como estruturas de navegação, de busca, formulários etc.

Os valores possíveis para essa categoria de roles são mostrados a seguir.

- application – região contendo uma aplicação. Contrário à região de um documento (ver role document em Document Structure Role).
- banner – região contendo informação sobre o conteúdo do site como um todo.
- complementary – região destinada a complementar o conteúdo principal com a mesma hierarquia deste no DOM, possuindo significado próprio mesmo quando dissociada do conteúdo principal.
- contentinfo – região contendo informações sobre o documento-pai.

- `form` – um container para controles de formulário.
- `main` – um container para o conteúdo principal da página.
- `navigation` – um container para os elementos de navegação na página.
- `search` – um container para os elementos que criam uma funcionalidade de busca na página.

11.1.2 Atributos

Atributo é a denominação genérica para o componente da marcação que define o estado e as propriedades de um elemento da aplicação. Cada um dos roles apresentados na seção anterior admite um conjunto de estados e propriedades que são a ele aplicáveis. A sintaxe geral para um atributo é `aria-*` na qual o caractere * é um curinga. O exemplo a seguir esclarece o uso e a sintaxe de atributos.

```
<li role="menuitemcheckbox" aria-checked="true">Ordenar por data de modifica&atilde;o</li>
```

Nesse exemplo usou-se o atributo `aria-checked` que define o estado do item de um menu como sendo o item marcado (`checked`) no menu. O valor do atributo foi definido como `true`.

As listas de atributos e propriedades de cada role, bem como suas características gerais e valores possíveis constam do item 5. *The Roles Model* da especificação para a WAI-ARIA. Trata-se de uma lista vasta que optamos por não transcrever neste livro. Recomendamos consultar a especificação no endereço <http://www.w3.org/TR/wai-aria/roles> para extrair as informações sobre atributos.

Para uma ideia geral da apresentação desses atributos, transcrevemos a título de informação a tabela 11.1 contendo informações sobre os atributos para o role `navigation`, conforme consta das especificações.

As superclasses e subclasses constam da taxionomia de roles cujo diagrama pode ser visto em http://www.w3.org/TR/wai-aria/rdf_model.svg.

Os atributos são listados na tabela e aqueles que se referem a estado estão devidamente identificados, os demais são propriedades. Os valores de estado normalmente são `true` e `false` e os valores das propriedades são definidos pelo autor.

Tabela 11.1 – Atributos do role navigation

Características	Valor
Superclasse do role:	landmark
Conceitos relacionados:	elemento nav
Propriedades e estados herdados	aria-atomic aria-busy (estado) aria-controls aria-describedby aria-disabled (estado) aria-dropeffect aria-expanded (estado) aria-flowto aria-grabbed (estado) aria-haspopup aria-hidden (estado) aria-invalid (estado) aria-label aria-labelledby aria-live aria-owns aria-relevant
Nome de:	autor

Observe a seguir um exemplo mostrando a sintaxe e uso de role e atributos.

```
<ul role="menubar">
  <li role="menuitem" aria-haspopup="true" aria-labelledby="arquivo">
    <span id="arquivo">Arquivo</span>
    <ul role="menu">
      <li role="menuitem">Novo</li>
      <li role="menuitem">Abrir...</li>
      ...
    </ul>
  </li>
  ...
</ul>
```

11.1.2.1 Atributos globais

Os seguintes atributos são globais, isto é, podem ser usados para todos os elementos independentemente do seu role.

- **aria-atomic** – indica se a tecnologia assistiva deve apresentar somente a região onde houve uma mudança ou toda a região onde ela ocorreu. Os valores possíveis para esse atributo são `true` e `false`, sendo `false` o valor-padrão. Para esse efeito as notificações de mudança são definidas pela propriedade `aria-relevant` com seus valores `additions`, `removals`, `text` ou `all`.
- **aria-busy** (estado) – indica o estado de um elemento e sua árvore, informando se ele está sendo atualizado ou não. Os valores possíveis para esse atributo são `true` e `false`, sendo `false` o valor-padrão.
- **aria-controls** – identifica um elemento ou elementos cujo conteúdo ou presença na página é controlado pelo elemento. O valor desse atributo é o ID do elemento controlado.
- **aria-describedby** – identifica um elemento ou elementos que descrevem um objeto. O valor desse atributo é o ID do elemento descrito.
- **aria-disabled** (estado) – identifica um elemento desabilitado, ou seja, não operacional. Os valores possíveis para esse atributo são `true` e `false`, sendo `false` o valor-padrão.
- **aria-dropeffect** – indica quais funções podem ser executadas quando um elemento arrastável é solto na área a ele destinada. Os valores possíveis para esse atributo são: `copy`, `move`, `link`, `execute`, `popup` e `none` é o valor-padrão.
- **aria-flowto** – identifica um elemento fora da ordem natural de leitura ou fluxo do documento permitindo que tecnologias assistivas escolham um fluxo de leitura alternativo.
- **aria-grabbed** (estado) – indica o estado de arraste de um elemento. Os valores possíveis para esse atributo são `true`, `false` e `undefined` sendo `undefined` o valor-padrão. O valor `true` indica que o elemento foi seguro e está em condições de ser arrastado, o valor `false` indica que o elemento é arrastável, mas não foi seguro para arraste e o valor `undefined` indica que o elemento não é arrastável.
- **aria-haspopup** – indica que o elemento tem um menu de contexto popup ou subníveis de menu.

- **aria-hidden** (estado) – indica que o elemento e seus descendentes não estão visíveis para nenhum usuário conforme definido pelo autor. Os valores possíveis para esse atributo são `true` e `false`, sendo `false` o valor-padrão.
- **aria-invalid** (estado) – indica que o valor fornecido não está de acordo com o formato esperado pela aplicação. Os valores possíveis para esse atributo são `grammar`, `spelling`, `true` e `false`, sendo `false` o valor-padrão. O valor `grammar` indica um erro gramatical, `spelling` em erro de grafia, `true` um erro de validação e `false` a não existência de erros.
- **aria-label** – indica um rótulo para o elemento. O valor desse atributo é uma string.
- **aria-labelledby** – identifica o elemento que serve de rótulo para o elemento corrente. O valor desse atributo é o ID do elemento que está sendo rotulado.
- **aria-live** – indica que um elemento será atualizado e define o tipo de atualização. Os valores possíveis para esse atributo são `polite` e `assertive`. O valor `polite` instrui as tecnologias assistivas a informar o usuário sobre uma atualização tão logo ela tenha sido completada. A atualização se faz no background sem interferir com o fluxo normal da interação. O valor `assertive` é usado quando a ocorrência de uma atualização altera o andamento natural de uma tarefa e o usuário precisa ser notificado tão logo a atualização inicie.
- **aria-owns** – identifica um elemento ou elementos com a finalidade de definir o relacionamento visual, funcional ou de contexto entre elementos-pai e elementos-filho quando o DOM não puder ser usado para estabelecer o relacionamento. O valor desse atributo é o ID do elemento que está sendo identificado.
- **aria-relevant** – indica quais são as notificações de mudanças ocorridas em uma região marcada com `aria-live` serão enviadas para as tecnologias assistivas. Os valores possíveis para essa propriedade são: `additions`, `removals`, `text` ou `all` conforme o envio seja para atualizações devidas, adição de conteúdos, remoção, inclusão de textos sem alterar o DOM ou todas as alterações.

11.2 Microdados

A especificação do W3C para Microdados encontra-se na fase de Rascunho de Trabalho, porém as funcionalidades dessa tecnologia, em sua maioria, são bem suportadas pelos navegadores e clientes para os quais foram criadas. Na especificação está dito o seguinte, em tradução adaptada:

Esta especificação define o mecanismo Microdados para a HTML. Este mecanismo permite que se incorporem dados na marcação HTML de uma maneira fácil e que se destinam a serem lidos por máquinas. Microdados é compatível com vários outros formatos de dados inclusive RDF e JSON.

Assim, Microdados destina-se a adicionar semântica aos conteúdos web para serem lidos por máquinas, como mecanismos de busca, web crawlers, navegadores etc., com a finalidade de enriquecer a experiência do usuário. Microdados fornece às máquinas informações que as permite melhor entender e interpretar os conteúdos web. Por exemplo: facilitando a scripts fornecer serviços personalizados por página ou possibilitando que conteúdos gerados por diferentes autores possam ser processados de uma maneira consistente.

11.2.1 Sintaxe

O termo *item* é usado para designar uma estrutura de dados a serem definidos com uso dos mecanismos de Microdados.

Um *vocabulário* para Microdados é um documento que define a semântica de *itens*. Ao desenvolvedor é facultado criar um vocabulário próprio para contemplar suas aplicações ou usar um dos vocabulários existentes disponíveis na web.

Um exemplo mostrando vários vocabulários, suportados pelo Google, está disponível em <http://data-vocabulary.org>. Lá encontramos vocabulários para adicionar semântica a conteúdos que dizem respeito a eventos, organizações, pessoas, produtos, revisões etc.

Os autores são encorajados a, sempre que possível, usar um vocabulário já existente.

Observe na tabela 11.2 o vocabulário para pessoas disponível em <http://www.data-vocabulary.org/Person/> e devidamente traduzido.

Tabela 11.2 – Exemplo de vocabulário para pessoas

Propriedade	Descrição
<code>name (fn)</code>	Nome da pessoa
<code>nickname</code>	Apelido da pessoa
<code>photo</code>	Link para uma imagem da pessoa
<code>title</code>	Título da pessoa (por exemplo: Gerente Financeiro)
<code>role</code>	Função da pessoa (por exemplo: Contador)
<code>url</code>	Link para uma página, tal como a home page pessoal

Propriedade	Descrição
affiliation (org)	Nome da organização a qual a pessoa é associada (por exemplo: empregado)
friend	Nome de outra pessoa de seu relacionamento social como amigo
contact	Nome de outra pessoa de seu relacionamento social como contato
acquaintance	Nome de outra pessoa de seu relacionamento social como conhecido
address (adr)	O local da pessoa. As subpropriedades são: street-address, locality, region, postal-code e country-name

Os nomes das propriedades entre parênteses são conforme a grafia definida para Microformatos.

Observe a seguir uma marcação HTML destinada a descrever uma pessoa.

<div>

Meu nome é Maurício Samy Silva, mas sou conhecido como "Maujor". Meu site é:

www.maujor.com.

Eu moro no Rio de Janeiro, RJ e sou um engenheiro civil da PLANGEN.

Meus amigos são:

João,

Maria

</div>

A seguir, a mesma marcação mostrada anteriormente com a inclusão dos mecanismos de Microdados.

<div itemscope itemtype="http://data-vocabulary.org/Person">

Meu nome é Maurício Samy Silva, mas sou conhecido como "Maujor".

Meu site é: www.maujor.com.

Eu moro no

Rio de Janeiro, RJ

e sou um engenheiro civil da
PLANGEN

Meus amigos são:

João,

Maria

</div>

A marcação mostrada resume a sintaxe geral para inclusão dos atributos de Microformato na marcação HTML, conforme segue.

- O atributo `itemscope` (sem declaração de valor) define um container para um *item*. A esse atributo segue-se o atributo `itemtype` cujo valor é um URL apontando para o vocabulário a ser adotado para a marcação dos dados no item.
- O atributo `itemprop` define a propriedade do dado marcado, conforme consta no vocabulário definido em `itemtype`.
- É válido aninhar `itemscope` e, em consequência, `itemtype`, como foi feito no exemplo mostrado anteriormente. Notar que aninhamos o vocabulário para *Address* dentro do vocabulário para *Person*.

11.2.1.1 Atributos globais

São previstos os seguintes atributos globais para um *item*.

- `itemscope` – é um atributo booleano, isto é, não admite valor (não confundir com atributo de valor booleano). Quando presente, declara o estado como verdadeiro; quando ausente (ou não definido), como falso. Destina-se a marcar um container no qual seus conteúdos são marcados com as funcionalidades de Microdados.
- `itemtype` – o valor desse atributo é um URL apontando para um arquivo contendo o vocabulário adotado para a marcação dos Microformatos.
- `itemid` – define um identificador global para o *item*.
- `itemprop` – define a propriedade de um item de acordo com os valores especificados no vocabulário adotado.
- `itemref` – define um ID de um elemento no qual se encontram informações adicionais sobre o item.

Observe no exemplo a seguir um exemplo de uso da sintaxe dos atributos globais.

```
<div itemscope itemid="amanda" itemref="a b" itemtype="http://exemplo.com/banda/></div>
<p id="a">Nome: <span itemprop="name">Amanda</span></p>
<div id="b" itemprop="band" itemscope itemref="c"></div>
<div id="c">
    <p>Banda: <span itemprop="name">Jazz Band</span></p>
    <p>Componentes: <span itemprop="size">12</span> músicos</p>
</div>
```

11.3 Microdados DOM API

A API para Microdados prevê um método e nove propriedades, conforme descritas a seguir.

`document.getItems([tipo])`

Esse método retorna uma lista dos nós do DOM (elementos) do documento que criam um *item*. Da lista não constam os nós que criam *itens* aninhados em outro *item*. O método admite um parâmetro facultativo de uma lista de tipos de *item* separados por vírgula. Os valores da lista de parâmetros são os URL designados no atributo `itemtype`.

`elemento.properties`

Para os elementos que possuem o atributo `itemscope` essa propriedade retorna um objeto contendo uma coleção das propriedades do elemento.

`elemento.itemValue[=valor]`

Essa propriedade retorna o valor do atributo `itemprop`. Pode ser ainda usada para definir por script o valor do atributo `itemprop`.

`elemento.itemScope`

Essa propriedade retorna um valor booleano conforme tenha ou não sido declarado o atributo `itemscope` para o elemento.

`elemento.itemType`

Essa propriedade retorna uma string com o valor do atributo `itemtype`.

`elemento.itemId`

Essa propriedade retorna uma string com o valor do atributo `itemid`.

`elemento.itemRef`

Essa propriedade retorna uma string com o valor do atributo `itemref`.

`elemento.itemProp`

Essa propriedade retorna uma string com o valor do atributo `itemprop`.

elemento.properties

Essa propriedade retorna uma string com o valor do atributo `itemtype`.

Os exemplos a seguir esclarecem a sintaxe de uso das propriedades mostradas.

```
if (element.itemProp.contains('cor'))  
    count += 1;
```

No exemplo mostrado, o script verifica se determinado elemento possui o atributo `itemprop` com o valor `cor`. Caso positivo, incrementa um contador.

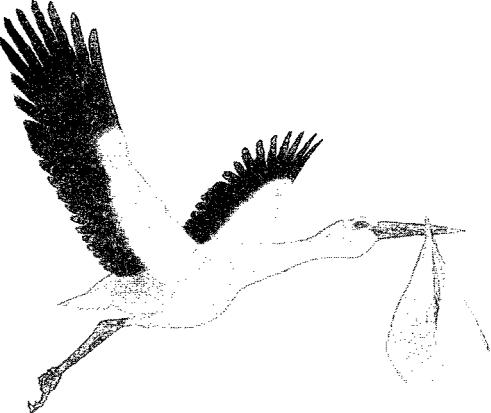
```
for (var index = 0; index < element.itemRef.length; index += 1)  
    process(document.getElementById(element.itemRef[index]));
```

No exemplo mostrado, o script percorre todos os elementos que contêm o atributo `itemref` e define um processamento para cada um deles.

11.3.1 Suporte nos navegadores

Atualmente nenhum navegador oferece suporte para as funcionalidades dessa API. A biblioteca Modernizr não prevê a verificação de suporte para essa API. Uma função simples para fazer a verificação do suporte é mostrada a seguir.

```
function suportaMicrodados() {  
    return !!document.getItems;  
};  
if (suportaMicrodados()) {  
    // script para navegadores que suportam  
} else {  
    // mensagem ou script para navegadores que não suportam  
}
```



APÊNDICE A

Elementos da HTML5

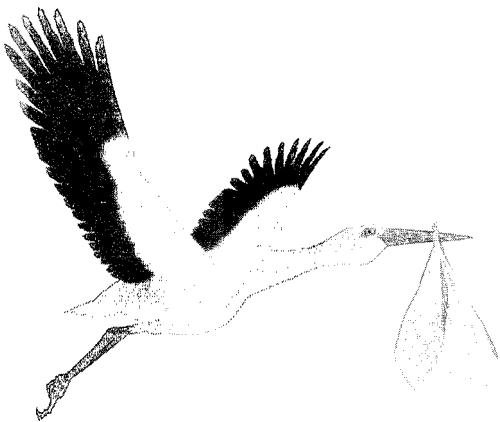
Esta tabela foi extraída do site do W3C e relaciona os elementos da HTML5. Fornece ainda uma breve descrição do elemento, além de relacionar os atributos a eles aplicáveis.

Elemento	Descrição	Atributos
a	Hyperlink	globais; href; target; rel; media; hreflang; type
abbr	Abreviação	globais
address	Informações de contato para uma página ou seção	globais
area	Área em um mapa de imagem com ou sem hyperlink	globais; alt; coords; shape; href; target; rel; media; hreflang; type
article	Conteúdo passível de sindicalização (distribuição independente) ou conteúdo reusável	globais
aside	Conteúdo tangencialmente relacionado a outro conteúdo	globais
audio	Player de áudio	globais; src; preload; autoplay; loop; controls
b	Marca palavras-chave	globais
base	URL base para hyperlinks	globais; href; target
bdi	Marca de direção de texto	globais
bdo	Formata direção de texto	globais
blockquote	Citação longa	globais; cite
body	Corpo do documento	globais; onafterprint; onbeforeprint; onbeforeunload; onblur; onerror; onfocus; onhashchange; onload; onmessage; onoffline; ononline; onpagehide; onpageshow; onpopstate; onredo; onresize; onscroll; onstorage; onundo; onunload
br	Quebra de linha	globais
button	Controle tipo botão	globais; autofocus; disabled; form; formaction; formenctype; formmethod; formnovalidate; formtarget; name; type; value

Elemento	Descrição	Atributos
canvas	Área de bitmap para scripts	globais; width; height
caption	Legenda de tabela	globais
cite	Título de um trabalho	globais
code	Código máquina	globais
col	Coluna de tabela	globais; span
colgroup	Grupamento de colunas de uma tabela	globais; span
command	Comando de menu	globais; type; label; icon; disabled; checked; radiogroup
datalist	Container para opções de um controle tipo combo box	globais
dd	Conteúdo para correspondente elemento(s) dt	globais
del	Conteúdo removido do documento	globais; cite; datetime
details	Controle para apresentar detalhes	globais; open
dfn	Instância de definição	globais
div	Container genérico	globais
dl	Container para listas de definição	globais
dt	Legenda conteúdos em elemento(s) dd	globais
em	Forte ênfase	globais
embed	Plugin	globais; src; type; width; height; any*
fieldset	Grapamento de controles de formulário	globais; disabled; form; name
figcaption	Legenda de figuras	globais
figure	Figura com legenda opcional	globais
footer	Rodapé para uma página ou seção	globais
form	Formulário	globais; accept-charset; action; autocomplete; enctype; method; name; novalidate; target
h1, h2, h3, h4, h5, h6	Cabeçalho de uma seção	globais
head	Container para metadados do documento	globais
header	Conteúdo introdutório ou de auxílio de navegação para uma página ou seção	globais
hgroup	Grupamento de cabeçalhos	globais
hr	Quebra temática de linha	globais
html	Elemento raiz	globais; manifest
i	Alteração de entonação	globais
iframe	Contexto de navegador aninhado	globais; src; srcdoc; name; sandbox; seamless; width; height
img	Imagem	globais; alt; src; usemap; ismap; width; height
input	Controle de formulário	globais; accept; alt; autocomplete; autofocus; checked; dirname; disabled; form; formaction; formenctype; formmethod; formnovalidate; formtarget; height; list; max; maxlength; min; multiple; name; pattern; placeholder; readonly; required; size; src; step; type; value; width

Elemento	Descrição	Atributos
ins	Acréscimo no documento	globais; cite; datetime
kbd	Entrada de usuário	globais
keygen	Controle de formulário gerador de chave criptográfica	globais; autofocus; challenge; disabled; form; keytype; name
label	Legenda para controle de formulário	globais; form; for
legend	Legenda para fieldset	globais
li	Item de uma lista	globais; value*
link	Metadado link	globais; href; rel; media; hreflang; type; sizes
map	Mapa de imagem	globais; name
mark	Destaque	globais
menu	Menu de comandos	globais; type; label
meta	Metadado texto	globais; name; http-equiv; content; charset
meter	Medidor	globais; value; min; max; low; high; optimum; form
nav	Seção contendo links de navegação	globais
noscript	Conteúdo alternativo para script	globais
object	Imagem, contexto de navegador aninhado ou plugin	globais; data; type; name; usemap; form; width; height
ol	Lista ordenada	globais; reversed; start
optgroup	Grupamento de opções em um controle list box	globais; disabled; label
option	Opção em um controle list box ou combo box	globais; disabled; label; selected; value
output	Container para um valor calculado	globais; for; form; name
p	Parágrafo	globais
param	Parâmetro para o elemento object	globais; name; value
pre	Bloco de texto pré-formatado	globais
progress	Barra de progresso	globais; value; max; form
q	Citação curta	globais; cite
rp	Parênteses para anotação ruby em texto	globais
rt	Anotação ruby em texto	globais
ruby	Anotação ruby	globais
s	Texto incorreto	globais
samp	Saída genérica de computador	globais
script	Script incorporado no documento	globais; src; async; defer; type; charset
section	Documento ou seção genérica de uma aplicação.	globais
select	Controle do tipo list box	globais; autofocus; disabled; form; multiple; name; required; size
small	Comentário à parte	globais
source	Mídia source para vídeo ou áudio	globais; src; type; media
span	Container genérico	globais
strong	Importância	globais

Elemento	Descrição	Atributos
style	Informa sobre estilização incorporada no documento	globais; media; type; scoped
sub	Subescrito	globais
summary	Legenda para o elemento details	globais
sup	Superescrito	globais
table	Tabela	globais; summary
tbody	Grupo de linhas em uma tabela	globais
td	Célula de tabela	globais; colspan; rowspan; headers
textarea	Campo de texto multilinhas	globais; autofocus; cols; disabled; form; maxlength; name; placeholder; readonly; required; rows; wrap
tfoot	Grupo de linhas de rodapé de uma tabela	globais
th	Célula de cabeçalho de uma tabela	globais; colspan; rowspan; headers; scope
thead	Grupo de células de cabeçalho de uma tabela	globais
time	Data e/ou data-hora	globais; datetime; pubdate
title	Título do documento	globais
tr	Linha de tabela	globais
track	Trilha de texto	globais; default; kind; label; src; srclang
ul	Container para lista	globais
var	Variável	globais
video	Player de vídeo	globais; src; poster; preload; autoplay; loop; controls; width; height
wbr	Quebra de linha	globais



APÊNDICE B

Atributos da HTML5

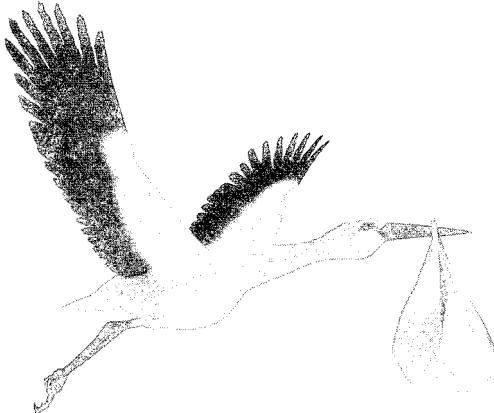
Esta tabela foi extraída do site do W3C e relaciona os atributos da HTML5. Fornece ainda uma relação dos elementos a que se aplicam, além de uma breve descrição do atributo.

Atributo	Elemento(s)	Descrição
accept	input	Dica para o tipo de arquivo esperado por um controle tipo upload
accept-charset	form	Codificação de caracteres usados nos dados enviados por um formulário
accesskey	elementos HTML	Atalho de teclado para dar o foco a um elemento
action	form	URL para envio e processamento dos dados de um formulário
alt	area; img; input	Teste alternativo para descrição de imagens quando não disponíveis
async	script	Execução assíncrona de script
autocomplete	form; input	Controla o autocompletamento em controles de formulário
autofocus	button; input; keygen; select; textarea	Dá o foco automaticamente quando a página carrega
autoplay	audio; video	Controla o início da execução de mídia quando a página é carregada
challenge	keygen	String a ser anexada ao pacote gerado para uma chave pública
charset	meta	Declaração de codificação de caracteres
charset	script	Declaração de codificação de caracteres de um script
checked	command; input	Marcar um comando ou controle
cite	blockquote; del; ins; q	Link para a fonte da citação ou informações adicionais sobre a citação
class	elementos HTML	Classe à qual o elemento pertence
cols	textarea	Número máximo de caracteres por linha
colspan	td; th	Número de colunas em uma célula
content	meta	Valor do elemento
contenteditable	elementos HTML	Define conteúdo editável de um elemento
contextmenu	elementos HTML	Menu de contexto
controls	audio; video	Controle de mídia nativo do agente de usuário
coords	area	Coordenadas de uma forma de um mapa de imagens
data	object	Endereço do recurso

Atributo	Elemento(s)	Descrição
datetime	del; ins	Data e opcionalmente hora de uma modificação
datetime	time	Valor do elemento
default	track	Usa track default se não houver um disponível
defer	script	Adia a execução do script
dir	elementos HTML	Direção do texto de um elemento
dirname	input; textarea	Define a direção do texto de um controle de formulário para ser enviada quando o formulário é submetido ao processamento
disabled	button; command; fieldset; input; keygen; optgroup; option; select; textarea	Define um controle de formulário desabilitado
draggable	elementos HTML	Define a condição de arraste de um elemento
dropzone	elementos HTML	Tipo de itens aceitos para drag-and-drop
enctype	form	Tipo de codificação de dados a ser usada no envio de um formulário para processamento
for	label	Associa um rótulo a um controle de formulário
for	output	Especifica o controle usado para o cálculo do valor de saída.
form	button; fieldset; input; keygen; label; meter; object; output; progress; select; textarea	Associa um controle a um formulário
formaction	button; input	URL para envio do formulário para processamento
formenctype	button; input	Tipo de codificação de dados a ser usada no envio de um formulário para processamento
formmethod	button; input	Método HTTP usado no envio do formulário para processamento
formnovalidate	button; input	Bypassa a validação do formulário
formtarget	button; input	Contexto do navegador para envio do formulário
headers	td; th	Células de cabeçalho para uma célula
height	canvas; embed; iframe; img; input; object; video	Dimensão vertical – altura
hidden	elementos HTML	Define a relevância do elemento
high	meter	Valor alto em uma faixa de valores
href	a; area	Endereço de um hiperlink
href	link	Endereço de um hiperlink
href	base	URL base do documento
hreflang	a; area; link	Idioma do destino do link
http-equiv	meta	Diretiva pragma
icon	command	Ícone para um comando
id	elementos HTML	ID de um elemento
ismap	img	É um mapa de imagem no lado do servidor
keytype	keygen	Tipo de chave criptográfica a gerar
kind	track	Tipo da trilha
label	command; menu; optgroup; option; track	Rótulo
lang	elementos HTML	Idioma do conteúdo do elemento
list	input	Lista de autocompletamento
loop	audio; video	Define mídia executada continuamente

Atributo	Elemento(s)	Descrição
low	meter	Valor baixo em uma faixa de valores
manifest	html	Manifesto para cache local
max	input	Valor máximo
max	meter; progress	Límite superior de uma faixa de valores
maxlength	input; textarea	Maior comprimento de um valor
media	a; area; link; source; style	Define a mídia aplicável
method	form	Método HTTP para envio de formulário
min	input	Valor mínimo
min	meter	Límite inferior de uma faixa de valores
multiple	input; select	Define aceitação de múltiplos valores
name	button; fieldset; input; keygen; output; select; textarea	Nome de controles de formulário a usar em form.elements API
name	form	Nome de formulário a usar em document.forms API
name	iframe; object	Nome de contextos de navegador aninhados
name	map	Nome de um mapa de imagem referenciado com o atributo usemap
name	meta	Nome de matadado
name	param	Nome de parâmetro
novalidate	form	Bypassa a validação de um controle de formulário
open	details	Define a visibilidade do elemento details
optimum	meter	Valor ótimo em um widget de medição
pattern	input	Expressão regular a casar com um valor de um controle de formulário
placeholder	input; textarea	Rótulo a ser inserido em um controle de formulário
poster	video	Frame inicial mostrado no início de um vídeo
preload	audio; video	Indicação da quantidade de buffer necessária para uma mídia
pubdate	time	Data da publicação de um artigo ou de uma página
radiogroup	command	Nome do grupo de comandos relacionado a um grupo de botões radio
readonly	input; textarea	Define que o valor é somente para leitura e não pode ser editado pelo usuário
rel	a; area; link	Relacionamento existente entre o documento que contém o link e o alvo do link
required	input; select; textarea	Define um controle de formulário de preenchimento obrigatório
reversed	ol	Numeração dos itens de listas em ordem inversa
rows	textarea	Número de linhas visíveis
rowspan	td; th	Número de linhas contidas em uma célula
	iframe	Regras de segurança para conteúdos aninhados
spellcheck	elementos HTML	Define correção gramatical para o conteúdo de um elemento
scope	th	Define a quais células uma célula cabeçalho se refere
scoped	style	Define o escopo de aplicação de estilos
seamless	iframe	Define aplicação de estilo em conteúdo aninhado no documento
selected	option	Define seleção padrão para uma opção
shape	area	O tipo de forma de um mapa de imagem
size	input; select	Tamanho do controle de formulário

Atributo	Elemento(s)	Descrição
sizes	link	Tamanho dos ícones (para rel="icon")
span	col; colgroup	Número de colunas no elemento
src	audio; embed; iframe; img; input; script; source; track; video	Endereço do alvo
srcdoc	iframe	Documento a ser renderizado em um iframe
srclang	track	Idioma da trilha da mídia
start	ol	Valor ordinal do primeiro item da lista
step	input	Intervalo granular da medida de um controle
style	elementos HTML	Instruções de apresentação e formatação
summary	table	Texto explicativo da estrutura de uma tabela complexa
tabindex	elementos HTML	Ordem sequencial de foco no elemento
target	a; area	Contexto de navegador para destino de hiperlink
target	base	Contexto de navegador padrão para destino de hiperlinks e envio de formulário
target	form	Contexto de navegador padrão para destino de envio de formulário
title	elementos HTML	Informação complementar sobre o elemento
title	abbr; dfn	Texto completo de uma abreviatura
title	command	Dica descritiva de um comando
title	link	Título de um link
title	link; style	Nome de uma folha de estilo alternativa
type	a; area; link	Dica para o tipo do alvo referenciado
type	button	Controle tipo botão
type	button; input	Tipo do controle de formulário
type	command	Tipo de comando
type	embed; object; script; source; style	Tipo do recurso incorporado
type	menu	Tipo de menu
usemap	img; object	Nome do mapa de imagem a usar
value	button; option	Valor a ser usado no envio do formulário
value	input	Valor do controle de formulário
value	li	Valor ordinal de um item de lista
value	meter; progress	Valor corrente do controle
value	param	Valor do parâmetro
width	canvas; embed; iframe; img; input; object; video	Dimensão horizontal – largura
wrap	textarea	Modo de quebra de linhas para o valor do controle



APÊNDICE C

Atributos para eventos da HTML5

Esta tabela foi extraída do site do W3C e relaciona os atributos para eventos da HTML5. Fornece ainda uma relação dos elementos a que se aplicam, além de uma breve descrição do atributo.

Atributo	Elemento(s)	Descrição
onabort	elementos HTML	Manipulador do evento abort
onafterprint	body	Manipulador do evento afterprint para o objeto Window
onbeforeprint	body	Manipulador do evento beforeprint para o objeto Window
onbeforeunload	body	Manipulador do evento beforeunload para o objeto Window
onblur	body	Manipulador do evento blur para o objeto Window
onblur	elementos HTML	Manipulador do evento blur
oncanplay	elementos HTML	Manipulador de evento canplay
oncanplaythrough	elementos HTML	Manipulador de evento canplaythrough
onchange	elementos HTML	Manipulador de evento change
onclick	elementos HTML	Manipulador de evento click
oncontextmenu	elementos HTML	Manipulador de evento contextmenu
oncuechange	elementos HTML	Manipulador de evento cuechange
ondblclick	elementos HTML	Manipulador de evento dblclick
ondrag	elementos HTML	Manipulador de evento drag
ondragend	elementos HTML	Manipulador de evento dragend
ondragenter	elementos HTML	Manipulador de evento dragenter
ondragleave	elementos HTML	Manipulador de evento dragleave
ondragover	elementos HTML	Manipulador de evento dragover
ondragstart	elementos HTML	Manipulador de evento dragstart
ondrop	elementos HTML	Manipulador de evento drop
ondurationchange	elementos HTML	Manipulador de evento durationchange
onemptied	elementos HTML	Manipulador de evento emptied
onended	elementos HTML	Manipulador de evento ended

Atributo	Elemento(s)	Descrição
onerror	body	Manipulador do evento error para o objeto Window e Manipulador para notificações de erros em script
onerror	elementos HTML	Manipulador de evento error
onfocus	body	Manipulador do evento focus para o objeto Window
onfocus	elementos HTML	Manipulador de evento focus
onhashchange	body	Manipulador do evento hashchange para o objeto Window
oninput	elementos HTML	Manipulador de evento input
oninvalid	elementos HTML	Manipulador de evento invalid
onkeydown	elementos HTML	Manipulador de evento keydown
onkeypress	elementos HTML	Manipulador de evento keypress
onkeyup	elementos HTML	Manipulador de evento keyup
onload	body	Manipulador do evento load para o objeto Window
onload	elementos HTML	Manipulador de evento load
onloadeddata	elementos HTML	Manipulador de evento loadeddata
onloadedmetadata	elementos HTML	Manipulador de evento loadedmetadata
onloadstart	elementos HTML	Manipulador de evento loadstart
onmessage	body	Manipulador do evento message para o objeto Window
onmousedown	elementos HTML	Manipulador de evento mousedown
onmousemove	elementos HTML	Manipulador de evento mousemove
onmouseout	elementos HTML	Manipulador de evento mouseout
onmouseover	elementos HTML	Manipulador de evento mouseover
onmouseup	elementos HTML	Manipulador de evento mouseup
onmousewheel	elementos HTML	Manipulador de evento mousewheel
onoffline	body	Manipulador do evento offline para o objeto Window
ononline	body	Manipulador do evento online para o objeto Window
onpagehide	body	Manipulador do evento pagehide para o objeto Window
onpageshow	body	Manipulador do evento pageshow para o objeto Window
onpause	elementos HTML	Manipulador de evento pause
onplay	elementos HTML	Manipulador de evento play
onplaying	elementos HTML	Manipulador de evento playing
onpopstate	body	Manipulador do evento popstate para o objeto Window
onprogress	elementos HTML	Manipulador de evento progress
onratechange	elementos HTML	Manipulador de evento ratechange
onreadystatechange	elementos HTML	Manipulador de evento readystatechange
onredo	body	Manipulador do evento redo para o objeto Window
onreset	elementos HTML	Manipulador de evento reset
onresize	body	Manipulador do evento resize para o objeto Window
onscroll	body	Manipulador do evento scroll para o objeto Window
onscroll	elementos HTML	Manipulador de evento scroll
onseeked	elementos HTML	Manipulador de evento seeked
onseeking	elementos HTML	Manipulador de evento seeking
onselect	elementos HTML	Manipulador de evento select
onshow	elementos HTML	Manipulador de evento show
onstalled	elementos HTML	Manipulador de evento stalled
onstorage	body	Manipulador do evento storage para o objeto Window
onsubmit	elementos HTML	Manipulador de evento submit

Atributo	Elemento(s)	Descrição
onsuspend	elementos HTML	Manipulador de evento suspend
ontimeupdate	elementos HTML	Manipulador de evento timeupdate
onundo	body	Manipulador do evento undo para o objeto Window
onunload	body	Manipulador do evento unload para o objeto Window
onvolumechange	elementos HTML	Manipulador de evento volumechange
onwaiting	elementos HTML	Manipulador de evento waiting



Referências

- KEITH, Jeremy. *DOM Scripting: Web Design with JavaScript and the Document Object Model*. New York: Friends of ED, 2005.
- KEITH, Jeremy. *HTML5 for Web Designers*. New York: Jeffrey Zeldman, 2010.
- KOCH, Peter-Paul. *Ppk on JavaScript*. Berkeley: NewRiders, 2007.
- LAWSON, Bruce; SHARP, Remy. *Introducing HTML5*. Berkeley: NewRiders – Voices That Matter, 2011.
- Mozilla Developer Network. Disponível em: <<https://developer.mozilla.org/en-US/>>.
- PILGRIM, Mark. *HTML5: Up and Running*. 1 ed. Sebastopol: O'Reilly, 2010.
- W3C. Disponível em: <<http://www.w3.org>>.
- WHATWG. Disponível em: <<http://www.whatwg.org/>>.

A

Alerta 16
API de canvas
 addColorStop() 154
 arc() 150
 arcTo() 149
 beginPath() 148
 clearRect() 146
 closePath() 152
 createLinearGradient() 154
 createPattern() 156
 createRadialGradient() 154
 drawImage() 164, 165
 fill() 151
 fillRect() 145
 fillStyle 145
 fillText() 167
 font 166
 globalAlpha 153
 globalCompositeOperation 157
 lineCap 153
 lineJoin 153
 lineTo() 148
 lineWidth 145
 moveTo() 148
 rect() 152
 rotate() 163
 scale() 161
 shadow 147
 shadowBlur 147
 shadowColor 147
 shadowOffset 147
 shadowOffsetX 147
 shadowOffsetY 147
 stroke() 148
 strokeRect() 144
 strokeStyle 144
 strokeText() 167
 textAlign 168
 textBaseline 169
 translate() 162
API do Google Static Maps 224
 center 225
 format 225
 language 226
 maptype 226

markers 226
mobile 226
path 227
sensor 228
size 225
visible 227
zoom 225
aria-* 294
ARIA 287
 Abstract roles 289
 composite 289
 input 289
 landmark 290
 propertie 288
 range 290
 role 288
 roletype 290
 section 290
 sectionhead 290
 select 290
 state 288
 structure 290
 widget 290
 window 290
Arquivos para download 18
Atributos globais 56
 accesskey 56
 class 57
 contenteditable 57
 contextmenu 58
 dir 59
 draggable 59
 hidden 60
 id 60
 item 61
 lang 61
 spellcheck 61
 style 62
 tabinde 62
 title 62
Atributos globais da HTML5 171
 accesskey 171
 class 173
 contenteditable 173
 contexmenu 174
 dir 174
 draggable 175

dropzone 176
hidden 176
id 177
itemid 178
itemprop 178
itemref 178
itemscope 178
itemtype 178
lang 178
spellcheck 178
style 179
tabindex 179
title 180

Atributos globais para roles 296
aria-atomic 296
aria-busy 296
aria-controls 296
aria-describedby 296
aria-disabled 296
aria-dropeffect 296
aria-flowto 296
aria-grabbed 296
aria-haspopup 296
aria-hidden 297
aria-invalid 297
aria-label 297
aria-labelledby 297
aria-live 297
aria-owns 297
aria-relevant 297

Atributos para API de mídia 115
autoplay 119
buffered 117
controls 119
currentSrc 116
currentTime 120
defaultPlaybackRate 118
duration 121
ended 118
endTime 121
error 115
loop 120
muted 120
networkState 116
paused 118
playbackRate 118
played 118

poster 119
preload 117
readyState 117
seekable 119
seeking 119
startTime 120
tracks 119
videoHeight 120
videoWidth 120
volume 120
width 120

Atributos para formulários
autocomplete 199
autofocus 198
dirname 200
form 201
formaction 202
formenctype 202
formmethod 202
formnovalidate 203
formtarget 203
list 204
min 204
novalidate 205
placeholder 198
required 198

Audio 84
autoplay 86
codec 85
controls 87
loop 86
media 85
pause() 87
play() 87
preload 86
source 85
src 84
type 85
volume 87

C

Canvas 140
contexto bidimensional 142
contexto tridimensional 143
height 141
width 141

CERN 22, 23

Chris Lilley 25
Convenções tipográficas 16
Cougar 25, 26

D

Dan Connolly 23
Dave Raggett 22
Destaques em geral 18
Document Structure roles 292
 article 292
 columnheader 292
 definition 292
 directory 292
 document 292
 group 292
 heading 293
 img 293
 list 293
 listitem 293
 math 293
 note 293
 presentation 293
 region 293
 row 293
 rowheader 293
 separator 293
 toolbar 293

E

Eventos para API de mídia 124
 abort 125
 canplay 126
 canplaythrough 126
 durationchange 127
 emptied 125
 ended 127
 error 125
 loadeddata 126
 loadedmetadata 125
 loadstart 124
 pause 125
 play 125
 playing 126
 progress 125
 ratechange 127
 seeked 126
 seeking 126

stalled 125
suspend 125
timeupdate 126
volumechange 127
waiting 126

F

frames 25

H

height
 Atributos para API de mídia 120
hiperlinks 21
hipertexto 20
HTML+ 22
HTML 2.0 23
HTML ERB 25

I
IETF 24

J
Jim Clark 24

L
Landmark roles 293
 application 293
 banner 293
 complementary 293
 contentinfo 293
 form 294
 main 294
 navigation 294
 search 294
Lynx 23

M
Manifest
 Application cache 279
 Atualização do manifesto 278
 cached 282
 CACHE MANIFEST 277
 downloading 282
 error 283
 FALLBACK 277
 manifest 278
 NETWORK 277

noupdate 281
obsolete 283
oncheking 281
progress 282
status 280
swapCache() 281
update() 281
updateready 282
Manifesto 276
Marcação 17
Marc Andreessen 24
Métodos para API de mídia 123
 addTrack() 124
 canPlayType() 123
 load() 123
 pause() 123
 play() 123
Métodos para Geolocation 217
 getCurrentPosition() 217
 watchPosition() 223
Microdados 287, 297
 document.getItems() 301
 element.itemId 301
 element.itemProp 301
 element.itemRef 301
 element.itemScope 301
 element.itemType 301
 element.itemValue[=value] 301
 element.properties 301, 302
item 298
itemid 300
itemprop 300
itemref 300
itemscope 300
itemtype 300
vocabulário para 298
Mosaic 23

N

NCSA 23
Netscape 24
Novos atributos da HTML5 180
 async 181
 autocomplete 181
 autofocus 181
 autoplay 182
 challenge 182

controls 182
default 182
dirname 183
form 183
formaction 183
formenctype 183
formmethod 184
formnovalidate 184
formtarget 184
icon 185
keytype 185
kind 185
list 186
loop 186
low 187
manifest 187
max 187, 204
min 188
novalidate 188
open 188
optimum 188
pattern 188
ping 189
placeholder 189
poster 190
preload 190
pubdate 190
radiogroup 190
required 191
reversed 191
sandbox 192
scoped 193
seamless 193
sizes 194
srcdoc 194
srclang 195
step 195
wrap 195

Novos elementos 62
 article 62
 aside 63
 audio 64
 canvas 64
 command 64
 datalist 67
 details 68
 figcaption 70

figure 70
footer 71
header 73
hgroup 74
keygen 75
mark 75
menu 65
meter 76
nav 77
output 78
progress 80
ruby 81
section 81
source 82
summary 69
time 82
track 83
video 83

S

SGML 22
Site do livro 19
style 62

T

Terminologia 16
Tim Berners-Lee 21
title 62
Track 99
 captions 99
 chapters 99
 charset 99
 descriptions 99
Firefogg 101
HandBrake 109
kind 99
label 100
metadata 99
srclang 100
subtitles 99
WebSRT 99
WebVTT 100
type 205
 color 213
 date 209
 datetime 209
 datetime-local 211

email 208
month 210
number 212
range 212
search 205
tel 206
time 210
url 207
week 211

V

Video 91
 aspect ratio 98
 audio 97
 auto 95
 autoplay 95
 codecs 94
 controls 96
 H.264 93
 height 98
 loop 95
 media 94
 metadata 95
 mp4 93
 muted 97
 ogv 93
 poster 96
 preload 95
 source 94
 src 95
 type 94
 WebM 93
 width 98

W

W3C 24
WAI-ARIA 287
Web Messaging 252
 message 254
 política da mesma origem 252
window.postMessage 253
WebSocket 260
 bufferedAmout 261
Eventos 261
onclose 262
onerror 263
onmessage 262

onopen 261
readyState 261
send() 260
Web Storage 233
 AJAX Massive Storage System 235
 Armazenagem com JSON 241
 Banco de dados 246
 clear() 240
 cookies 234
 Evento storage 244
 ey() 241
 Gears 235
 getItem() 240
 length 241
 localStorage 237
 LSO - Local Shared Object 234
 removeItem() 240
 sessionStorage 235
 setItem() 239
 userData Behavior 234
Web Workers 263
 close() 274
 error 271
 importScripts 273
 message 269
 multi-thread 264
 new Worker() 268
 postMessage() 269
 single-thread 264
 terminate() 274
WHATWG 26
Widget roles 290
 alertdialog 290
 button 290
 checkbox 290
 combobox 292
 dialog 290
 grid 292
 gridcell 291
 link 291
 listbox 292
 log 291
 marquee 291
 menu 292
 menubar 292
 menuitem 291
 menuitemcheckbox 291
 menuitemradio 291
 option 291
 progressbar 291
 radio 291
 radiogroup 292
 scrollbar 291
 slider 291
 spinbutton 291
 status 291
 tablist 292
 tree 292
 treegrid 292
WWW-talk 22